CASCON 2020 Proceedings

Sponsored By
IBM Centre for Advanced Studies
IBM Canada Lab

Edited By
Lily Shaddick - IBM Canada Ltd.
Guy-Vincent Jourdan – University of Ottawa
Vio Onut - IBM Canada Ltd.
Tinny Ng - IBM Canada Ltd.

Toronto, Ontario, Canada November 10 - November 13, 2020

Table of Contents

Workshops

AI & Blockchain & Robotics

Cloud Computing

Systems & Innovations

IoT & Smart Cities

Quantum Computing

Message from the Conference Chair

CASCON x EVOKE 2020

Message from the Conference General Chair

Welcome to CASCON x EVOKE 2020!

Happy 30th Anniversary!

For the past 30 years, IBM Centre for Advanced Studies (CAS) has been hosting the Annual International Conference on Computer Science and Software Engineering. This conference is a testimony to our commitment to Academia and Applied Research in Canada. In 2019 we joined forces with the EVOKE conference, which created a unique Industry-Academic conference. Through this partnership, we can bring together the worlds of academia, research, development, and every industry for a four-day marathon to discuss research and technology, exciting challenges, achievements and success stories. We continue this journey in 2020 and beyond.

Our conference follows all the academic rigour of selecting and screaming its content, including academic talks, industry talks, workshops, networking, and the Expo. In 2020 we have four keynote speakers, 26 academic talks, 64 industry talks, 29 workshops, and 47 expo presentations contributed by 143 university researchers and 180 industry professionals. As with previous years, the proceedings, including the technical papers, position papers, and detailed workshop abstracts, are also available online in the ACM Digital Library.

Whether we are coming from industry or academia, we all have a common goal and interest in technology. Technology breakthroughs are what we strive for, the technology that can make a difference, that is innovative, unique, solve unsolved problems and establishes us as leaders.

This year's theme, VISION, UNITY, INNOVATION, couldn't be timelier. With the new world-wide pandemic crisis that hit hard on industry and academia, the need for innovation is even more acute. With this new challenge, the society had to adapt to social distancing and new virtual environments that will continue to be the real norm for at least a while. While CASCON x EVOKE participants will not be able to participate in person this year, we see an opportunity to create a higher digital footprint for our conference, to reach an audience across Canada and the world that is a click of a button away from us. Looking at 2021 and beyond, we believe that a hybrid approach will become the norm for our conference. We know our community values an in-person event. Unfortunately, we can not offer that this year; however, we are committed to providing alternative virtual networking sessions and an engaging online environment that will foster serendipitous and spontaneous technical discussions.

We packed full a four-day schedule with six parallel tracks on eight main technological themes: Cloud Computing; Everything Data, IoT & Smart Cities; Security & Privacy; Quantum; Systems & Innovations; Compilers, Languages, Runtimes; AI; and Software Engineering. I am confident that you will find the technical content exciting and engaging.

None of these would be possible without our dedicated community of academics, IBMers and partners. As Conference General Chair, I am fortunate to be immersed in an exceptional team of professionals that make that vision come true. I want to start by thanking the Canada Lab Director, Mr. Steven Astorino, for his thought leadership and his aspiration to create one of Canada's best industry-academic conference. The partnership with the EVOKE conference is now in his second year and growing stronger because of him. Equally important is the support and leadership of Mr. Marcellus Mindel, Head of IBM Canada Advanced Studies, whose understanding and direction significantly impact the conference.

A robust academic conference relies upon the expertise and guidance of a Steering Committee. Big thank you to the CASCON Steering Committee members (Prof. Guy-Vincent Jourdan, Prof. Hausi Müller, Mr. Joe Wigglesworth, Prof. Ken Wong, Prof. Kenneth Kent, Mr. Marcellus Mindel, Prof. Marin Litoiu, Dr. Robert Enenkel, and Mrs. Tinny Ng)

Prof. Julia Rubin, University of British Columbia, acted this year as Conference Program Chair. She has positively impacted the content this year, working tirelessly to orchestrate the paper submissions, revisions, and paper awards for our conference. A big thank you to the 86 Program Committee members who diligently peer-reviewed the papers and selected the top candidates.

Prof. Guy-Vincent Jourdan, Publication Chair, and Ms. Lily Shaddick, Conference Proceedings Editor, diligently took care of our proceedings and ensured that all content was filtered, approved and published in the ACM Library.

For the first time in our conference history, we have the Industry track shaped by Chairs. Mr. Joe Wigglesworth and Dr. Michael Kwok raised to the challenge and formed the Industry Talks Agenda. Special thanks go to the EVOKE Foundation team for their invaluable expertise, enabling us to have a solid Industry Track: Mr. Patrick Kasebzarif, Executive Producer, Ms. Loren Amaral, Creative Lead, Mr. Matthew Di Liddo, Program Lead and Mr. Andrew Kelly, Partnership Lead.

For the third year in a row, we have the privilege of having Mrs. Tinny Ng, IBM, joined this year by Prof. Ken Wong from the University of Alberta acting as Workshop Co-Chairs. I want to thank them both for preparing a rich program consisting of top workshops. I extend this thank you note to the Workshop Selection Committee members for making sure that the best workshops are accepted.

CASCON Technology Expo is the collaboration hub of the conference. With 47 technical exhibits and new content that is changing daily. Accomplished under the leadership of our Expo co-chairs Prof. J. Nelson Amaral from the University of Alberta and Dr. Kit Barton, IBM.

Special thanks to our IBM CAS Canada Team for all the heavy lifting that goes behind the scenes and often is unnoticed but without which nothing is possible: Mr. Dennis Buttera, Mrs. Jennifer Collins, Ms. Maria Gallaher, Mrs. Tinny Ng, and for our exceptional group of interns Ms. Aysha Anwar, Ms. Maxine Arbez Cheung, Mr. Sandy Bagga, Mr. Ali Hosny Hamdy, Mr. Gursehaj Harika, Mr. Alexander Mah, Ms. Alix Mailhot, Ms. Tima Pakfetrat, Ms. Maria Katrina Ronquillo, Ms. Lily Shaddick, and Mr. Kevin Yu.

I want to thank all the volunteers and Prof. Marin Litoiu (Volunteer Chair) for all the conference support. Big thank you to our Virtual Platform Admin team who made the virtual experience possible: Ms. Katina Kelly, Mr. Matthew Luzius, Mr. Chris Kale, Ms. Christine Gokool, Ms. Corey Gray, Ms. Diane Beauvais, Mr. Madni Ahmed, Mr. Michael Keillor, Ms. Sonia Singh, Mr. Ali Hosny Hamdy, Mr. Gursehaj Harika, Mr. Kevin Yu, Ms. Aysha Anwar and Ms. Lily Shaddick.

Finally, I would personally like to thank all the persons that submitted content to our conference and all our CAS Collaborators for promoting and contributing to this event. Finally, **a big thank you to all CASCONxEVOKE participants** for all the idea exchanges and thoughtful discussions during the conference.

I wish you all a wonderful and productive time at CASCONxEVOKE 2020!

Iosif-Viorel (Vio) Onuț, Ph.D.,

Conference General Chair | CASCON 2020
Principal R&D Strategist | Centre for Advanced Studies | IBM Canada Lab
Adjunct Professor | University of Ottawa

Message from the Program Chair

CASCON x EVOKE 2020

Message from the Program Chair

CASCON x EVOKE 2020

Welcome to CASCON x EVOKE 2020, the 30th Annual International Conference on Computer Science and Software Engineering hosted by the IBM Centre for Advanced Studies (CAS) and EVOKE Foundation!

The theme of CASCON x EVOKE 2020 is VISION, UNITY, INNOVATION. This year we received a total of 65 technical paper submissions. Each paper was rigorously reviewed by at least three members of the Program Committee. In the end, the Program Committee members decided to accept 26 papers (40% acceptance rate). The Program Committee also selected the papers that received the Best Paper and the Best Student Paper awards. The CASCON x EVOKE 2020 Best Paper Award goes to authors Britta Sennewald, Rainer Herpers, Marco Hülsmann, and Kenneth B. Kent for their paper, *Voting For Authorship Attribution Applied To Dark Web Data*. The Best Student Paper Award goes to student author Sanjena Krishnakumar for the paper, *Towards Interpretable And Maintainable Supervised Learning Using Shapley Values In Arrhythmia*, co-authored with supervisor Tamer Abdou.

We are extremely happy to have four fantastic keynote speakers, Andrew Pelling from the University of Ottawa; Alexandre Blais, form the Universitaire de Sherbrooke; Nicolas Papernot from the University of Toronto; and Niina Haiminen from T.J. Watson IBM Research Lab, USA, who will talk about cutting-edge work in biomaterial, quantum computing, security, and computational biology. Thank you for your thought-provoking talks!

The program of the conference is organized into eight tracks: Cloud Computing; Everything Data, IoT & Smart Cities; Security & Privacy; Quantum; Systems & Innovations; Compilers, Languages, Runtimes; AI; and Software Engineering. As in previous years of CASCON, the CASCON x EVOKE 2020 proceedings are archived in the ACM Digital Library for ease of access.

One highlight of the conference planning process is the selection of the Most Influential Paper, which is awarded to a paper published a decade earlier at CASCON, in order to recognize the lasting contributions and impact of such paper to theory and practice. Selecting the Most Influential Paper is a process that takes into account several factors. These factors include the impact the paper and its corresponding research had in the subject area, the evolution and significance of the topics discussed in the paper during the past decade, and the consequent work spawned by the paper.

The CASCON x EVOKE 2020 Most Influential Paper of 2010 was selected by the MIP Selection Committee, consisting of Ettore Merlo, École Polytechnique de Montréal; Joe Wigglesworth, IBM Canada; Hausi Muller, University of Victoria; Kostas Kontogiannis, Western University; Robert Enenkel, IBM Canada; and Julia Rubin, University of British Columbia (chair).

The committee followed a selection process similar to that established in previous years of CASCON: first, a list of CASCON 2010 papers, with their citation counts, types of citations, related work conducted during the past decade, and evolution and significance of the areas each, were collected and six papers were short-listed. Each committee member reviewed the short-listed papers and then the members conferred to discuss and debate each candidate paper.

After the detailed discussion, the committee selected the Most Influential Paper for this year, which was awarded to the paper “Improving Program Navigation With an Active Help System” by Petcharat Viriyakattiyaporn and Gail C. Murphy. The committee considered the paper visionary and paving the way for other researchers to work in the area of recommender systems. This work also provided foundations for future DevOps and AI-Ops tools. I would like to congratulate the authors for their outstanding contribution and thank the MIP Award Committee for their work reviewing and deliberating candidates for the award.

I am also immensely grateful to the many people who helped and supported us in organizing CASCON x EVOKE 2020. I would like to thank all the authors of technical papers and the hard-working members of the Program Committee for their dedication to excellence in completing the reviews and engaging in online discussion of the submissions. A special thank you goes to the CASCON x EVOKE 2020 organizing team, including Vio Onut, the general chair of the conference; Tinny Ng and Ken Wong, who coordinated the workshop selection; Kit Barton and J. Nelson Amaral, who orchestrated the technology expo selection; Joe Wigglesworth and Michael Kwok, who coordinated the industry talks; Guy-Vincent Jourdan, who assembled the proceedings; Marin Litoiu who organized the student volunteers; Lily Shaddick, who was the publication lead; and Tinny Ng, who kept the conference website up-to-date. Finally, I would like to thank the CASCON Steering Committee for their valuable support towards compiling this year’s program.

Even though CASCON x EVOKE 2020 is a fully online event this year, we plan on plenty of inspiring discussions, networking events, and interactions. I wish you a wonderful experience at the conference.

Welcome to CASCON x EVOKE 2020!

Julia Rubin
The University of British Columbia, Vancouver, Canada
CASCON x EVOKE | 2020 Program Chair

Organizing Committee

CASCON x EVOKE 2020

Organizing Committee

Conference Chair

Iosif Viorel Onut — IBM Canada Ltd.

Conference Program Chair

Julia Rubin — University of British Columbia

Industry Talks Co-Chairs

Joe Wigglesworth — IBM Canada Ltd.

Michael Kwok — IBM Canada Ltd.

Workshops Co-Chairs

Ken Wong — University of Alberta

Tinny Ng — IBM Canada Ltd.

Exhibits Co-Chairs

Kit Barton — IBM Canada Ltd.

J. Nelson Amaral — University of Alberta

Finance and Registration Chair

Marcellus Mindel — IBM Canada Ltd.

Website Chair

Tinny Ng — IBM Canada Ltd.

Volunteer Chair

Marin Litoiu — York University

Publication Chair

Guy-Vincent Jourdan — University of Ottawa

Conference Proceedings Editor

Lily Shaddick — IBM Canada Ltd.

Executive Producer

Patrick Kasebzarif	Evoke Canada

Program Lead

Matthew Di Liddo	Evoke Canada

Partnership Lead

Andrew Kelly	Evoke Canada

Creative Co-Lead

Loren Amaral	Evoke Canada
Sandy Bagga	IBM Canada Ltd.

Steering Committee

Marin Litoiu	York University
Marcellus Mindel	IBM Canada Ltd.
Hausi Müller	University of Victoria
Tinny Ng	IBM Canada Ltd.
Iosif Viorel Onut	IBM Canada Ltd.
Joe Wigglesworth	IBM Canada Ltd.
Ken Wong	University of Alberta
Robert Enenkel	IBM Canada Ltd.
Guy-Vincent Jourdan Kenneth Kent	University of Ottawa University of New Brunswick

Program Committee

Julia Rubin	University of British Columbia, Program Chair
Jose Nelson Amaral	University of Alberta
Christopher K. Anand Giuliano Antoniol	McMaster University École Polytechnique de Montréal

Akramul Azim	University of Ontario Institute of Technology
Ebrahim Bagheri	Ryerson University
Ayse Bener	Ryerson University
Jeremy Bradbury	University of Ontario Institute of Technology
Paula Branco	University of Ottawa
David Bremner	University of New Brunswick
Sebastian Carbajales	IBM Canada Ltd
Allen Chan	IBM Canada Ltd
Yee-Kang Chang	IBM Canada Ltd
Alexander Chatzigeorgiou	University of Macedonia
Marsha Chechik	University of Toronto
Tse-Hsun Peter Chen	Concordia University
Mark Chignell	University of Toronto
Andrew Craik	IBM Canada Ltd
Eyal De Lara	University of Toronto
Renato De Mori	McGill University
Thomas Dean	Queen's University
Frank Dehne	Carleton University
Chen Ding	University of Rochester
Juergen Dingel	Queen's University
Gerhard Dueck	University of New Brunswick
Ghizlane El Boussaidi	École de technologie supérieure
Robert Enenkel	IBM Canada Ltd
Marios Fokaefs	École Polytechnique de Montréal
James Green	Carleton University
Hadi Hemmati	University of Calgary
Reid Holmes	University of British Columbia
Daqing Hou	Clarkson University
Guy-Vincent Jourdan	University of Ottawa
Wolfram Kahl	McMaster University
Foutse Khomh	École Polytechnique de Montréal
Kostas Kontogiannis	University of Western Ontario
Diwakar Krishnamurthy	University of Calgary
Michael Kwok	IBM Canada Ltd

Alexei Lapouchnian	University of Toronto
Diana Lau	IBM Canada Ltd
Timothy Lethbridge	University of Ottawa
Jin Li	PointClickCare
Sam Lightstone	IBM Canada Ltd
Ramiro Liscano	University of Ontario Institute of Technology
Marin Litoiu	York University
Hanan Lutfiyya	University of Western Ontario
Kelly Lyons	University of Toronto
Nazim Madhavji	University of Western Ontario
Daryl Maier	IBM Canada Ltd
Ettore Merlo	École Polytechnique de Montréal
Piotr Mierzejewski	IBM Canada Ltd
James Miller	University of Alberta
Andriy Miranskyy	Ryerson University
Marc Moreno Maza	University of Western Ontario
Hausi Müller	University of Victoria
John Mylopoulos	University of Toronto
V. Krishna Nandivada	IIT Madras
Maleknaz Nayebi	École Polytechnique de Montréal
Manos Papagelis	York University
Panos Patros	University of Waikato
Fred Popowich	Simon Fraser University
Shaikh Quader	IBM Canada Ltd
Suprio Ray	University of New Brunswick
Tony Renaud	IBM Canada Ltd
Juergen Rilling	Concordia University
Chanchal K. Roy	University of Saskatchewan
Mohammad Sadoghi	University of California
Ken Salem	University of Waterloo
Vivek Sarkar	Georgia Institute of Technology
Mohammed Sayagh	Queen's University
Jun Shirako	Georgia Institute of Technology
Michael Smit	Dalhousie University

Ulrike Stege	University of Victoria
Mark Stoodley	IBM Canada Ltd
Vijay Sundaresan	IBM Canada Ltd
Jaroslaw Szlichta	University of Ontario Institute of Technology
Ladan Tahvildari	University of Waterloo
Alex Thomo	University of Victoria
Whitney Tsang	IBM Canada Ltd
Norha M. Villegas	Universidad Icesi
Paul Ward	University of Waterloo
Joe Wigglesworth	IBM Canada Ltd
Murray Woodside	Carleton University
Morteza Zihayat	Ryerson University
Farhana Zulkernine	Queen's University
Calisto Zuzarte	IBM Canada Ltd

MIP Selection Committee

Ettore Merlo	École Polytechnique de Montréal
Hausi Muller	University of Victoria
Joe Wigglesworth	IBM Canada Ltd.
Julia Rubin	University of British Columbia
Robert Enenkel	IBM Canada Ltd.
Kostas Kontogiannis	Western University

Most Influential Paper of 2010

CASCON x EVOKE 2020

Improving Program Navigation with an Active Help System

Petcharat Viriyakattiyaporn and Gail C. Murphy
Department of Computer Science
University of British Columbia

Abstract

When performing software change tasks, software developers spend a substantial amount of their time navigating dependencies in the code. Despite the availability of numerous tools to aid such naviga-tion, there is evidence to suggest that developers are not using these tools. In this paper, we intro-duce an active help system, called Spyglass, that suggests tools to aid program navigation as a devel-oper works. We report on the results of a laboratory study that investigated two questions: will develop-ers act upon suggestions from an active help sys-tem and will those suggestions improve developer behaviour? We found that with Spyglass we could make developers as aware of navigational tools as they are when requested to read a tutorial about such tools, with less up-front effort. We also found that we could improve developer behaviour as de-velopers in the Spyglass group, after being given recommendations in the context of their work, nav-igated programming artifacts more efficiently than those in the tutorial group.

Find the full paper at https://dl.acm.org/doi/10.1145/1923947.1923951

Full Papers

CASCON x EVOKE 2020

Multiple Pedestrian Tracking Based on Modified Mask R-CNN and Enhanced Particle Filter using an Adaptive Information Driven Motion Model

Mufleh Al-Shatnawi
Amir Asif
mufleh@eecs.yorku.ca
asif@eecs.yorku.ca
York University
Toronto, Ontario, Canada

Vida Movahedi
Aijun An
vida@eecs.yorku.ca
aan@eecs.yorku.ca
York University
Toronto, Ontario, Canada

Yonggang Hu
Junfeng Liu
yhu@ca.ibm.com
jfliu@ca.ibm.com
IBM Canada Ltd
Markham, Ontario, Canada

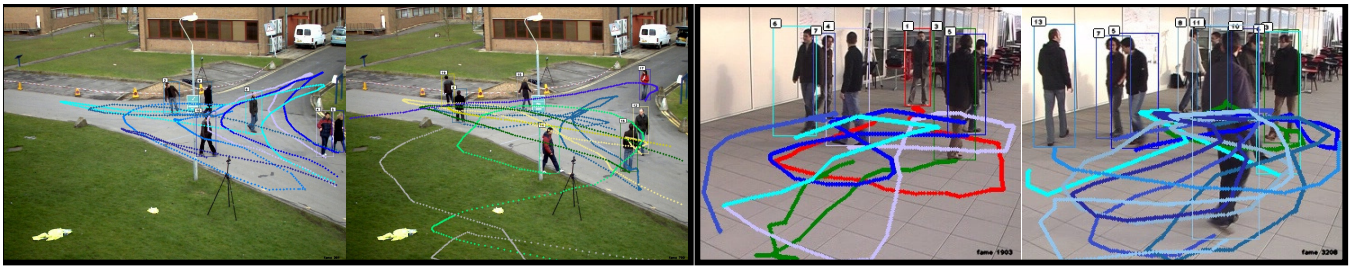


Figure 1: The motion of some pedestrians from PETS2009S2L1-View1 [13] and EPFL-terrace [14] datasets where the trajectory for each pedestrian marked by different color. This figure shows that the motion of pedestrians is highly dynamic, as they are often stopping, moving backward, or turning in circles.

ABSTRACT

In the recent years, multiple pedestrian tracking (MPT) has been one of the most important components in a wide range of applications in computer vision, such as video surveillance, traffic monitoring, and sports analysis, to name a few. In these applications, the scene is in continuous motion hence typical tracking systems that are using background modeling and handcrafted features fail to detect pedestrians efficiently. Furthermore, the scene in these applications shifts between random and continuous pedestrian motion. Most of the existing MPT algorithms based on particle filters assume that the motion of pedestrians is mostly or piecewise linear and predictable. Hence, these tracking algorithms adopt a linear constant velocity motion model for pedestrian tracking. However, the motion of some pedestrians is highly dynamic, as they are often stopping, moving backward, or turning around in real-world surveillance video. To overcome these problems, we propose an approach for multiple pedestrian tracking that can be divided into two main components: detection and tracking. For the detection component, we combine novel post-processing steps with the Mask Region Convolutional Neural Network (Mask R-CNN) to identify multiple pedestrians in a given video frame. For the tracking component, we propose a robust MPT algorithm using enhanced particle filtering with an adaptive information driven motion model and resampling scheme. The proposed tracking algorithm is suitable for online and real-time applications. Since data association is a key issue in tracking-by-detection schemes, we propose a combination between an efficient adaptive information driven motion model and a new resampling scheme that retains information pertaining to the weighted particles during the particle propagation and resampling steps. Experimental results show the benefits of using the proposed post-processing steps and the adaptive information driven motion model for detecting and tracking pedestrians with unpredictable movements. Moreover, the tracking accuracy and precision are significantly improved, and the number of tracker identification (ID) switches is reduced simultaneously.

CASCON'20, November 10–13, 2020, Toronto, Canada

CCS CONCEPTS

• **Computing methodologies** → **Tracking**; **Object detection**; **Object identification**.

KEYWORDS

Multiple pedestrian tracking, particle filter, tracking-by-detection, data association, resampling.

ACM Reference Format:
Mufleh Al-Shatnawi, Amir Asif, Vida Movahedi, Aijun An, Yonggang Hu, and Junfeng Liu. 2020. Multiple Pedestrian Tracking Based on Modified Mask R-CNN and Enhanced Particle Filter using an Adaptive Information Driven Motion Model. In *CASCON'20 Novem-ber 10–13, 2020, Toronto, ON.* , 9 pages.

1 INTRODUCTION

Multiple Pedestrian tracking (MPT) system is an important component in a wide range of applications in computer vision, such as video surveillance, traffic monitoring, and sports analysis, to name a few. Visually, it is quite easy and intuitive for humans to see other humans, recognize or track their actions. However, designing and building an automatic MPT system without any human intervention is a challenging task. There are many sources of uncertainty that effect MPT systems, such as *irregular pedestrian motion*, clutter, changing backgrounds, significant occlusions, and pedestrians being identical in their appearance.

In MPT systems, tracking-by-detection is regarded as a most popular tracking paradigm wherein the tracking performance is dependent on the detection quality. Despite efforts to generate accurate and reliable pedestrian detections, it is still a challenging task for researchers to develop a perfect Multiple Pedestrian Detector (MPD). Normally, MPDs produce both a bounding box and confidence score for each detected pedestrian in a given video frame. The confidence score represents the confidence level of the detector in affirming that the object enclosed by the bounding box is a person/pedestrian.

The traditional approach for pedestrian detection is based on background-subtraction [3, 8, 23, 33]. In this approach, pedestrians are detected in every frame by segmenting the moving objects out of the background, while taking into account pixel-wise time consistency. However, the background-subtraction methods are unreliable and error-prone in noisy video sequences. For instance, the background-subtraction methods detect all moving objects in the scene even these that are not pedestrians [3, 8, 23, 33]. In recent years, multiple pedestrian detection (MPD) methods have been developed either by using a deep Convolutional Neural Network (CNN), or by building a specific pedestrian detector added to these networks [18, 19, 28, 39, 40]. These CNN MPD methods are able to learn discriminative features directly from raw pixels of an image, and they are producing a confidence score between zero and one for the detected pedestrians. Hence, these methods have notable performance gains over the background-subtraction methods, and they normally provide a high detection accuracy [40].

Given the initial state (e.g., position and size) of a target pedestrian in the reference video frame, the objective of a tracking algorithm is to build a posterior probability distribution for the state of the tracked target using noisy detections (observations). Although many tracking algorithms have been developed over the years, the particle filter (PF) [17] based MPT approaches [6, 15, 16, 24, 29, 31, 34, 36, 38] have shown more promise. The PF operates on the principle of approximating the posterior state distribution by a set of weighted samples, also referred to as particles [4]. Traditional PF approaches suffer from the degeneracy problem [17], wherein after a few iterations, except for a few particles all the others have negligible weights. This problem is overcome using the resampling procedure, which represents the posterior by a new set of particles [4, 9, 22, 25, 30]. Consequently, the PF has also been termed as the sequential importance resampling (SIR) filter.

(a) (b) (c)

Figure 2: Three consecutive frames: (a) Frame 708**, (b) Frame** 709**, and (c) Frame** 710 **as taken from the MOT17-05 video sequence [27]. The KDNT [39] detector detects the same pedestrian with three different confidence scores in successive frames.**

2 RELATED WORK

2.1 CNN For MPD Methods

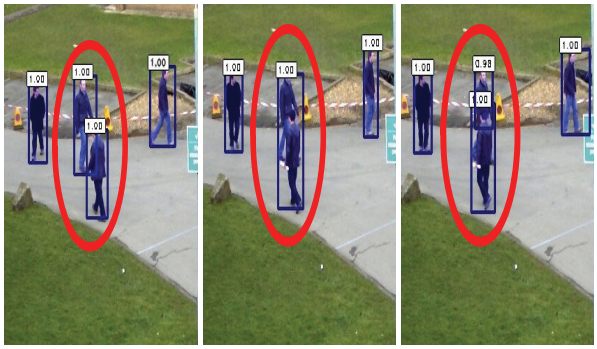
In recent years, multiple pedestrian detection (MPD) methods have been developed either by using a deep Convolutional Neural Network (CNN), or by building a specific pedestrian detector added to these networks [18, 19, 28, 39, 40]. In [28], a pedestrian detector is proposed by using the Faster Region Convolutional Neural Network (Faster-RCNN). The Faster-RCNN can be represented as an end-to-end framework that consists of two sub-CNN networks. The first network extracts features and proposes regions for the second network which in turns classifies the object in the proposed relevant regions. The Faster-RCNN parameters are shared between these two networks and constitute an efficient framework for object detection in general. Furthermore, the Faster R-CNN can be viewed as a CNN based MPD without using any hand-crafted features. The confidence scores of the reported pedestrian detections were between 0.05 and 1.0. In [39], another MPD based approach is developed by using a combination of an additional convolutional neural network and the Faster R-CNN [28]. The additional network is used to calculate the appearance descriptor value for each detected bounding box. The calculated value is then used to determine the data association metric for later stages. The confidence scores of the reported pedestrian detections were between 0.0990 and 0.9998.

In [18], a flexible and efficient framework for instance segmentation and object detection is developed using Mask R-CNN. The Mask R-CNN [18] adds a branch to predict segmentation masks in parallel to the existing branches in Faster R-CNN [28] for classification and bounding box regression. Therefore, the Mask R-CNN consists of three parts: feature pyramid network (FPN), regional proposal network (RPN), and detection. Hence, the Mask R-CNN can perform three tasks: object recognition, detection, and segmentation.

In general, MPDs apply some constraints on the reported bounding boxes to improve the performance (i.e. accuracy and precision). The two most common constraints are the bounding box area/size

(a) (b) (c)

Figure 3: Same as Figure 2 except for the detection of a different pedestrian using the FRCNN [28] detector. Three consecutive frames: (a) Frame 666, (b) Frame 667, and (c) Frame 668 as taken from the MOT17-05 video sequence [27]. As was the case for the KDNT detector, the FRCNN detector detects the same pedestrian with three different confidence scores in successive frames.



(a) (b) (c)

Figure 4: Three consecutive frames: (a) Frame 311, (b) Frame 312, and (c) Frame 313 are taken from PETS2009S2L1-View1 video sequence. In (b), the Mask R-CNN [18] detects two pedestrians with single bounding box despite the fact that it detects them correctly before (in (a)) and after (in (c)).

and the bounding box confidence score. In [5], CNN MPD is used to detect pedestrians in a given video frame, wherein the detected bounding boxes with confidence score greater than 0.5 are accepted as true positive. In [7], a fixed threshold for upper confidence is used to create a confidential detection set, wherein detections with low confidence scores are removed from the original detection set at first step. In [10], fixed thresholds for upper and lower confidence scores are used and a sparse optical flow filter is applied to enhance the quality of detections, wherein the upper and lower confidence score thresholds are fixed for all frames in a given video.

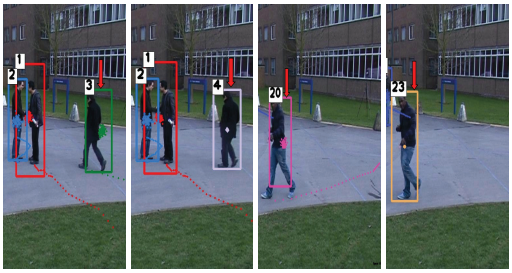
In contrast, applying a lower confidence threshold on the reported bounding boxes to detect all existing pedestrians in the video frames at the cost of increasing the number of false positive detections. This is the case for KDNT [39] and FRCNN [28] where all detected bounding boxes are reported. It should be noted that the same person can appear very differently during its presence in a given video depending on the changes in the background, local illumination, contrast, etc. Thus, the same person may be detected with different confidence scores in two consecutive frames. Therefore, applying upper or lower confidence score thresholds is not a desirable a approach, because the threshold value may vary during a given video or over different videos. Furthermore, most of the CNN MPD methods, mentioned above, and some other MPDs generate pedestrian detections for each frame independently, ignoring inter-frame relationships that exist between consecutive frames. It should be noted that if a pedestrian is present in a frame at time $t-1$ with a high confidence score it will most likely be present in the next frame at time t. For the purpose of illustration, Figure 2 shows that KDNT [39] detects the same pedestrian with three different confidence scores in three consecutive frames. Figure 3 shows similar example for the FRCNN [28] detector. In addition, Figure 4 shows that the Mask R-CNN [18] detects two different pedestrians with single bounding box at a middle frame given that it was able to detect them correctly before and after that middle frame. In this paper, for pedestrian detection component, we develop an efficient online method to detect multiple pedestrians from a video stream by integrating the Mask R-CNN [18] with the post-processing steps proposed in [2] to improve the performance of the Mask R-CNN for multiple pedestrian detections.

2.2 Particle Filter For MPT Algorithms

Many tracking algorithms have been developed over the years, the particle filter (PF) [17] based MPT approaches [6, 15, 16, 24, 29, 31, 34, 36, 38] have shown more promise. The PF operates on the principle of approximating the posterior state distribution by a set of weighted samples, also referred to as particles [4].

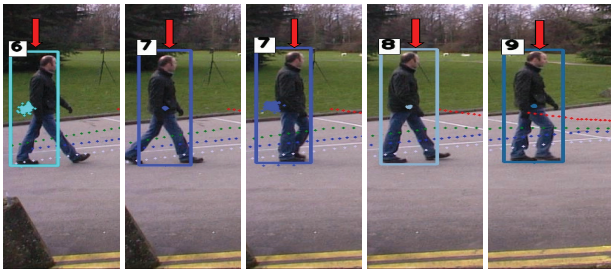
Theoretically, the PF should carry out the ideal prediction (or sampling) step using the actual posterior probability distribution $p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k})$, where $\boldsymbol{x}_k$ is the state of tracked target at time instant k, and $\boldsymbol{z}_{1:k}$ represents all the observations up to time instant k. However, the actual $p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k})$ is unknown, and it does not have a closed-form solution in nonlinear non-Gaussian environment such as MPT system. Alternatively, the SIR filter used a suboptimal probability distribution, called the proposal distribution function, $q(\boldsymbol{x}_k) = p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}, \boldsymbol{z}_{1:k-1})$, which does not consider the latest observation, $\boldsymbol{z}_k$, in the process of sampling, whereas it includes the state transition distribution form previous state to next state given all previous observations except $\boldsymbol{z}_k$ [17]. Thus, the SIR filter employs the information in the latest observation, $\boldsymbol{z}_k$, by updating the weight of the sampled particles using the likelihood function $p(\boldsymbol{z}_k|\boldsymbol{x}_k)$. Then, the SIR filter carries out a resampling procedure over the reweighted particles to solve the degeneracy problem and generate a new set of particles that approximate the actual posterior probability distribution $p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k})$.

Many recent approaches have been proposed for developing a better resampling scheme and choosing an appropriate proposal distribution function, $q(\boldsymbol{x}_k)$, that matches the actual posterior distribution as much as possible in order to avoid degeneracy problem and accurately track the target [11, 26, 32, 37]. In [11, 26], layered or heretical multiple resampling schemes are used to determine different proposal distributions. In [37], a feedback PF is proposed



(a) (b) (c) (d)

Figure 5: Consecutive pairs of frames: (a)-(b) Frames 32 **and** 33**, (c)-(d) Frames** 189 **and** 190 **from the PETS2009S2L1–View5 video. The MPT-LCVMPF failed to track pedestrian moving forward with nonlinear nonconstant velocity. In (a) and (b),** $ID = 3$ **of the pedestrian switched to** 4**. In (c) and (d)** $ID = 20$ **switched to** 23**.**



(a) (b) (c) (d) (e)

Figure 6: Same as Fig. 5. (a)-(b) Frames 71 **and** 72**, (c)-(d) Frames** 74 **and** 75 **from the PETS2009S2L1–View7 video. In (a) and (b), pedestrian's** $ID = 6$ **switched to** 7**. In (c) and (d),** $ID = 7$ **switches to** 8**. In (e), Frames**79 **shows that pedestrian's** $ID = 8$ **switches to** 9**.**

wherein a resampling scheme focusing only on the observations to reweight the particles. In [32], a likelihood-free PF is proposed wherein particles are reweighted without using the likelihood function. However, in tracking scenarios involving multiple pedestrians the above mentioned approaches weaken the impact of the propagated particles using the state transition model on the resampling procedure of the SIR filter [36]. In spite of considerable research and efforts that have recently been deployed for improving the performance of MPT algorithms based on particle filters, finding a balance between using the informative observations and the propagated particles by the state transition model to accurately approximate the actual posterior probability distribution is still a challenging problem [24, 36].

In tracking scenarios involving multiple pedestrians, some pedestrians in the video may move with different and non-uniform velocities for periods of time. Hence, the motion of some pedestrian is highly dynamic, as they often stop, move backward, or turn around. Most of the existing MPT algorithms based on particle filters [6, 15, 16, 24, 29, 31, 34, 36, 38] assume that the motion of pedestrians is often linear and predictable. Hence, these tracking algorithms adopt the linear constant velocity motion (LCVM) model for their state transition model, and they use the state transition distribution $q(\boldsymbol{x}_k) = p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1})$ as the proposal distribution function. Therefore, a major problem facing these tracking algorithms is that its LCVM model fails to propagate particles accurately when a given tracked pedestrian moves with nonlinear nonconstant velocity [36]. Consequently, these MPT tracking algorithms based on particle filter report a high number of ID switches in their tracking results [6, 15, 16, 24, 29, 31, 34, 36, 38]. In addition, they fail to predict pedestrian trajectory correctly. In this paper, we refer to a MPT algorithm implemented using a particle filter with a linear constant velocity motion model as MPT-LCVMPF. For the purpose of illustration, we implemented a MPT-LCVMPF, and we selected a single pedestrian tracking result to demonstrate the effect of using LCVM as a state transition model on ID switches (IDSW). The IDSW effects can be seen when the MPT algorithm is changing the assigned identity of a given pedestrian. Figure 5 shows that MPT-LCVMPF fails to track a pedestrian with the same identification (ID) in two consecutive frames. The ID is changed in the next frame. Figure 6 illustrates a similar situation for a different video sequence.

In this paper, for pedestrian tracking component, we focus on motion model and resampling procedure to improve the tracking abilities of the particle filter. Since data association is a key issue in tracking-by-detection scheme, we propose an efficient adaptive information driven motion (AIDM) model that retains information contained in the particles with higher weights associated with a pedestrian and injects new particles generated from associated pedestrian detection with this tracker. Hence, the proposed MPT algorithm based on the particle filter with an adaptive information driven motion model (referred to as MPT-AIDMPF) can accurately track pedestrians with unpredictable movements and adapted to the motion of the scene. In a video surveillance network, a distributed particle filtering comprised of several localized particle filters (one for each subset of neighboring video cameras) and that the MPT-AIDMPF is a step in that direction. An accurate MPT is essential for any distributed particle filtering. Otherwise. individual localized particle filtering errors would accumulate in the overall tracking estimate formed by combining the outputs of localized particle filters.

We present here a novel multiple pedestrian tracking system based on modified Mask R-CNN and enhanced Particle Filter (PF) using an adaptive information driven motion model for video surveillance. In particular, the main contributions of this paper are:

(1) Develop an efficient online method to detect multiple pedestrians from a video stream by integrating the Mask R-CNN [18] with the post-processing steps proposed in [2] to improve the performance of the Mask R-CNN for multiple pedestrian detections.
(2) Improve the resampling scheme for MPT algorithm using the particle filter by retaining information contained in the propagated particles and injecting new particles generated from the informative observations.
(3) Propose a MPT algorithm based on the particle filter with a novel adaptive information driven motion model (referred to as **MPT-AIDMPF**) that can accurately track pedestrians with unpredictable movements and adapte to the motion of the scene.

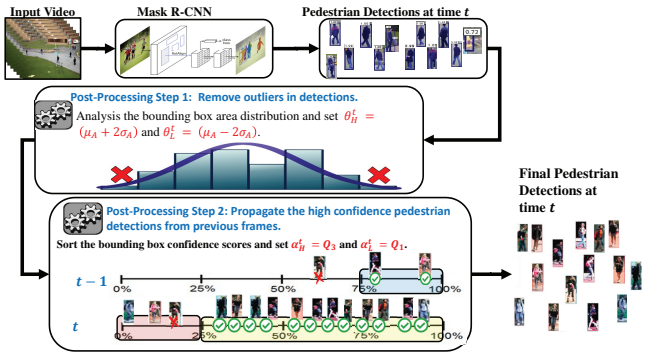


Figure 7: Block diagram of the proposed MPD method. The diagram shows the sequence of post-processing steps, where α_H^t and α_L^t represent the upper and lower confidence threshold values for each frame, respectively. Q_3 is the third quartile which is the median of the upper half of the data set, and Q_1 is the first quartile which is the median of the lower half of the data set.

3 THE PROPOSED MPD METHOD

The proposed method is described in terms of the proposed post-processing steps. These post-processing steps enable MPDs to be more accurate, precise and tolerant to false positive detections in generating pedestrian detections [2]. In [2], an adaptive approach has been used to set both area and confidence score constraints. The proposed method block diagram is depicted in Figure 7.

For Post-Processing Step 1, we calculate the area of the detected bounding boxes and analyze the area distribution in each frame. For frame at time t, the bounding boxes with associated area less than the lower area threshold, denoted by θ_L^t, will be removed. Also, the bounding boxes with associated area greater than the upper area threshold, denoted by θ_H^t, will be removed. For each frame, we calculate both mean, denoted by μ_A, and standard deviation, denoted by σ_A, for the area distribution. We remove outlier pedestrian detections for each frame by assigning $\theta_L = (\mu_A - 2\sigma_A)$ and $\theta_H = (\mu_A + 2\sigma_A)$ [2].

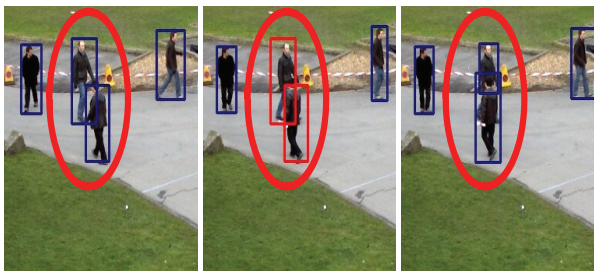
For Post-Processing Step 2, we follow Algorithm 1 in [2] to propagate the high confidence pedestrian detections from the previous frame, and create the final detection set for the current frame. It should be noted that using the Mask R-CNN [18] with the post-processing steps proposed in [2], wherein an adaptive approach to determine both area and confidence scores, generate more accurate pedestrian detection results compared to using the Mask R-CNN [18] alone. Figure 8(b) shows that the proposed MPD method is able to correctly detect the two pedestrians compared to the Mask R-CNN [18] in Figure 4(b).

4 THE PROPOSED MPT-AIDMPF TRACKING ALGORITHM

The proposed MPT algorithm consists of multiple pedestrian tracker and data association components. The proposed pedestrian tracker uses PF with an efficient adaptive information driven motion (AIDM) model and a new resampling procedure to retain information contained in the highly weighted particles of a given pedestrian tracker and injects new particles generated from the associated pedestrian detection with this tracker. The combination between PF and the AIDM model allows the pedestrian tracker to track pedestrians having unpredictable motions with higher tracking accuracy and lower ID switches.

Algorithm 1 MPT-AIDMPF over a Single Camera

Input: $D^k = \left\{d_1^k, d_2^k, \cdots, d_M^k\right\}$, $T^{k-1} = \left\{\tau_1^{k-1}, \tau_2^{k-1}, \cdots, \tau_L^{k-1}\right\}$
Output: T^k
LOOP Process
for every $\tau^{k-1} \in T^{k-1}$ **do**
  state prediction by particle filter
end for
$T^k \leftarrow$ DataAssociation(T^{k-1}, D^k)
LOOP Process
for every $\tau^k \in T^k$ **do**
  if (τ^k does not represent a new pedestrian) **then**
    Update the associated tracker by using the adaptive information driven motion model
  end if
end for

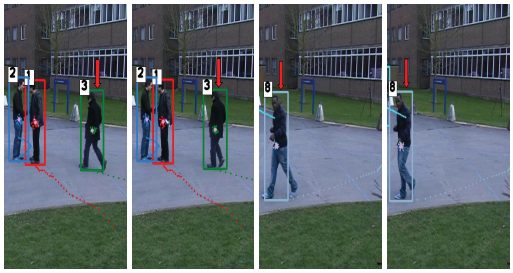


(a) (b) (c)

Figure 8: Same frames as Figure 4. (b) illustrates that the proposed MPD method is able to identify and recover the pedestrians as a true positive detections even if they are detected with single bounding box by the Mask R-CNN [18] in Figure 4(b).

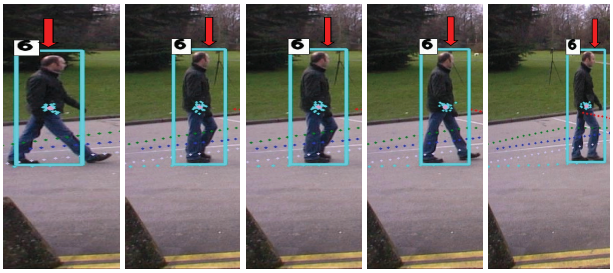
4.1 Outline of the Algorithm

The overview of the algorithm is presented in **Algorithm 1**. For each new frame f_k captured by a single camera at time step k, the previous trackers, $T^{k-1} = \left\{\tau_1^{k-1}, \tau_2^{k-1}, \cdots, \tau_L^{k-1}\right\}$, and current pedestrian detection list, $D^k = \left\{d_1^k, d_2^k, \cdots, d_M^k\right\}$, are used for tracking. The data association component is used to construct the similarity matrix to find the association between existing trackers, $\tau^{k-1} \in T^{k-1}$, and detections, $d^k \in D^k$, at time k. Furthermore, it controls the initialization and termination status of the trackers, and supports the tracker with key-particles from associated detection. In this paper, the target state $\boldsymbol{x}$ consists of two-dimensional



(a) (b) (c) (d)

Figure 9: Consecutive frames: (a)-(b) Frames 71 and 72, (c)-(d) Frames 189 and 190 from the PETS2009S2L1–View5 video. The proposed MPT-AIDMPF algorithm correctly tracks the pedestrian without any ID switch. Frames (a) and (b) show pedestrian with the same $ID = 3$. Similarly, (c) and (d) show pedestrian with $ID = 8$. Note: The pink particle cloud represents the newly generated particles added to the original set of particles. The complete video is available at `https://youtu.be/jQCaMViCd8M`



(a) (b) (c) (d) (e)

Figure 10: Same as Fig. 9 except for a different pedestrian. Consecutive frames: (a)-(b) Frames 71 and 72, (c)-(d) Frames 74 and 75 from the PETS2009S2L1–View7 video. In (e), Frame 79 shows that pedestrian hold his ID 6. The complete video is available at `https://youtu.be/2u1sLMlaUCM`

position (x, y), and velocity in direction of x and y, $(x^{'}, y^{'})$.

$$\boldsymbol{x} = \left[x, y, x^{'}, y^{'}\right]^{T}$$

4.2 Pedestrian Colour-based Appearance Model

When people are walking with significant pose changes, the colour information is considered to be the most trustworthy feature. However, extracting colour information from occluded pedestrian is unreliable. So, we only update the pedestrian appearance model if a pedestrian does not overlap with another pedestrian. In our experiment, the combination of RGB with 8 bins per channel and Hue-Saturation (HS) with 12 bins per channel features yields the best result. Also, we constantly update the pedestrian appearance template as

$$\boldsymbol{F}^{k}_{\tau_i^k} = \alpha \boldsymbol{F}^{k}_{d_j^k} + (1-\alpha)\, \boldsymbol{F}^{k-1}_{\tau_i^{k-1}} \tag{1}$$

where $\boldsymbol{F}^{k}_{\tau_i^k}$ represents the colour feature for tracker τ_i at time k, parameter α specifies the learning (or updating) rate between the last and current features given by $\boldsymbol{F}^{k}_{d_j^k}$. In this paper, $\alpha = 0.05$.

4.3 Data Association

In this paper to link detections to trackers, the similarity matrix $\boldsymbol{S}_k$ between τ_i^{k-1} and d_j^k at time step k is defined as

$$\boldsymbol{S}_k(\tau_i^{k-1}, d_j^k) = A(\tau_i^{k-1}, d_j^k) * O(\tau_i^{k-1}, d_j^k) \tag{2}$$

where $A(\tau_i^{k-1}, d_j^k)$ measures both the appearance similarity using Bhattacharyya coefficient [1, 20] and Euclidean distance between tracker τ_i^{k-1} and detection d_j^k in the logarithmic scale. The $O(\tau_i^{k-1}, d_j^k)$ is the intersection-over-union of the bounding boxes of tracker τ_i^{k-1} and detection d_j^k in the logarithmic scale. We compute the intersection-over-union score based on the PASCAL VOC criterion [12]. The proposed similarity matrix, $\boldsymbol{S}_t$, avoids updating a given pedestrian tracker with confusing nearby detection because it incorporates the proposed pedestrian colour-based appearance model(Section 4.2). In this paper, we compute the appearance similarity measure between the colour feature of $\boldsymbol{F}^{k}_{d_j^k}$ and $\boldsymbol{F}^{k-1}_{\tau_i^{k-1}}$ using the popular Bhattacharyya coefficient [1, 20]. Finally, the assignment between a detection and a tracker is solved optimally using the well-known Hungarian algorithm [21].

The outputs of the data association step are matched trackers, unmatched trackers, and unmatched detections. For unmatched trackers, we used an age threshold T_{age} to terminate these trackers if they are not updated for at least T_{age} frames. This prevents an unbounded growth in the number of unmatched trackers. For unmatched detections, we create a new tracker. For matched tracker, we used the proposed adaptive information driven motion model described next.

4.4 Adaptive information driven motion model and resampling scheme

Given the current particle set at time step k, $\{\boldsymbol{x}_k^i, w_k^i\}_{i=1}^N$, obtained by applying the resampling technique of the particle filter, the matched pedestrian detection is used to update the propagated particles of a given pedestrian tracker. It should be noted that the weights for these particles are normalized so they sum to 1. The new adaptive information driven motion model (AIDM) starts by sorting the current particle set in *descending order*. Then, selecting the highest weighted particles based on resampling proportion coefficient, β, $0 \leq \beta \leq 1$, the number of selected particles will be equal to $(\beta)(N)$, where N is the total number of particles that are used for a pedestrian tracker. After that we use the matched pedestrian detection to generate $(1-\beta)(N)$ new particles with equal weights.

We adopt a dynamic approach to set the resampling proportion coefficient value for the frame at time k, which is denoted by β_k. In each frame, we analyze the distribution of the weight scores

Table 1: Quantitative comparison between proposed MMaskRCNN algorithm and the original Mask R-CNN algorithm. The best results are shown in bold.

MPD algorithm	MODA	MODP
MMaskRCNN (ours)	**73.1%**	**81.90%**
Mask R-CNN [18]	68.15%	77.81%

$\{w_k^i\}_{i=1}^N$ for a pedestrian tracker, and we use the third quartile value to set the resampling proportion coefficient value [2]. The third quartile, denoted by Q_3 , is the median of the upper half of the data set. Then, we generate a new particle set, $(1-\beta)(N)$, from a multivariate Gaussian distribution with means equal to the central coordinates and standard deviations proportional to the height and width of the pedestrian detection, respectively. Finally, we combine the two sets, the propagated particles and the newly generated particles, to create the final set of particles. The weights of the final particle set are normalized so when these particles are reused in the subsequent time step, they accurately describe the previous particles' influence on the tracker. Moreover, the proposed AIDM and resampling scheme have used both the informative observations (pedestrian detections) and the propagated particles to accurately approximate the actual posterior distribution $p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k})$.

5 EXPERIMENTAL RESULTS

We evaluated and tested the proposed MPT system on nine challenging video sequences taken from two publicly available datasets: View1, View5, View6, View7 and View8 from the PETS2009S2L1 [13], and Camera0, Camera1, Camera2, and Camera3 from the EPFL-terrace video sequence [14]. The PETS2009S2L1 is a very challenging video dataset because the pedestrians often change direction and groups form and split frequently. Moverover, the PETS2009S2L1 is widely used in evaluating MPT tracking algorithms. The EPFL-terrace sequence is outdoor sequence consisting of up to nine people appearing one after the other and walking in front of the cameras. It tests the ability of our algorithm to cope with crowded environment. For this purpose, three performance comparisons are taken into consideration. First, we compare the performance of the proposed MPD algorithm (referred to as MMaskRCNN) with that of the original Mask R-CNN algorithm. Second, we compare the performance of the proposed MPT-AIDMPF algorithm with that of the MPT-LCVMPF algorithm. The MPT-LCVMPF represents any tracking algorithm based on PF that adopts the linear constant velocity motion (LCVM) model for its state transition model. Then, we compare the performance of the proposed MPT-AIDMPF with that of other state-of-the-art MPT algorithms [15, 16, 38], which use particle filters in their online MPT algorithm.

5.1 Evaluation metrics

We follow the most frequently used criteria, the CLEAR MOT [35] metrics, i.e., multiple object detection accuracy (MODA), multiple object detection precision (MODP), multiple object tracking accuracy (MOTA), multiple object tracking precision (MOTP), and identity switches (IDSW) to evaluate the performance of both the proposed MPD and MPT algorithms. The MOTA score combines three types of errors: false positive (FP), missed targets (FN), and identity switches (IDSW). The MOTP score shows spatiotemporal overlap between the ground truth tracks and the proposed MPT algorithm output tracks. Finally, the IDSW score shows the number of times the reported identity of a ground-truth track changes.

5.2 Results

For the purpose of illustration, we selected a single pedestrian tracking result to demonstrate the effect of using the proposed MPT-AIDMPF on ID switches. Figure 9 and 10 show that the proposed MPT-AIDMPF algorithm tracks pedestrians without any ID switch as compared to the outputs of MPT-LCVMPF shown in Figure 5 and Figure 6, respectively. Furthermore, Figure 9 and 10 show that the proposed MPT-AIDMPF algorithm accurately predicts the size of bounding box (which fits the pedestrian) as compared to MPT-LCVMPF shown in Figure 5 and Figure 6, respectively.

Table 1 shows the average quantitative evaluations for the performance of the proposed MPD algorithm (referred to as MMaskRCNN) with that of the original Mask R-CNN algorithm [18] for the two datasets.

Table 2 compares the performance of our MPT-AIDMPF algorithm with MPT-LCVMPF for all nine video sequences in term of identity switches (IDSW). Table 2 shows that our proposed MPT-AIDMPF algorithm reduces the number of ID switches to zero for most of the video sequences. To further verify the effectiveness of the proposed MPT-AIDMPF algorithm, a comparative experiment is carried out on the PETS2009S2L1–View1 video sequence using the pedestrian detections provided by the website[1]. It should be noted that the PETS2009S2L1–View1 video sequence is the most commonly used for comparing the performance of MPT algorithms [24]. The PETS2009S2L1–View1 video sequence shows pedestrians walking across an intersection in various directions at variable speed [13]. Table 3 shows the quantitative comparison of the proposed MPT-AIDMPF with other state-of-the-art MPT algorithms. It can be seen that compared with the other MPT algorithms, the proposed MPT-AIDMPF algorithm achieves the best performance in MOTA, MOTP and IDSW scores. The reason for the improved performance is because the proposed MPT-AIDMPF algorithm uses both the informative observations (pedestrian detections) and the propagated particles to accurately track pedestrians.

6 CONCLUSION

In this paper, we improve the performance of the Mask R-CNN for multiple pedestrian detection by using the post-processing steps. Furthermore, we presented a robust adaptive information driven motion model for multiple pedestrian tracking particle filter, MPT-AIDMPF, which enhances the tracking accuracy and precision as well as reduces the number of tracker ID switches in tracking-by-detection approaches. The proposed MPT-AIDMPF algorithm uses an efficient adaptive information driven motion model that retains information contained in the highly weighted particles of a given pedestrian tracker and injects new particles generated from the associated pedestrian detection with the tracker. The proposed MPT-AIDMPF algorithm was evaluated on multiple video sequences taken from two publicly available datasets, where it achieves superior performance as compared to the MPT-LCVMPF algorithm and

[1] http://www.milanton.de/

Table 2: The identity switches (IDSW) score for the proposed MPT-AIDMPF and MPT-LCVMPF algorithms.

Method	PETS2009S2L1					EPFL-Terrace			
	View1	View5	View6	View7	View8	Camera0	Camera1	Camera2	Camera3
MPT-LCVMPF	8	48	96	47	71	2	4	5	0
MPT-AIDMPF	**0**	**2**	**1**	**0**	**1**	**0**	**0**	**0**	**0**

Table 3: Quantitative comparison between proposed MPT-AIDMPF algorithm and different MPT algorithms on the PETS2009S2L1–View1 video sequence. The best results are shown in bold.

MPT algorithm	MOTA↑	MOTP↑	IDSW↓
Gomez [16]	51.1%	75.0%	27
Yoon [38]	66.6%	57.4%	34
GSDL [15]	80.3%	61.5%	33
MPT-AIDMPF	**92.0%**	**81.20%**	**12**

Evaluation metrics with symbol (↑) indicates higher score is better; while for evaluation metrics with symbol (↓) indicates lower score is better

other state-of-the-art MPT algorithms. Moreover, it was shown that considering adaptive information driven motion (AIDM) model is important to improve the performance of MPT algorithm in video surveillance applications.

7 ACKNOWLEDGMENTS

This research is supported in part by Natural Science and Engineering Research Conference (NSERC), Canada through the Create grant entitled *CreateDAV: Data Analytics and Visualization*. The authors would like to acknowledge support from IBM in carrying out the project. Computations were performed on the SOSCIP Consortium's [Parallel-CPU, GPU and/or Cloud Analytics] computing platform(s). SOSCIP is funded by FedDev Ontario, IBM Canada Ltd. and Ontario academic member institutions.

REFERENCES

[1] Frank J. Aherne, Neil A. Thacker, and Peter I. Rockett. 1998. The Bhattacharyya Metric as an Absolute Similarity Measure for Frequency Coded Data. *Kybernetika* 34 (1998), 363–368.

[2] M. Al-Shatnawi, V. Movahedi, A. Asif, and A. An. 2018. Improving Real-Time Pedestrian Detection Using Adaptive Confidence Thresholding and Inter-Frame Correlation. In *2018 IEEE 20th International Workshop on Multimedia Signal Processing (MMSP)*. 1–5. https://doi.org/10.1109/MMSP.2018.8547103

[3] M. Andriluka, S. Roth, and B. Schiele. 2008. People-Tracking-by-Detection and People-Detection-by-Tracking. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*. 1–8. https://doi.org/10.1109/CVPR.2008.4587583

[4] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. 2002. A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking. *IEEE Transactions on Signal Processing* 50, 2 (Feb. 2002), 174–188. https://doi.org/10.1109/78.978374

[5] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft. 2016. Simple Online and Realtime Tracking. In *2016 IEEE International Conference on Image Processing (ICIP)*. 3464–3468. https://doi.org/10.1109/ICIP.2016.7533003

[6] M. D. Breitenstein, F. Reichlin, B. Leibe, E. Koller-Meier, and L. Van Gool. 2011. Online Multiperson Tracking-by-Detection from a Single, Uncalibrated Camera. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 9 (Sept. 2011), 1820–1833. https://doi.org/10.1109/TPAMI.2010.232

[7] J. Chen, H. Sheng, Y. Zhang, and Z. Xiong. 2017. Enhancing Detection Model for Multiple Hypothesis Tracking. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2143–2152. https://doi.org/10.1109/CVPRW.2017.266

[8] P. Dollar, C. Wojek, B. Schiele, and P. Perona. 2009. Pedestrian Detection: A Benchmark. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 304–311. https://doi.org/10.1109/CVPR.2009.5206631

[9] R. Douc and O. Cappe. 2005. Comparison of Resampling Schemes for Particle Filtering. In *ISPA 2005. Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis, 2005.* 64–69. https://doi.org/10.1109/ISPA.2005.195385

[10] V. Eiselein, E. Bochinski, and T. Sikora. 2017. Assessing Post-Detection Filters for a Generic Pedestrian Detector in a Tracking-by-Detection Scheme. In *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. 1–6. https://doi.org/10.1109/AVSS.2017.8078484

[11] Víctor Elvira, Luca Martino, David Luengo, and Mónica F. Bugallo. 2016. Heretical Multiple Importance Sampling. *IEEE Signal Processing Letters* 23, 10 (Oct. 2016), 1474–1478. https://doi.org/10.1109/LSP.2016.2600678

[12] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. 2010. The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision* 88, 2 (June 2010), 303–338. https://doi.org/10.1007/s11263-009-0275-4

[13] J. Ferryman and A. Shahrokni. 2009. PETS2009: Dataset and Challenge. In *2009 Twelfth IEEE International Workshop on Performance Evaluation of Tracking and Surveillance*. 1–6. https://doi.org/10.1109/PETS-WINTER.2009.5399556

[14] F. Fleuret, J. Berclaz, R. Lengagne, and P. Fua. 2008. Multicamera People Tracking with a Probabilistic Occupancy Map. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30, 2 (Feb. 2008), 267–282. https://doi.org/10.1109/TPAMI.2007.1174

[15] Zeyu Fu, Pengming Feng, Federico Angelini, Jonathon Chambers, and Syed Mohsen Naqvi. 2018. Particle PHD Filter Based Multiple Human Tracking Using Online Group-Structured Dictionary Learning. *IEEE Access* 6 (2018), 14764–14778. https://doi.org/10.1109/ACCESS.2018.2816805

[16] David Gerónimo Gomez, Frédéric Lerasle, and Antonio M. López Peña. 2012. State-Driven Particle Filter for Multi-Person Tracking. In *Advanced Concepts for Intelligent Vision Systems*, 14th International Conference) (Ed.). Springer, Berlin, Heidelberg, 467–478. https://doi.org/10.1007/978-3-642-33140-4_41

[17] N. J. Gordon, D. J. Salmond, and A. F. M. Smith. 1993. Novel Approach to Nonlinear/Non-Gaussian Bayesian State Estimation. *IEE Proceedings F (Radar and Signal Processing)* 140, 2 (April 1993), 107–113. https://doi.org/10.1049/ip-f-2.1993.0015

[18] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. 2017. Mask R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*. 2961–2969.

[19] Jan Hosang, Mohamed Omran, Rodrigo Benenson, and Bernt Schiele. 2015. Taking a Deeper Look at Pedestrians. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4073–4082.

[20] T. Kailath. 1967. The Divergence and Bhattacharyya Distance Measures in Signal Selection. *IEEE Transactions on Communication Technology* 15, 1 (Feb. 1967), 52–60. https://doi.org/10.1109/TCOM.1967.1089532

[21] H. W. Kuhn. 2005. The Hungarian Method for the Assignment Problem. *Naval Research Logistics (NRL)* 52, 1 (Feb. 2005), 7–21. https://doi.org/10.1002/nav.20053

[22] Tian-cheng Li, Gabriel Villarrubia, Shu-dong Sun, Juan M. Corchado, and Javier Bajo. 2015. Resampling Methods for Particle Filtering: Identical Distribution, a New Method, and Comparable Study. *Frontiers of Information Technology & Electronic Engineering* 16, 11 (Nov. 2015), 969–984. https://doi.org/10.1631/FITEE.1500199

[23] Xi Li, Weiming Hu, Chunhua Shen, Zhongfei Zhang, Anthony Dick, and Anton Van Den Hengel. 2013. A Survey of Appearance Models in Visual Object Tracking. *ACM Trans. Intell. Syst. Technol.* 4, 4 (Oct. 2013), 58:1–58:48. https://doi.org/10.1145/2508037.2508039

[24] Wenhan Luo, Junliang Xing, Anton Milan, Xiaoqin Zhang, Wei Liu, Xiaowei Zhao, and Tae-Kyun Kim. 2017. Multiple Object Tracking: A Literature Review. *arXiv:1409.7618 [cs]* (May 2017). arXiv:1409.7618 [cs]

[25] L. Martino, V. Elvira, and F. Louzada. 2016. Weighting a Resampled Particle in Sequential Monte Carlo. In *2016 IEEE Statistical Signal Processing Workshop (SSP)*. 1–5. https://doi.org/10.1109/SSP.2016.7551711

[26] L. Martino, V. Elvira, D. Luengo, and J. Corander. 2017. Layered Adaptive Importance Sampling. *Statistics and Computing* 27, 3 (May 2017), 599–623. https://doi.org/10.1007/s11222-016-9642-5

[27] Anton Milan, Laura Leal-Taixe, Ian Reid, Stefan Roth, and Konrad Schindler. 2016. MOT16: A Benchmark for Multi-Object Tracking. *arXiv:1603.00831 [cs]* (March 2016). arXiv:1603.00831 [cs]

[28] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 91–99.

[29] Ricardo Sanchez-Matilla, Fabio Poiesi, and Andrea Cavallaro. 2016. Online Multi-Target Tracking with Strong and Weak Detections. In *Computer Vision – ECCV 2016 Workshops (Lecture Notes in Computer Science)*. Springer, Cham, 84–99. https://doi.org/10.1007/978-3-319-48881-3_7

[30] S. Santhoshkumar, S. Karthikeyan, and B. S. Manjunath. 2013. Robust Multiple Object Tracking by Detection with Interacting Markov Chain Monte Carlo. In *2013 IEEE International Conference on Image Processing*. 2953–2957. https://doi.org/10.1109/ICIP.2013.6738608

[31] Guang Shu, Afshin Dehghan, Omar Oreifej, Emily Hand, and Mubarak Shah. 2012. Part-Based Multiple-Person Tracking with Partial Occlusion Handling. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 1815–1821. https://doi.org/10.1109/CVPR.2012.6247879

[32] Fabian Sigges, Marcus Baum, and Uwe D. Hanebeck. 2017. A Likelihood-Free Particle Filter for Multi-Obiect Tracking. In *2017 20th International Conference on Information Fusion (Fusion)*. 1–5. https://doi.org/10.23919/ICIF.2017.8009796

[33] A. W. M. Smeulders, D. M. Chu, R. Cucchiara, S. Calderara, A. Dehghan, and M. Shah. 2014. Visual Tracking: An Experimental Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36, 7 (July 2014), 1442–1468. https://doi.org/10.1109/TPAMI.2013.230

[34] Nan Song, Kezhi Li, and Wei Chen. 2018. Robust Visual Tracking Via Adaptive Structure-Enhanced Particle Filter. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 1578–1582. https://doi.org/10.1109/ICASSP.2018.8461727

[35] Rainer Stiefelhagen, Keni Bernardin, Rachel Bowers, John Garofolo, Djamel Mostefa, and Padmanabhan Soundararajan. 2006. The CLEAR 2006 Evaluation. In *Multimodal Technologies for Perception of Humans (Lecture Notes in Computer Science)*. Springer, Berlin, Heidelberg, 1–44. https://doi.org/10.1007/978-3-540-69568-4_1

[36] Xuedong Wang, Tiancheng Li, Shudong Sun, and Juan M. Corchado. 2017. A Survey of Recent Advances in Particle Filters and Remaining Challenges for Multitarget Tracking. *Sensors (Basel, Switzerland)* 17, 12 (Nov. 2017). https://doi.org/10.3390/s17122707

[37] Tao Yang, Richard S. Laugesen, Prashant G. Mehta, and Sean P. Meyn. 2016. Multivariable Feedback Particle Filter. *Automatica (Journal of IFAC)* 71, C (Sept. 2016), 10–23. https://doi.org/10.1016/j.automatica.2016.04.019

[38] J. H. Yoon, M. H. Yang, J. Lim, and K. J. Yoon. 2015. Bayesian Multi-Object Tracking Using Motion Context from Multiple Objects. In *2015 IEEE Winter Conference on Applications of Computer Vision*. 33–40. https://doi.org/10.1109/WACV.2015.12

[39] Fengwei Yu, Wenbo Li, Quanquan Li, Yu Liu, Xiaohua Shi, and Junjie Yan. 2016. POI: Multiple Object Tracking with High Performance Detection and Appearance Feature. In *Computer Vision – ECCV 2016 Workshops (Lecture Notes in Computer Science)*. Springer, Cham, 36–42. https://doi.org/10.1007/978-3-319-48881-3_3

[40] Zhong-Qiu Zhao, Peng Zheng, Shou-Tao Xu, and Xindong Wu. 2019. Object Detection With Deep Learning: A Review. *IEEE Transactions on Neural Networks and Learning Systems* 30, 11 (Nov. 2019), 3212–3232. https://doi.org/10.1109/TNNLS.2018.2876865

Understanding Brain Dynamics for Color Perception Using Wearable EEG Headband

Mahima Chaudhary
Lassonde School of Engineering
York University, Toronto, Canada
cmahima@yorku.ca

Sumona Mukhopadhyay
Lassonde School of Engineering
York University, Toronto, Canada
mukhopas@yorku.ca

Marin Litoiu
Lassonde School of Engineering
York University, Toronto, Canada
mlitoiu@yorku.ca

Lauren E Sergio
Faculty of Health
York University, Toronto, Canada
lsergio@yorku.ca

Meaghan S Adams
Faculty of Health
York University, Toronto, Canada
msadams@yorku.ca

ABSTRACT

The perception of color is an important cognitive feature of the human brain. The variety of colors that impinge upon the human eye can trigger changes in brain activity which can be captured using electroencephalography (EEG). In this work, we have designed a multiclass classification model to detect the primary colors from the features of raw EEG signals. In contrast to previous research, our method employs spectral power features, statistical features as well as correlation features from the signal band power obtained from continuous Morlet wavelet transform instead of raw EEG, for the classification task. We have applied dimensionality reduction techniques such as Forward Feature Selection and Stacked Autoencoders to reduce the dimension of data eventually increasing the model's efficiency. Our proposed methodology using Forward Selection and Random Forest Classifier gave the best overall accuracy of 80.6% for intra-subject classification. Our approach shows promise in developing techniques for cognitive tasks using color cues such as controlling Internet of Thing (IoT) devices by looking at primary colors for individuals with restricted motor abilities.

KEYWORDS

Wearable computing, Machine learning, Brain Computer Interface

1 INTRODUCTION

The advancements in sensor technologies have facilitated the growth of wearable headband devices for development in Brain-Computer Interface (BCI) applications. One such device is the Muse 2 headband [1] which is a portable non-invasive device that allows capturing of EEG signals. In this work, we analyzed the relationship between EEG signals and color stimuli using the Muse 2 headband. The objective was to extract the information (features) from EEG signals to classify or distinguish them based on the red(R), green(G), and blue(B) colors that were used to stimulate the cortical activity. The classification result could be potentially used in an integrated IoT environment where it could be used to control appliances [28, 36]. One such application could be, where people with restricted motor ability could switch on/off appliances by looking at a particular color. The study that we do is a proof of concept, it can be extended to other colors also. However, the proposed application would require input and effort from specialists in other fields too. The work can also be expanded in the healthcare field where it could be used to detect color blindness [11].

Previously, classification tasks like these have been performed [8, 13, 29] using sophisticated medical-grade EEG devices with multiple sensors but in our work, we used a simple four-electrode/channel consumer-friendly device to record the raw EEG signals. The use of Muse headband allowed portability to our work and its integration with IoT. Also contrary to previous approaches, in our study, we used features like power, variance in power, various pairwise cross-correlation features and several other statistical features from the signal band power obtained from continuous Morlet wavelet transform for classification task instead of raw EEG signals or event-related potential (ERP) values. The raw EEG data was preprocessed and features that were important to study the effect of color stimuli on EEG were extracted from the data using digital signal processing techniques.

We mainly focused on Alpha and Beta frequency bands, as these are most likely to be stimulated when a person is alert, attentive, or concentrating and not performing a high cognitive activity. We employed various linear and non-linear Machine Learning (ML) algorithms namely, K Nearest Neighbors, Support Vector Machine (SVM), Logistic Regression, Random Forest, models like Artificial Neural Networks, and boosting approaches like Gradient Boosting, to perform the three-class classification task. We investigated the classification performance of ML algorithms both on a single person's data (intra-subject) as well as on combining the data from different people (inter-subject). We also applied dimensionality reduction techniques like forward feature selection and stacked autoencoders to increase the performance of the architecture. The main research questions addressed in this paper are:

- Is it possible to distinguish EEG signals from a four-channel wearable headband, produced by RGB color exposure, by training ML models on features that account for statistical, spectral and correlation properties of EEG?

[1] https://choosemuse.com/

CASCON'20, November 10-13 2020, Toronto, Canada

- Can feature reduction techniques like Forward-Feature Selection(supervised) and Autoencoders (unsupervised) make the ML algorithms for EEG classification more efficient?
- Does the performance of ML algorithms differ for inter-subject and intra-subject classification?

The rest of the paper is organized as follows. Section 2 contains the related work. In Section 3 we describe our proposed methodology. Section 4 presents our evaluation metrics followed by experimental results in Section 5. Concluding remarks are presented in Section 6.

2 RELATED WORK

In recent years, researchers have used wearable headbands to analyze the response of EEG under different stimuli. K. Johannesen et. al.[19] used SVM to derive useful EEG features in order to predict working memory performance in schizophrenia and healthy adults. The authors in [9] used a regression model trained on data gathered from cognitive tasks (collected from a 6-channel EEG headset) in order to model mental workload using EEG features for intelligent systems. In [3, 23] the EEG data has been used to examine driver's alertness during driving sessions.
Diane Aclo et al.[1] used a 14 channel EEG device to monitor the effect of color stimuli on people. They used features like power spectral density and waveform length for classification using an Artificial Neural network. In [2] feature selection algorithm has been investigated for EEG signal due to RGB colors using screwable gold EEG electrodes. Arnab Rakshit et al. [27] proposed the use of a fuzzy space classifier to discriminate colors from EEG by using a 10 electrode device. In [26], an Emotiv headset has been used to study separation and classification of EEG response to color stimuli by using SVM. Zhang et el. in [34] showed how alpha and beta band powers are affected by stimuli from RGB colors. All the above classification tasks have been conducted using complicated EEG devices in contrast to our work. Furthermore, our proposed method achieves a high accuracy using the Muse headband. Recently, the use of portable headband devices have gained popularity due to their ease of use and accessibility. The authors in [35, 36] have used G.tec's MOBIlab four channel portable device in a problem similar to ours. However, their method yielded a lower accuracy of 58% in comparison to our proposed approach. A headband from Mindwave Neurosky has also been used [4] in a task similar to our. However, the authors achieved a lower accuracy of 53% with their method. Our results have shown significant improvement. In many studies, Muse has also been used to acquire EEG signals for various classification tasks. EEG-based excitement detection in immersive environments has been studied by Jason et al in [30]. Krigolson et al. [21]used Muse headband to Assess Human visual attention by assigning subjects an "oddball"task wherein they saw a series of infrequently and frequently appearing circles and were instructed to count the number of target circles that they saw. However they did not apply any ML model in their work. In [7] classification task has been performed to classify recreational and instructional video sessions using Muse. They used spectral power and connectivity features from raw EEG in their work and got the best performance with SVM and Logistic Regression model.

3 PROPOSED METHODOLOGY

The main frequencies captured by EEG data are in form of specific human EEG signals namely Delta with frequency 3Hz or below (Deep dreamless sleep), Theta with frequency from 3.5-8 Hz (Deep meditation), Alpha with frequency 8-12 Hz (Calm relaxed yet alert state), Beta with frequency 13-30 Hz (Active, busy thinking) and Gamma with a frequency greater than 41 Hz (Higher mental activity)[20]. Each type of frequency band signal represents a different state of consciousness of mind ranging from sleep to active thinking. We mainly focused on Alpha and Beta frequency bands. The work in this paper has been accomplished in the following five phases: data acquisition, data cleaning and preprocessing, feature extraction, dimensionality reduction, and classification of data into red, green, and blue. We shall explain each component in detail.

3.1 Data Acquisition

Muse headband consists of four channels/electrodes namely AF7 and TP9 on the left and AF8 and TP10 on the right. These are named and positioned according to the International 10-20 System [2], as shown in Figure 1. The sampling rate of Muse is 256Hz. The data from all channels was collected. There were eight subjects (aged 18-30yrs) who participated in the visual experiment. The experiment was conducted using the University of Nottingham's Psychopy 3 [25] toolbox. Five trials each four minutes long were conducted for each participant at different times. In each trial, a color from RGB was shown in a random order, twenty times each, for a period of two seconds each, such that a black color was shown for two seconds between each of the RGB colors to provide a baseline to the experiment. The experiment was conducted in a dark room and the subjects were told to do minimum facial movements and eye blinks. A similar protocol has been followed in previous experiments too [1, 28, 31]. The time period of the stimulus or the main color was kept small to only capture the effect of color on the cortical excitability. The data from Muse headband was collected using Muse SDK[3] and a third party application for Muse called Mind Monitor[4]. The Mind Monitor application indicates potential jaw clenches and eye blinks in the EEG data. We used this capability as a marker to get the starting timestamp of the data. The experiment started with a jaw clench which was captured by Mind Monitor and from that time stamp, the data was separated according to the color stimuli. The architecture we used is shown in Figure 2 and the detailed experimental setup is shown in Figure 3.

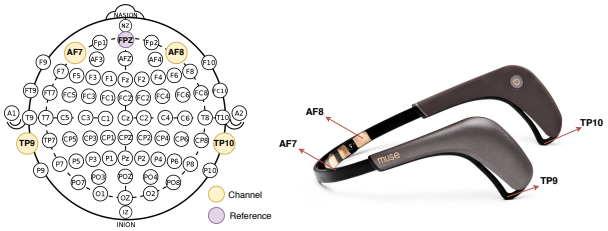


Figure 1: The 10-20 system of electrode placement for Muse

[2] https://en.wikipedia.org/wiki/10%E2%80%9320_system_(EEG)
[3] https://choosemuse.com/development/
[4] https://mind-monitor.com/

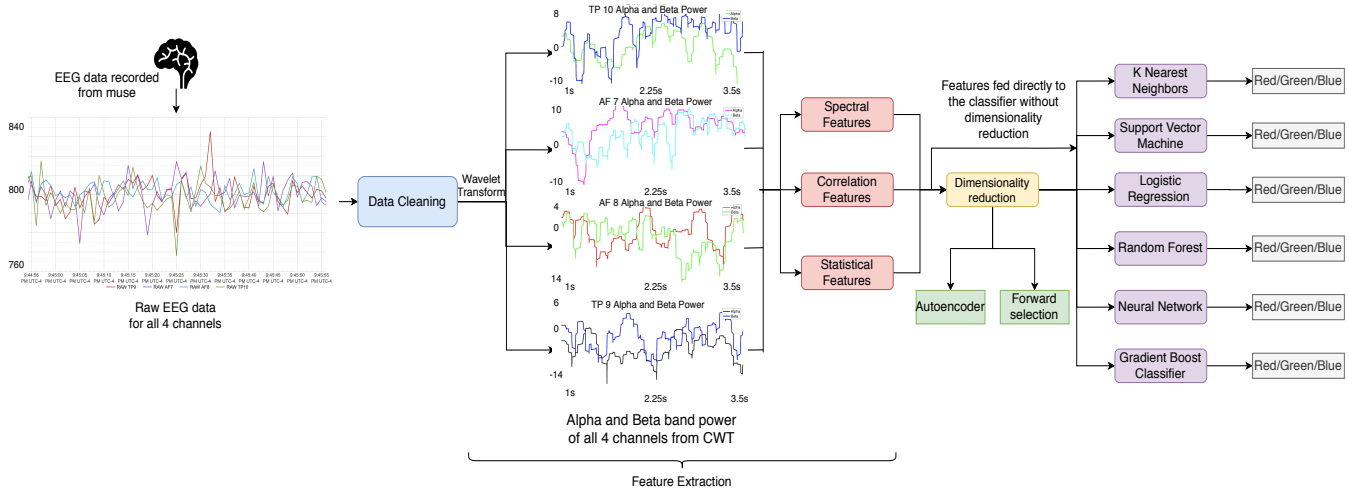


Figure 2: The architecture used in the methodology.

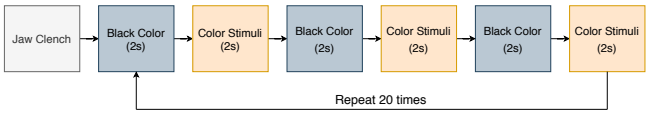


Figure 3: The experiment protocol for data acquisition

3.2 Data cleaning and preprocessing

The raw EEG data is generally very noisy and it needs to be cleaned and pre-processed in order to remove artifacts from it. In our methodology, we cleaned the data in two steps. Firstly we analyzed the data using Matlab's EEG lab software [5] and labeled any visible unwanted spikes and noise manually from the data. Secondly, we divided the data into small time windows of 50 ms and computed the variance of data in each window, if it was more than a selected threshold then the time window was flagged. We also examined the individual subject's data and used the trial that has a minimum number of jaw clenches and eye blinks for further experimentation.

3.3 Feature Extraction

Feature extraction is a very vital part of our problem. The use of raw EEG data did not give good results in our experiment and so we used Time-Frequency analysis to find frequency band coefficients that were most relevant for our problem i.e. Alpha coefficients(8-12Hz) and Beta coefficients(13-30Hz). In past works, [7, 10] Discrete wavelet transform(DWT) has been used to extract the frequency bands of interest. However in our case, we were not interested in all the frequency bands, instead, we only considered alpha and beta bands. The use of DWT would have given us an improper breakdown of bands with the Alpha band in the range of 8-16Hz and beta in range of 16-32Hz and therefore to avoid this we used Continuous wavelet transform method as done in [22, 32] to extract the bands of interest. The mother wavelet that we used was the Morlet wavelet. The Morlet wavelet has a peak in the center after which it tapers to the edges. The complex Morlet wavelet can be obtained by the convolution of a Gaussian with a sine wave and it is represented by the following equation:

$$w(t,f) = A * exp(-t^2/2\sigma_t^2) exp(2\pi f t) \quad (1)$$

where t is time, A=$(\sigma_t\sqrt{\pi})^{-1/2}$, where σ_t is duration of the wavelet and f is the frequency of wavelet. We extracted the power of alpha and beta bands from our EEG signal by convolution of Morlet wavelets of frequencies ranging from 8Hz to 30Hz along the whole signal at each time point. This was done with the help of Fast Fourier Transform (FFT). For each frequency, we first performed FFT of the signal and then the FFT of the Morlet wavelet. We then performed the convolution of the two transformed signals and applied Inverse Fourier transform to get the time-domain representation of data. The magnitude of the complex transformed signal was then extracted and it was squared to obtain the power across all time points. The important thing here is that we rejected the imaginary part as it gave us the phase information and the real part just gave us the band-passed signal but what we were more interested in was the power therefore we extracted the magnitude of the complex signal. We got a spectrogram like representation of the power of the signal, with columns denoting the time points and rows denoting the frequencies from 8Hz to 30Hz. Figure 4 shows the spectrograms for RGB colors. The Morlet wavelet helped to reduce edge artifacts and noise from the data. It also helped to obtain a balance in temporal precision and frequency precision. The sampling rate of the signal and the Morlet wavelet was kept the same in order to perform convolution. Figure 5 shows the EEG data with and without the application of the Morlet wavelet. In Algorithm 1 we summarize the procedure we followed to find alpha and beta band power from the raw EEG signal. We removed the flagged artifacts that we got in Data cleaning step after applying the wavelet transform. This step was done after the transform was applied so that we did not reduce the points below the sampling frequency of 256Hz. The features were extracted from the remaining data. The features accounted for the spectral, correlation as well as statistical properties of the data which were normalized using z-score. We experimented with different time windows of length 100ms, 200ms, 500ms, 1000ms. Each window was taken with a 50% overlap with the next window. Each window was used

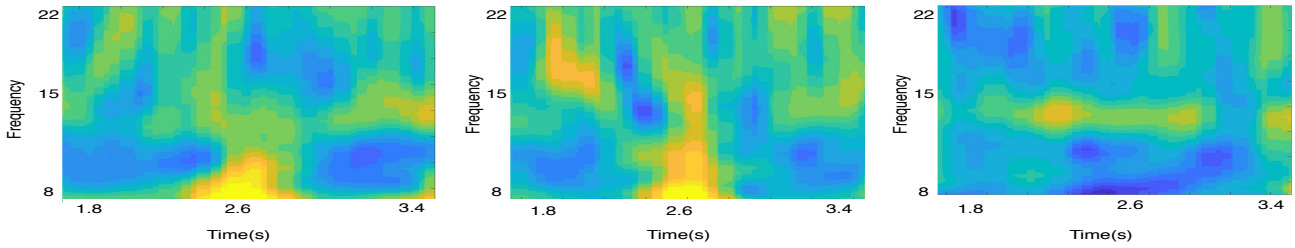


Figure 4: Spectrograms of Red, Green, and Blue respectively. Each spectrogram is made by the EEG obtained from an average of 20 trials (also called Event-related potential) of Subject 1 from channel AF7. The spectrograms show that on the onset of activity at 2.4 sec there is an increase in the power of alpha and low beta frequency band in case of green and red however the power of alpha-band decreases and that of low beta increases in case of blue on the onset of stimuli.

to extract a single row of features vector. The next feature vector was obtained by moving the window half of its length.

Algorithm 1: How to extract alpha and beta band powers from Raw EEG data from Morlet wavelet convolution

Input: Raw EEG data
Output: Alpha and Beta band power in form of spectrogram
1. Initialize the FFT parameters i.e. the minimum and maximum frequency, the time period of the wavelet which is equal to the sampling rate of the signal, the result matrix.
2. Find the FFT of the Raw EEG data.
3. while $Frequency \leq MaxFrequency$ **do**
  4. *Create a complex morlet wavelet from frequency by convolution of sine wave and gaussian.*
  5. *Find the FFT of the wavelet.*
  6. *Find the convolution of FFT of signal and FFT of wavelet by pointwise multiplication.*
  7. *Find inverse fourier transform of convoluted signal to convert back to time domain.*
  8. *Extract the magnitude of the complex signal and square it to get the absolute power component of the signal and add it to the result matrix.*
end

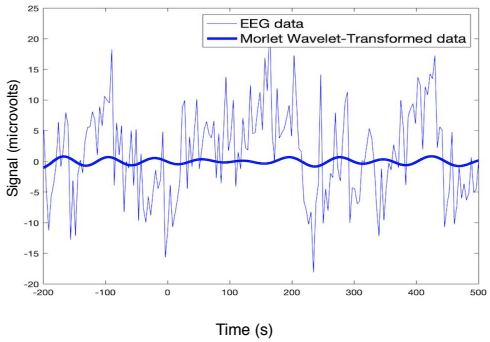


Figure 5: The original EEG signal and its Morlet-convolution version using a wavelet of 30 Hz

3.3.1 Spectral features. These features were calculated by taking into account the average power of each band, the variance in the power of each band, and the hemispherical difference in each band over a time window for each of the four channels. Thus we got 18 features(8 average power coefficients for each channel, 8 variance power coefficient for each channel, and 2 hemispheric difference coefficients) for each sample that was formed by a single time window. This method was similar to the one followed in [7]. This set of spectral features are the most commonly used features in many EEG related studies as they allow the model to evaluate any potential changes in the absolute band power due to stimuli.

3.3.2 Pairwise Correlation features. In addition to the spectral features, it is also important to study the correlation among different frequency bands from different channels. We calculated this using a pairwise correlation in each time window for each band and each electrode. [7] follows this method too. We got a total of 28 correlation features using this method from a single time window. These features were helpful to find cross-region similarity as some of our data was discontinuous because of artifact removal.

3.3.3 Statistical Features. Features that represent the statistical properties of the signal like Kurtosis, Skewness, Shannon Entropy and Hjorth Parameters were also extracted.

Kurtosis, Skewness and Shannon Entropy . Kurtosis is a measure of outliers in data. Data with less value of Kurtosis has less number of outliers. The Skewness measures the asymmetry in data. The entropy is a measure of information in data. We calculated each of these parameters both for alpha and beta bands, therefore we got 24 features from these properties.

Hjorth Parameters. They are indicators of statistical properties used in signal processing in the time domain introduced by Bo Hjorth in 1970 [6]. We obtained 16 Hjorth parameters for alpha and beta band for all 4 channels. We calculated two Hjorth parameters namely, the mobility parameter as in equation 2 and complexity parameter as in equation 3 on alpha and beta power bands that we obtain from CWT.

$$Mobility = \sqrt{\frac{var\frac{dy(t)}{dt}}{var\,y(t)}} \tag{2}$$

$$Complexity = \sqrt{\frac{Mobility\frac{dy(t)}{dt}}{Mobility(y(t))}} \tag{3}$$

Here $y(t)$ is the alpha or beta band power for a time window. We got a total of 40 statistical features. Table 1 shows all the features

Table 1: The features obtained from raw data.

	Avg. Power features	Var. Power features	Hem. diff features	Correlation features	Kurtosis	Sknewness	Shannon Entropy	Hjorth Parameters
Alpha Band	TP9, TP10, AF7, AF8 (4)	TP9, TP10, AF7, AF8 (4)	|Left sensors-right sensors| (1)	Cross corr of alpha & beta of all sensors	TP9, TP10, AF7, AF8 (4)	TP9, TP10, AF7, AF8 (4)	TP9, TP10, AF7, AF8 (4)	TP9, TP10, AF7, AF8 (8)
Beta Band	TP9, TP10, AF7, AF8 (4)	TP9, TP10, AF7, AF8 (4)	|Left sensors-right sensors| (1)	Cross corr of alpha & beta of all sensors	TP9, TP10, AF7, AF8 (4)	TP9, TP10, AF7, AF8 (4)	TP9, TP10, AF7, AF8 (4)	TP9, TP10, AF7, AF8 (8)
No. of Features	8	8	2	28	8	8	8	16

obtained from raw EEG data. Figure 6 shows the visualization of features in 2-D space by applying Linear Discriminant Analysis [33]. It shows that the three classes are almost separable.

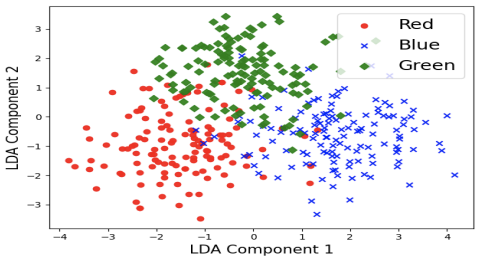


Figure 6: Visualization of data of Subject 1 for a single trial in 2-D space using Linear Discriminant Ananlysis

3.4 Dimensionality reduction

The process of feature reduction is important because it has many advantages like reduced training times, simplified and interpretable models, reduced chances of overfitting i.e. lesser variance and less impact of the curse of dimensionality. We performed feature selection/dimensionality reduction by two different methods. Firstly we used the Forward Feature selection technique which is a supervised approach and secondly, we used Autoencoders [15] which is an unsupervised approach for feature reduction. We elaborate on them in the following subsection.

3.4.1 Forward Feature Selection. In this method, we started by selecting one feature and calculating the metric value for each feature on the cross-validation dataset. The feature offering the best metric value was selected and appended to a list of features. The process was repeated next time with two features, one selected from the previous iteration and the other one selected from the set of all features not present in the set of already chosen features. The metric value(f-measure) was computed for each set of two features and features offering the best metric value were appended to the list of relevant features. This process was repeated until we had the desired number of features. The number of features was reduced to 10 features. The reduced feature set of size ten was chosen after experimenting with feature sets of different sizes, the top ten features gave the best balance between accuracy and number of features.

3.4.2 Stacked Autoencoders for Feature Extraction. Autoencoders are neural networks that can be used to reduce the data into a low dimensional latent space by stacking multiple non-linear transformations(layers). They have an encoder-decoder architecture. The encoder maps the input to latent space and the decoder reconstructs the input. The data in latent space is supposed to have encoded the most important features and has a dimension lesser than the original dimension of data. This data in the latent space can be used as a reduced feature set and the models can be trained on this data. The number of features was reduced to 10(similar to that using forward feature selection) using a stacked autoencoder structure shown in Figure 7. This architecture was chosen after an exhaustive experiment with various architectures.

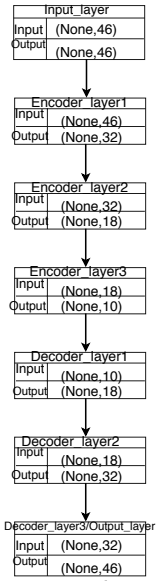


Figure 7: Final Autoencoder architecture used

3.5 Classification Task

We applied ML models on the 86 features that we extracted by the procedure explained in Section 3.3. The classification task

was done in two folds. We first considered the data from individual subjects and applied models to that data to perform intra-subject classification for which we achieved an accuracy of 80.6%. Intra-subject classification helped us to study subject-specific differences of the EEG reactivity patterns. Then we considered the combined data from all the subjects and performed inter-subject classification and got an accuracy of 58.1%. The inter-subject case helped us to make a more generalized model. The classification was done in two ways and their performances have been compared. We performed classification using the original feature set as well as the reduced feature set from forward selection and autoencoders for both intra-subject and inter-subject. Below we elaborate on the models that we have used along with the chosen hyperparameters. We tuned the hyperparameters using Grid Search. The range of hyperparameters chosen was based on previous works [7, 36].

3.5.1 K Nearest Neighbor (KNN). KNN is a non-parametric and lazy learning algorithm. Non-parametric means there is no assumption for underlying data distribution and that's why we tested it in our problem. K is a critical hyperparameter that we varied in the range 4 to 8 in our experiment. The Euclidean distance was used as the distance metric. As KNN is a lazy learner, therefore it is not advisable to use it in our application, we use it for comparison purposes only.

3.5.2 Logistic Regression (LR). We used logistic regression model both with ridge and lasso regularization. We varied the parameter C or penalty term in the range 0.01 to 100. We found out that lasso regularization gave better results on our data.

3.5.3 Random Forest (RF). The random forest algorithm is an ensemble approach that uses multiple decision trees and makes a classification decision by voting from all the trees. The number of estimators in our problem were varied from 10 to 100.

3.5.4 Artificial Neural Network (NN). We have used ANN with the following architecture: First hidden layer with 300 neurons and second hidden layer with 100 neurons. The activation function used was sigmoid. L2 regularization had been used to avoid overfitting, with a regularization rate of 0.0001. The hyperparameter tuning was done using grid search.

3.5.5 Support Vector Machine (SVM). SVM with RBF kernel has been used in our experiment. The hyperparameters C and Gamma were varied between 0.001 and 100 and 0.01 and 10 respectively.

3.5.6 Gradient Boosting (GB). Gradient boosting is an ensemble learning approach that produces a prediction model in the form of an ensemble of weak prediction models. Gradient boosting combines weak learners into a single strong learner. In our Gradient boosting model we varied the hyperparameter estimators from 10 to 100.

4 EVALUATION METRICS

Many metrics are used to evaluate ML Models like average accuracy, precision, recall, F-measure, ROC-AUC score, MCC score etc. In our case, we used three metrics for performance evaluation of our models- Average Accuracy, Average ROC-AUC score, and Average Matthews Correlation Coefficient (MCC). Since our data is balanced i.e. each class has almost equal representation the average accuracy score would have sufficed but we used the other two additional metrics to verify the performance of our models. We used scikit learn [24] library of python to evaluate the models.

4.1 Accuracy Score

The accuracy score in our problem was calculated as :

$$\text{Average Accuracy Score}(y, \hat{y}) = \frac{1}{n_{sample}} \Sigma_{i=0}^{n_{samples}-1} 1(y_i = \hat{y}_i) \tag{4}$$

In equation 8, $\hat{y}_i$ is the predicted value of the i-th sample and y_i is the corresponding true value and $1(x)$ is the indicator function. $n_{samples}$ is the total number of samples. The accuracy indicates the samples that were correctly classified from all the samples.

4.2 ROC-AUC score

ROC-AUC stands for Receiver operator characteristics- Area under the curve, it basically calculates the area under the receiver operator curve.The ROC curve is created by plotting the true positive rate (TPR = $\frac{TP}{TP+FN}$) against the false positive rate (FPR = $\frac{FP}{TN+FP}$) at various threshold settings. We find the area under the curve to evaluate our model. Since our problem is multiclass therefore we computed the average AUC of all possible pairwise combinations of classes using equation 5 as suggested in [17].

$$\text{Average ROC-AUC Score} = \frac{2}{c(c-1)} \Sigma_{j=1}^{c} \Sigma_{k>j}^{c} (\text{AUC}(j|k) + \text{AUC}(k|j)) \tag{5}$$

where c is the number of classes and AUC($j|k$) is the AUC with j as the positive class and k as the negative class and AUC($k|j$) is vice versa. In general, AUC($j|k$) $\neq$ AUC($k|j$)) in the multiclass case.

4.3 Matthews Correlation Coefficient

The Matthews correlation coefficient [12] is used to evaluate the quality of binary and multiclass classifications. The MCC is a kind of correlation coefficient value between -1 and +1. A coefficient of +1 represents a perfect prediction, 0 an average random prediction and -1 an inverse prediction. In the multiclass case, Matthews correlation coefficient can be defined in terms of a confusion matrix C for K classes. The MCC for multiclass as suggested in [18] is calculated as follows:

$$MCC = \frac{c \times s - \Sigma_k^K p_k \times t_k}{\sqrt{(s^2 - \Sigma_k^K p_k^2) \times (s^2 - \Sigma_k^K t_k^2)}} \tag{6}$$

where $t_k = \Sigma_i^K C_{ik}$ is the number of times class k truly occurred, $p_k = \Sigma_i^K C_{ki}$ is the number of times class k predicted, $c = \Sigma_k^K C_{kk}$ is the total number of samples correctly predicted and $s = \Sigma_i^K \Sigma_j^K C_{ij}$ the total number of samples.

5 EXPERIMENT RESULTS

We performed two categories of classification namely, intra-subject and inter-subject classification. For intra-subject classification, we applied 5-folds cross-validation for data from five trials of each subject and in the second classification, we applied leave out one

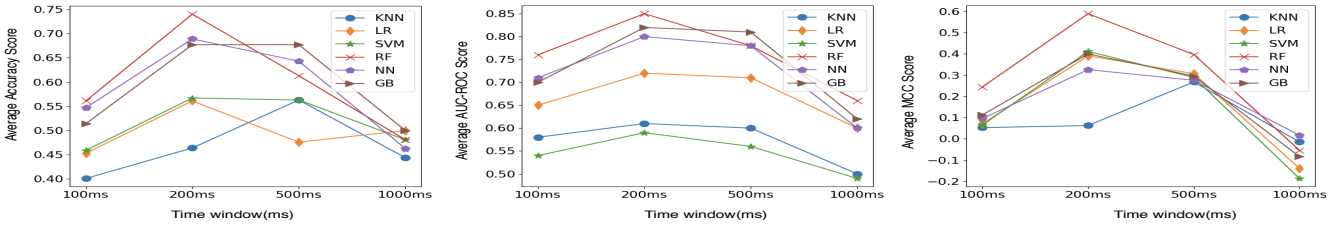


Figure 8: Average Accuracy, Average ROC-AUC score and Average MCC score for different time windows for intra-subject classification

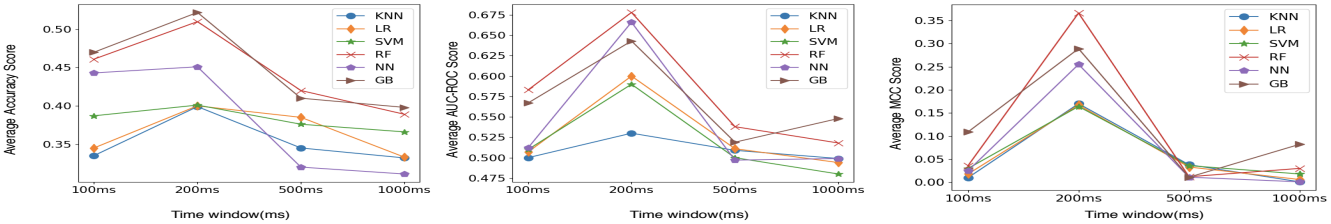


Figure 9: Average Accuracy, Average ROC-AUC score and Average MCC score for different time windows for inter-subject classification

subject cross-validation where we trained the model on seven subjects data and validated it using a single subject data and we repeated it for all subjects. The performance metrics that we used to evaluate our model are average cross validation accuracy, average ROC-AUC score, and average MCC. We report the results on the complete dataset as well on the reduced dataset from dimensionality reduction techniques that we mentioned in Section 3.4. The average accuracy, average AUC score, and average MCC score with different time windows for both intra-subject and inter-subject classification are shown in Figure 8 and Figure 9 respectively.

We got the best results for a time window of 200ms. We considered the time window of 200ms for further experimentation. We discuss the results in three segments, the results without dimensionality reduction, results after dimensionality reduction from the forward selection algorithm and results after dimensionality reduction from Autoencoder. In the following subsections, we show the results in Table 2-11, considering the average metric score of all the subjects, the best metric score among all subjects and the inter-subject metric score for the three metrics explained in Section 4 . In all the tables the number in brackets is the standard deviation. We have highlighted the highest metrics for each case in all tables. The code for all the experiments is available online [5].

5.1 Results without dimensionality reduction

Table 2 shows the accuracy score by using all features. We saw that the Random Forest algorithm performed the best and Neural Network and Gradient Boosting classifier also showed comparable results. The highest accuracy for an individual was 70.2% which was reasonably better than the accuracy of random guess i.e. 33%. The inter-subject accuracy of 56.8% was also very promising considering the fact that we applied leave one subject out cross-validation in this case. The results were better for intra-class classification which means that a customized model could be trained on an individual's data and then it can be used for predictions for a particular subject rather than using data from different people which might also cause privacy issues.

Table 2: Accuracy by using all the features at 200ms time window.

Metrics	KNN	SVM	Logistic Regression	Random Forest	Neural Network	Gradient Boost
Avg Subject Accuracy	0.414 (0.056)	0.474 (0.018)	0.506 (0.028)	**0.625 (0.018)**	0.523 (0.013)	0.608 (0.057)
Best Subject Accuracy	0.513 (0.021)	0.578 (0.031)	0.600 (0.022)	**0.702 (0.000)**	**0.700 (0.026)**	**0.669 (0.039)**
Inter-subject Accuracy	0.338 (0.033)	0.377 (0.030)	0.408 (0.030)	**0.568 (0.025)**	0.490 (0.033)	0.472 (0.040)

In Table 3 we see the ROC-AUC scores. The highest average score of 0.851 was achieved by the Random Forest classifier, the average score of 0.803 was also better than an AUC score of 0.5 in the case of a random classifier.

Table 3: ROC-AUC Score by using all the features at 200ms time window

Metrics	KNN	SVM	Logistic Regression	Random Forest	Neural Network	Gradient Boost
Avg Subject Auc score	0.552 (0.019)	0.514 (0.032)	0.676 (0.027)	**0.803 (0.012)**	0.696 (0.035)	0.763 (0.015)
Best Subject Auc score	0.600 (0.043)	0.620 (0.057)	0.710 (0.037)	**0.851 (0.020)**	0.822 (0.024)	0.810 (0.012)
Inter-subject Auc score	0.530 (0.039)	0.570 (0.021)	0.601 (0.046)	**0.654 (0.051)**	0.611 (0.035)	0.643 (0.034)

[5]https://github.com/cmahima/MuseProject

The MCC scores by using all features are in Table 4. A MCC score of 0 means that the classifier is predicting randomly, in our case the highest MCC score was 0.523 which was much higher than a random prediction.

Table 4: MCC score by using all the features at 200ms time window

Metrics	KNN	SVM	Logistic Regression	Random Forest	Neural Network	Gradient Boost
Avg Subject MCC	0.120 (0.050)	0.291 (0.040)	0.310 (0.041)	**0.433 (0.033)**	0.303 (0.056)	0.433 (0.033)
Best Subject MCC	0.203 (0.102)	0.333 (0.070)	0.318 (0.041)	**0.523 (0.097)**	0.431 (0.070)	0.479 (0.091)
Inter-subject MCC	0.207 (0.072)	0.1645 (0.057)	0.167 (0.059)	**0.366 (0.075)**	0.155 (0.060)	0.189 (0.058)

5.2 Results with Forward Feature Selection

We saw significant improvement in the results, especially for Random Forest classifier, with the use of forward feature selection which is a supervised feature selection technique. The irrelevant and noisy features were removed and the feature set was reduced to 10. This methodology helped us to curb the overfitting issue too and thus the performance on the validation set improved. In Table 5 we see the average accuracy scores by using the top 10 features. There was an increase in average accuracy by nearly 10% and we got the highest accuracy of almost 80.6% which was much better than any other previous approaches that have been used for EEG classification using RGB colors using wearable devices. The average subject accuracy of 72% showed that the classifier performed well for all the subjects. The average accuracy increased by 9.5%. In this case, also the results of intra-subject classification were better than that of inter-subject classification. The inter-subject classification accuracy improved by 1.3%. Random Forest algorithm had given us the best results in this case too with Neural network and Gradient Boost with comparable performance.

Table 5: Accuracy by using 10 features by forward selection at 200ms time window

Metrics	KNN	SVM	Logistic Regression	Random Forest	Neural Network	Gradient Boost
Avg Subject Accuracy	0.492 (0.038)	0.487 (0.045)	0.492 (0.028)	**0.720 (0.035)**	0.513 (0.036)	0.597 (0.048)
Best Subject Accuracy	0.615 (0.051)	0.604 (0.028)	0.590 (0.050)	**0.806 (0.041)**	0.766 (0.039)	0.720 (0.035)
Inter-subject Accuracy	0.377 (0.013)	0.366 (0.024)	0.388 (0.012)	**0.581 (0.032)**	0.475 (0.040)	0.411 (0.019)

We see in Table 6 the AUC scores after forward feature selection. The best AUC score increased by 0.037 and the average AUC score has increased by 0.054. The MCC scores with forward feature selection are in Table 7 which also increased. Thus forward feature selection not only made our architecture efficient computationally but also increased the overall performance of the architecture. In fact, we got the best accuracy of 80.6% with the use of the Random Forest classifier with forward selection.

Table 6: ROC-AUC Score by using 10 features by forward selection at 200ms time window

Metrics	KNN	SVM	Logistic Regression	Random Forest	Neural Network	Gradient Boost
Avg Subject Auc score	0.546 (0.017)	0.588 (0.034)	0.688 (0.014)	**0.857 (0.031)**	0.699 (0.033)	0.740 (0.037)
Best Subject Auc score	0.775 (0.018)	0.670 (0.018)	0.712 (0.045)	**0.901 (0.013)**	0.879 (0.011)	0.860 (0.015)
Inter-subject Auc score	0.540 (0.023)	0.580 (0.062)	0.611 (0.026)	**0.676 (0.023)**	0.610 (0.025)	0.632 (0.014)

Table 7: MCC score by using 10 features by forward selection at 200ms time window

Metrics	KNN	SVM	Logistic Regression	Random Forest	Neural Network	Gradient Boost
Avg Subject MCC	0.150 (0.030)	0.289 (0.072)	0.302 (0.037)	**0.578 (0.061)**	0.437 (0.031)	0.310 (0.061)
Best Subject MCC	0.531 (0.054)	0.346 (0.054)	0.364 (0.067)	**0.638 (0.013)**	0.553 (0.068)	0.515 (0.056)
Inter-subject MCC	0.017 (0.112)	0.189 (0.051)	0.197 (0.017)	**0.476 (0.015)**	0.255 (0.012)	0.289 (0.043)

5.3 Results with Autoencoder

We applied autoencoder to observe how an unsupervised feature reduction technique would work on our data. With the autoencoder, a reduced feature set of 10 was obtained. Using this reduced feature set as input to the ML models, we achieved a lower average CV accuracy in comparison to classification using forward feature selection. Therefore autoencoders are not recommended for our application. Table 8-10 show the metrics achieved with the use of autoencoders. The highest accuracy we got was from Gradient Boost classifier which was around 51%. This accuracy is far from any practical use and very less when compared to the accuracy from forward feature selection, thus we conclude that unsupervised feature reduction technique of autoencoder does not work well on our EEG data.

Table 8: Accuracy by using 10 features by Autoencoder at 200ms time window

Metrics	KNN	SVM	Logistic Regression	Random Forest	Neural Network	Gradient Boost
Avg Subject Accuracy	0.398 (0.010)	0.362 (0.026)	0.393 (0.022)	**0.430 (0.011)**	0.409 (0.009)	0.417 (0.000)
Best Subject Accuracy	0.417 (0.000))	0.406 (0.000)	0.434 (0.000)	0.489 (0.008)	0.473 (0.024)	**0.510 (0.000)**
Inter-subject Accuracy	0.358 (0.012)	0.348 (0.027)	0.347 (0.023)	0.397 (0.017)	0.355 (0.028)	**0.398 (0.012)**

Table 9: ROC-AUC Score by using 10 features by Autoencoder at 200ms time window

Metrics	KNN	SVM	Logistic Regression	Random Forest	Neural Network	Gradient Boost
Avg Subject Auc score	0.501 (0.037)	0.499 (0.012)	0.518 (0.044)	**0.637 (0.024)**	0.598 (0.026)	0.600 (0.017)
Best Subject Auc score	0.655 (0.000)	0.560 (0.018)	0.602 (0.000)	0.686 (0.007)	0.688 (0.020)	**0.730 (0.001)**
Inter-subject Auc score	0.505 (0.019)	0.499 (0.062)	0.520 (0.026)	**0.548 (0.018)**	0.510 (0.037)	0.511 (0.036)

Table 10: MCC score by using 10 features by Autoencoder at 200ms time window

Metrics	KNN	SVM	Logistic Regression	Random Forest	Neural Network	Gradient Boost
Avg Subject MCC	0.099 (0.019)	0.189 (0.072)	0.191 (0.026)	**0.225 (0.019)**	0.218 (0.039)	0.199 (0.067)
Best Subject MCC	0.261 (0.000)	0.267 (0.000)	0.213 (0.000)	0.229 (0.027)	**0.301 (0.038)**	0.261 (0.000)
Inter-subject MCC	0.015 (0.022)	0.024 (0.053)	0.022 (0.051)	**0.212 (0.031)**	0.120 (0.054)	0.115 (0.022)

In the results, we saw that the accuracies for inter-subject and intra-subject classification varied. This can possibly be due to inter-subject variability in EEG as suggested in [14, 16]. There sometimes exits a variability in the amplitude of different EEG peaks, so some people might show more excitability at given time-points than others. There is also often a bit of variability in latency - signals might vary in when they occur, but the range is pretty tight, like 5-10ms either way. Thus these reasons may have resulted in our model's lower performance in inter-subject case.

The final proposed model for our application is that of Random Forest classifier with forward feature selection. In Table 11 we compare our results with previous efforts that have been done to classify EEG signals on the basis of color stimuli using wearable EEG devices. In Figure 10 we show the ROC curve for the proposed model with AUC-ROC score of individual classes for all the subjects where 0 represents red, 1 represents green and 2 represents blue.

Table 11: Performance of other methods on EEG classification into color stimuli. Our aprroach shows 5-folds average accuracy value for intra-class classification

Algorithms	Best Average Accuracy
Martin Angelovski et al.[4] using 2 channel portable EEG	53%
Sara Åsly et al.[35, 36] using 4 channel portable EEG	58%
Kyle Phillips et al.[26] using 14 channel emotiv EEG	79.6%
Rakshit et al.[27] using 10 channel medical EEG	81.2%
Our approach using 4 channel portable EEG	**80.6%**

6 CONCLUSIONS AND FUTURE WORK

We have used EEG signals from a wearable consumer-grade EEG headband to classify the raw EEG data into three classes of colors, red, green, and blue. In our approach, we focussed mainly on Alpha and Beta frequencies and discarded all other lower and higher frequencies which otherwise would have added noise to the data. We extracted various spectral, correlation and statistical features from the data and apply ML models to it. Our proposed model of Random Forest with forward feature selection showed significant improvement when compared to previous approaches. Our methodology achieved an improvement of almost 20% in the average accuracy of classification.

Despite having a fewer number of electrodes Muse performed well in the classification task and gave promising results. The intra-class classification accuracy of 80.6% shows that wearable devices can be used in integrated IoT frameworks where they can be used in various control applications. The IoT pipeline for this application must take into account the data preprocessing and feature extraction in real-time. The time window for our particular application was small to capture the effect of color stimuli only and avoid unnecessary artifacts in data. This time-window might vary for different applications. One drawback of Muse that we encountered during experiments was that it cannot be worn for a long time due to comfort issues and also the connection can become weak sometimes however one can overcome this problem by applying water to the channels. With the advancement in wearable computing, more comfortable devices are now available that would not bother one if used for a longer time like the new Muse S headband. The Muse 2 device is also sensitive to muscle movements but that is not an issue in our application as we are only interested in a small time window of data when a person focuses on a color. Our work

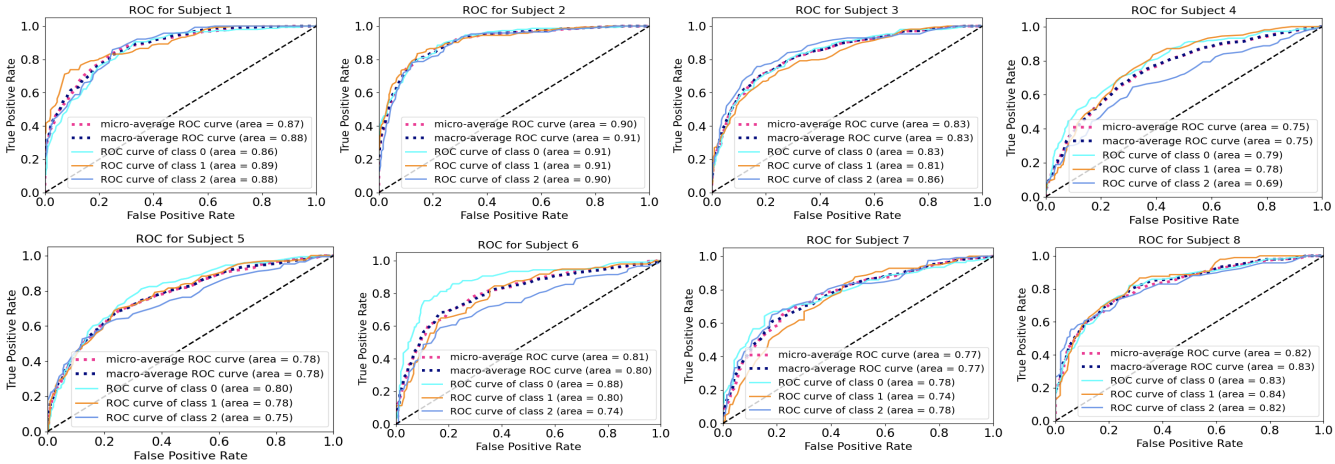


Figure 10: The ROC-AUC curve using our proposed architecture for all the subjects

has thus highlighted the capability of these wearable devices to detect and classify the EEG signal on the basis of color stimuli and the results are encouraging. This study opens up a new door to integrate these devices in our day to day lives to use brain signals to control various devices.

REFERENCES

[1] Diane Aclo et al. 2015. EEG-based Color Classification System using Artificial Neural Networks. *LPU-Laguna Journal of Engineering and Computer Studies* (10 2015).

[2] Eman Alharbi, Saim Rasheed, and Seyed Buhari. 2016. Feature selection algorithm for evoked EEG signal due to RGB colors. In *2016 9th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*. 1503–1520.

[3] Mohammad Almogbel, Anh Dang, and Wataru Kameyama. 2018. EEG-signals based cognitive workload detection of vehicle driver using deep learning. In *2018 20th International Conference on Advanced Communication Technology (ICACT)*. 256–259.

[4] Martin Angelovski et al. 2012. Application of BCI Technology for Color Prediction Using Brainwaves. *ICT Innovations 2012, Web Proceedings ISSN 1857-7288* (09 2012).

[5] Scott Makeig Arnaud Delorme. 2004. EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics including independent component analysis. *J Neurosci Methods* (2004).

[6] Hjorth B. 1970. EEG analysis based on time domain properties. *Electroencephalogr Clin Neurophysiol*. 227 (1970), 306-310.

[7] Pouya Bashivan, Irina Rish, and Steve Heisig. 2016. Mental State Recognition via Wearable EEG. *CoRR* (2016).

[8] Alberto Bozal. 2017. *Personalized Image Classication from EEG Signals using Deep Learning*. Master's thesis.

[9] Maher Chaouachi, Imène Jraidi, and Claude Frasson. 2011. Modeling Mental Workload Using EEG Features for Intelligent Systems. In *User Modeling, Adaption and Personalization*. 50–61.

[10] Thiago da Silveira, Alice Kozakevicius, and Cesar Rodrigues. 2016. Automated drowsiness detection through wavelet packet analysis of a single EEG channel. *Expert Systems with Applications* 55 (03 2016).

[11] Wm Dobelle. 2000. Artificial Vision for the Blind by Connecting a Television Camera to the Visual Cortex. *ASAIO journal* (01 2000), 3–9.

[12] Baldi Pierre et al. 2000. Assessing the accuracy of prediction algorithms for classification: An overview. *Bioinformatics (Oxford, England)* (06 2000), 412–24.

[13] Chris Berka et al. 2004. Real-Time Analysis of EEG Indexes of Alertness, Cognition, and Memory Acquired With a Wireless EEG Headset. *International Journal of Human–Computer Interaction* 17, 2 (2004), 151–170.

[14] Galin D et al. 1982. Sex and handedness differences in EEG measures of hemispheric specialization. *Brain and Language* (1982).

[15] K. Han et al. 2018. Autoencoder Inspired Unsupervised Feature Selection. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2941–2945.

[16] Richard J. Riding et al. 1997. Cognitive Style and Individual Differences in EEG Alpha During Information Processing. *Educational Psychology* (1997).

[17] David Hand et al. 2001. A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems. *Hand, The* (11 2001), 171–186.

[18] Gorodkin J. 2004. Comparing two K-category assignments by a K-category correlation coefficient. *Comput Biol Chem* (2004), 367-374.

[19] Jason Johannesen, Jinbo Bi, Ruhua Jiang, Joshua Kenney, and Chi-Ming Chen. 2016. Machine learning identification of EEG features predicting working memory performance in schizophrenia and healthy adults. *Neuropsychiatric Electrophysiology* 2 (12 2016).

[20] Zuzana Koudelková and Martin Strmiska. 2018. Introduction to the identification of brain waves based on their frequency. *MATEC Web of Conferences* (01 2018), 05012.

[21] Olav Krigolson, Chad Williams, and Francisco Colino. 2017. Using Portable EEG to Assess Human Visual Attention. *International Conference on Augmented Cognition* (05 2017), 56–65.

[22] Hyeon Kyu Lee and Young-Seok Choi. 2019. Application of Continuous Wavelet Transform and Convolutional Neural Network in Decoding Motor Imagery Brain-Computer Interface. *Entropy* 21 (12 2019), 1199.

[23] Chin-Teng Lin et al. 2007. EEG-Based Assessment of Driver Cognitive Responses in a Dynamic Virtual-Reality Driving Environment. *IEEE transactions on biomedical engineering* 54 (08 2007), 1349–52.

[24] F Pedregosa et al. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[25] J. Peirce et al. 2019. PsychoPy2: Experiments in behavior made easy. (51 2019).

[26] Kyle Phillips, Olli Fosu, and Ismail Jouny. 2015. Separation and classification of EEG responses to color stimuli. In *2015 41st Annual Northeast Biomedical Engineering Conference (NEBEC)*. 1–2.

[27] Arnab Rakshit and Rimita Lahiri. 2016. Discriminating different color from EEG signals using Interval-Type 2 fuzzy space classifier. In *2016 IEEE 1st International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES)*. 1–6.

[28] Saim Rasheed and Daniele Marini. 2015. Classification of EEG Signals Produced by RGB Colour Stimuli. *Journal of Biomedical Engineering and Medical Imaging* (10 2015).

[29] Concetto Spampinato et al. 2017. Deep Learning Human Mind for Automated Visual Classification. *CVPR 2017* (09 2017).

[30] Jason Teo and Jia Chia. 2018. EEG-based excitement detection in immersive environments: An improved deep learning approach. *AIP Conference Proceedings* (09 2018), 020145.

[31] David Vivancos. 2018. MindBigData, The Imagenet of the Brain. (51 2018).

[32] Hao Wang et al. 2010. The continuous analysis of EEG's alpha wave by morlet wavelet transform. *National Center for Biotechnology Information* (08 2010), 746–8, 752.

[33] Jieping Ye. 2007. Least squares linear discriminant analysis. *Proceedings of the 24th international conference on Machine learning* (01 2007), 1087–1093.

[34] Huiran Zhang and Zheng Tang. 2011. To judge what color the subject watched by color effect on brain activity. In *IJCSNS International Journal of Computer Science and Network Security*. 80.

[35] Sara Åsly. 2019. *Supervised learning for classification of EEG signals evoked by visual exposure to RGB colors*. Ph.D. Dissertation.

[36] Sara Åsly, Luis Moctezuma, Monika Gilde, and Marta Molinas. 2019. Towards EEG based classification of RGB color-based stimuli. *8th Graz Brain-Computer Interface Conference* (09 2019).

Towards Interpretable and Maintainable Supervised Learning Using Shapley Values in Arrhythmia

Sanjena Krishnakumar
Ryerson University
Toronto, Ontario, Canada
sanjena.krishnakumar@ryerson.ca

Tamer Abdou
Ryerson University
Toronto, Ontario, Canada
tamer.abdou@ryerson.ca

ABSTRACT

This paper investigates the application of a model-agnostic interpretability technique, Shapley Additive Explanations (SHAP), to understand and hence, enhance machine learning classification models using Shapley values in the prediction of arrhythmias[1]. Using the Arrhythmia dataset[2], three different feature selection techniques, Information Gain (IG), Recursive Feature Elimination-Random Forest (RFE-RF), and AutoSpearman, were used to select features for machine learning models to predict the arrhythmia class. Four multi-class classification models, Naïve Bayes (NB), k-Nearest Neighbours (kNN), Random Forest (RF), and stacking heterogeneous ensemble (Ensemble) were built, evaluated, and compared. SHAP interpretation method was applied to find reliable explanations for the predictions of the classification models. Additionally, SHAP values were used to find 'bellwether' instances to enhance the training of our models in order to improve their performances in the prediction of arrhythmia. The most stable and top-performing classification model was RF, followed by Ensemble in comparison to NB and kNN. SHAP provided robust and reliable explanations for the classification models. Furthermore, improving the training of our models with 'bellwether' instances, found using SHAP values, enhanced the overall model performances in terms of accuracy, AUC, and F1 score. In conclusion, we recommend using SHAP value explanations as a robust and reliable method for local model-agnostic interpretability and to enhance machine learning models for arrhythmia prediction.

KEYWORDS

SHAP, LIME, Shapley value, local model-agnostic interpretation, bellwether, multi-class classification, machine learning, healthcare, arrhythmia

ACM Reference Format:
Sanjena Krishnakumar and Tamer Abdou. 2020. Towards Interpretable and Maintainable Supervised Learning Using Shapley Values in Arrhythmia.

[1]This paper was written as part of the Certificate in Data Analytics, Big Data, and Predictive Analytics at Ryerson University
[2]https://archive.ics.uci.edu/ml/datasets/Arrhythmia

CASCON'20, November 10-13 2020, Toronto, Canada

1 INTRODUCTION

Arrhythmia is a cardiac condition in which the heart beats abnormally. There are several types of arrhythmia with varying severity: asymptomatic to causing sudden cardiac deaths (SCD). According to CANet, SCD due to arrhythmia causes about 40,000 deaths annually in Canada and was expected to be the top cause of morbidity and mortality in 2020[3]. The electrocardiogram (ECG) measures the electrical activity of the heart and is important for accurate diagnoses, preventive measures, and treatments for patients with arrhythmia. Machine learning algorithms would further help deepen our understanding of arrhythmia and further improve the accuracy and precision of medical diagnoses. In healthcare and medicine, physicians and clinicians require explanations and transparency in diagnosing medical conditions. However, complex machine learning models are often black box models that lack interpretability, making them difficult to trust in practice even though these models often have high accuracies. Therefore, interpreting these models is invaluable for clinicians in understanding how the models' predictions are decided using features and their values as well as improving model performances to provide more reliable medical diagnoses.

Local Interpretable Model-agnostic Explanation (LIME) is a valuable method using local surrogate models to explain individual predictions of a model [18]. LIME uses one explanation model on different prediction models [18]. However, its disadvantages include the neighbourhood of a data point of interest being large, its inconsistency and instability of explanations for two close data points, and repeating sampling could result in different explanations, making it hard to trust their explanations [15, 18]. However, Shapley Additive Explanations (SHAP) overcomes these shortcomings by using Shapley value, where the difference between an instance's prediction and average prediction of a dataset is fairly distributed among the feature values to provide exact explanations [18]. Moreover, SHAP provides more reliable explanations for models by integrating LIME and Shapley value, coming from coalitional game theory [15, 18]. Since calculating Shapley values is computationally expensive, SHAP approximates the Shapley values using a specified number of samples from the dataset [15, 18]. Furthermore, the additive property of LIME in SHAP allows for global interpretation methods using aggregations of Shapley values on prediction models [18]. Therefore, SHAP would be a reliable and robust local model-agnostic method for the interpretation of arrhythmia prediction models in this study.

[3]https://canet-nce.ca/

'Bellwether' instances are examples within a dataset that could be used to train machine learning models to provide better predictions for other data points and reduce the instability of class predictions [11, 12]. Additionally, the feasibility and effectiveness of using 'bellwether' instances to improve prediction models in arrhythmia will be investigated. Using 'bellwether' examples would bring forth valuable instances while reducing undesirable noise and erroneous data, due to example mistyping, within datasets to build enhanced machine learning models. Therefore, we propose to evaluate four different multi-class classification models, employ SHAP for model interpretability, and use its Shapley values to discover 'bellwether' instances to enhance the proposed models and their performances. Here are the research questions for this study:

RQ1: How do the classification models perform and compare in predicting arrhythmia?
RQ2: What interpretations for the applied classification models are found using local model-agnostic Shapley value explanations, and how reliable is this method?
RQ3: How does selecting 'bellwether' instances using SHAP values improve the classification models for the prediction of arrhythmia?

This paper will firstly discuss the heart, ECG, and arrhythmias, works related to this paper, and the methodology for dimensionality reduction, building of different classification models for the prediction of arrhythmia, interpreting these models using SHAP, and improving these models using 'bellwether' instances found using Shapley values. Next, in results, feature subsets found from feature selection, the evaluation of the classification models, local and global interpretation of the models using SHAP, and the evaluation of 'bellwether' training instances to improve these models will be presented. Lastly, this paper concludes with threats to validity, and conclusion and future work.

2 BACKGROUND AND RELATED WORK

2.1 Heart, ECG, and Arrhythmias

The contraction and relaxation of the heart are independent of the nervous system, where the nervous system controls the increase and decrease of the heart rate [23]. The heart has four chambers: left atrium, left ventricle, right atrium, and right ventricle (Figure 1). Electrical activity starts at the sinoatrial (SA) node and propagates through the heart muscle [16]. First, depolarization starts at the SA node, seen in Figure 1, and the rapid conduction of depolarization occurs through the atria (upper two heart chambers) [23]. This wave of depolarization moves through the internodal pathways, causing atrial contraction. Next, depolarization slows down at the atrioventricular (AV) node, resulting in a delay between the atrial and ventricular contractions, allowing blood to flow from the atria into the ventricles (lower two heart chambers) [23]. The depolarization wave then moves through the Bundle of His, left and right Bundle Branches and Purkinje fibers, where the ventricles depolarize and contract starting from the apex (bottom of the heart) and moving towards the systematic and pulmonary circulation (top, back of the heart). The heart relaxes and this cycle repeats itself.

The electrical activity from the heart creates electrical currents on the surface of the body that fluctuate the electrical potential of the skin, which could be detected by pairs of leads placed on the body and recorded as a pattern by an electrocardiogram (ECG) [16, 23]. The 12-lead ECG uses 10 electrodes placed on the chest, arms, and legs. The difference in electrical potential between these leads helps construct the ECG signal [16, 23]. The waves shown by an ECG correspond to the electrical activity of the heart: atrial depolarization (P wave), ventricular depolarization (QRS complex wave) and ventricular repolarization (T wave) [16, 23]. Figure 1 shows a normal heartbeat pattern with a P wave, QRS complex, and T wave [23]. The P wave shows the contraction of atria (depolarization) while atrial relaxation (repolarization) is masked by the QRS complex [23]. The QRS complex shows ventricular contraction (depolarization) and is large due to the ventricles having larger tissue mass to pump blood to the lungs and body [23]. Lastly, the T wave shows ventricular relaxation (repolarization) [23].

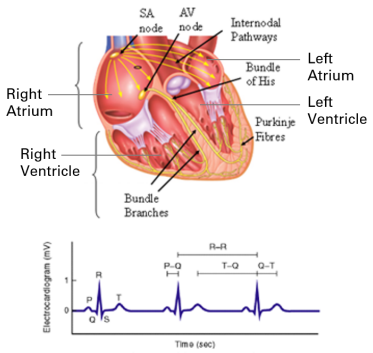


Figure 1: Anatomy of the Heart and ECG for Normal Heart Rhythm [23]

However, the presence of arrhythmias can drastically change these wave patterns [16]. Arrhythmia varies depending on its rate, origin, and other characteristics. Heart rates can be slow (bradycardia), normal, or fast (tachycardia) [23]. It can originate from the SA nodes, AV nodes, supraventricular, or ventricular regions. The heart size can also affect heart rates, like in hypertrophy, where the heart is too large, which could be atrial or ventricular. Heart diseases affecting coronary circulation include ischemia, injury, and infarction [23]. Different arrhythmias vary in its speed, origin, and severity from asymptomatic to life-threatening. Therefore, differentiating types of arrhythmia from normal heart rates in ECG recordings using multi-class machine learning models to detect specific arrhythmias is vital for early diagnoses, preventative treatments, interventions, and medications for arrhythmic patients.

2.2 Related Work

In 1997, Güvenir et al. developed a classification algorithm, the Voting Feature Intervals (VFI5), to distinguish regular heart rate and different arrhythmias from ECG recording measurements in the Arrhythmia dataset [8]. The VFI5 obtained 62% and 68% (with feature weights) accuracies, performing better than Naïve Bayesian (NB) and Nearest Neighbor (kNN) classifiers with 50% and 53% accuracies [8]. However, these accuracies can be considered low in the present-day, and other performance measures, like sensitivity, specificity, and AUC, were not reported. Our study also uses this Arrhythmia dataset [5, 8].

With recent technological advancement and more health monitoring devices, larger volumes of ECG data with high dimensionality are being collected from homes and hospitals [20]. These large volumes of data require efficient and accurate detection of arrhythmias and arrhythmic events from ECG recordings and using machine learning algorithms could help physicians and cardiologists better diagnose arrhythmia efficiently [20]. High-dimensional datasets are standard in the biomedical and healthcare fields and often include irrelevant and redundant features for class prediction, increasing computational costs and decreasing performance of predictive models [2, 9]. Feature selection methods retain data interpretability without transforming the reduced original features and have shorter computational time. In contrast, feature extraction transforms original features to a reduced number of new features and has higher discriminatory power but loses the original data interpretability and is computationally more expensive [9]. Studies investigated feature selection methods for binary classification using the Arrhythmia dataset, such as the contribution-selection algorithm (CSA) with backward elimination that resulted in a 21-feature subset with 84% accuracy and grafting algorithm with 75% accuracy [4, 21]. Although these studies used binary classification, our paper investigates feature selection techniques to choose a feature subset for multi-class classification with 13 classes.

Feature selection includes filter, wrapper, and hybrid methods. Information Gain (IG) and Recursive Feature Elimination-Random Forest (RFE-RF) perform well among the filter and wrapper feature selection techniques, respectively [10]. These two methods are applied to the Arrhythmia dataset to help find a proper feature subset to train prediction models. Hybrid, like embedded, methods tend to perform computationally better than wrapper techniques [2, 9, 10]. AutoSpearman is a hybrid method that removes correlated variables while retaining information about the class [10]. It improved classifiers' accuracy by 1-2% and was highly consistent for software defects metrics [10]. As AutoSpearman has not yet been applied to healthcare, our study investigates its effectiveness for a high-dimensionality reduction in arrhythmia.

Parvaneh et al. found studies using deep learning algorithms for predicting arrhythmia had often not reported computational efficiencies of the models and model interpretability had not been investigated, which are essential as more complex models often require more computational time and are more difficult to interpret [13, 14, 20]. Also, traditional supervised models often perform well but can be overlooked. Lessmann et al. compared supervised learning algorithms, performance measures and statistical hypothesis tests for credit scoring amidst the current technical advancements [13]. Dynamic ensemble classifiers predicted less accurately than simple models, multiple classifiers obtained high accuracy, and Random Forest (RF) performed well [13, 14]. Li et al. updated this study, benchmarking 17 classification models using 27 datasets and focusing on class distribution sampling, performance measures and testing procedures for software defect prediction [14]. RF was found to be one of the top-performing classifiers, NB as one of the lowest, and kNN was found among the top ten performing classifiers for software defect prediction [14].

Moreover, Li et al. emphasized the importance of model comprehensibility as well [14]. Accurate interpretations of a model's predictions enable users to trust the prediction model, discover insights into improving the model, and understand the process behind a model [15]. Machine learning interpretation methods are applied to interpret black box models [18]. Model-specific methods rely on the inner workings of prediction models and thus, different methods are required to interpret different types of prediction models; whereas, model-agnostic interpretation methods are flexible and can be applied on different types of prediction models as it only depends on the data and prediction function [18]. Therefore, model-agnostic techniques separate the type of interpretable model from the type of prediction models [18]. While global surrogate interpretable models are applied to approximate a model's predictions, local surrogate interpretable models interpret a model's individual predictions [18].

Local interpretable model-agnostic explanations (LIME) is a method applying local surrogate models to explain a model's individual predictions [18]. These models ensure good local approximation but not global approximation [18]. LIME's advantages are: one explanation model can be applied on different prediction models, it is easy to use, and it can be trained with interpretable features that are different from the ones used in training the prediction model [18]. Although LIME's additivity property also makes this method advantageous, its disadvantages include: the local neighbourhood of a data point is unclear, the unstable and inconsistent explanations in which two close data points can have widely varying explanations, repeating sampling could result in different explanations, and missing features should have an attribution value of 0 [15, 18]. This makes it difficult to trust its explanations. SHAP overcomes these shortcomings by incorporating LIME with Shapley value from coalitional game theory that calculates features' marginal contributions to the difference between a machine learning model's individual prediction and the average prediction for the dataset [6, 15, 18]. SHAP allows for contrasting and full explanations, its efficiency property ensures the difference between a prediction and the average prediction is fairly distributed among feature values that is not found with LIME, it provides reliable and reasonable explanations based on game theory and has efficiency, symmetry, dummy, and additivity properties [15, 18]. Shapley value satisfies all three properties, local accuracy, missingness, and consistency; however, methods not based on Shapley value are not guaranteed to fulfill all these properties [6, 15, 18]. El Mokhtari et al. employed SHAP interpretation method and SHAP values to find the features that most contributed in the prediction of commentaries with financial time series data and evaluated the impact of additional datasets on models' performances [6]. They compared kNN, RF, Support Vector Machine (SVM), XGBoost, and Long Short-term Memory Network (LSTM) and found binary kNN classifier outperformed in terms of F1-score [6]. It was found the additional Point of Sales (POS) dataset had not improved the classification models' performances that used the VAR, discrepancy dataset [6]. Moreover, using SHAP values as a data transformation method allowed for a natural clustering that helped in the models' accuracy [6].

Krishna used a project's data that provided the best predictions on other projects' data as 'bellwethers' to mitigate conclusion instability in software analytics [11]. The simple technique of using 'bellwethers' for transfer learners provided comparable predictions

to other transfer learning methods and provided stable conclusions [11]. He described a 'bellwether effect' as a 'bellwether' (an exemplary project existing within the historical dataset) to use in training an accurate prediction model [11]. 'Bellwether' finds reliable data and does not restrict leveraging the full benefits of the model [11]. Models trained from specialized regions within a dataset sometimes performed better than those trained across all data [11]. Highly competitive performances were obtained using the 'bellwether' dataset, RF, and four evaluation metrics: accuracy, recall, precision, and F1 score [11]. In addition, effect-size tests were employed to ensure that differences were not due to small effects [11]. Kudjo et al. developed an algorithm using X-means clustering algorithm and mean absolute error to select 'bellwether,' an exemplar training set, for improved software quality's vulnerability severity prediction using four models, deep neural network, logistic regression, kNN, and RF, and evaluation metrics included precision, recall, and F1 score [12]. The 'bellwether' approach showed improved performance compared to benchmark techniques having 14.3-97.8% for F1 score [12]. In this paper, Shapley values for all instances in the Arrhythmia dataset, all classification model's predictions, and selected features were used to find and choose a set of 'bellwether' instances for a training set for the classification model using a minimum absolute Shapley value threshold.

3 METHODOLOGY

3.1 Dataset

The Arrhythmia dataset is open source and publicly available from the UCI Machine Learning Repository [5] [8]. This dataset contains 279 features, an arrhythmia class attribute, and 452 observations, where each observation represents a patient record and its class determined by an expert cardiologist [8]. The features include: age, height, weight, heart rate and 12-lead ECG recording measurements of the patients using the IBM-Mt. Sinai Hospital program[4].

This multi-class arrhythmia class contains 13 classes (three additional classes had no instances and were not included): 'normal', coronary artery disease (CAD), old anterior myocardial infarction (OAMI), old inferior myocardial infarction (OIMI), sinus tachycardy (ST), sinus bradycardy (SB), ventricular premature contraction (VPC), supraventricular premature contraction (SVPC), left bundle branch block (LBBB), right bundle branch block (RBBB), left ventricule hypertrophy (LVH), atrial fibrillation (AFib), and 'other'.

3.2 Data Preparation

First, the Arrhythmia dataset was randomly partitioned into training and test sets using 70:30 ratio. This ensured that the test set was independent of the training set [22]. This was performed for 10 iterations using different seed numbers. Missing values were imputed using medians and the kNN algorithm in a donor-based imputation technique [25]. Zero-variance variables were also removed.

3.3 Dimensionality Reduction

To retain human interpretability for our prediction models, feature selection was chosen over feature extraction [2, 9]. Different feature selection techniques were combined to incorporate three different perspectives in selecting features to predict the arrhythmia class [2, 9, 10]. Three feature selection techniques were applied to the training set: IG, RFE-RF, and AutoSpearman. The top features with the highest information gain coefficients, the wrapper subset with the highest accuracy, and features chosen by AutoSpearman were found. Common features were chosen between: 1) IG and RFE-RF, 2) IG and AutoSpearman, 3) RFE-RF and AutoSpearman, and 4) IG, RFE-RF, and AutoSpearman. A feature subset was selected with all common features among the three techniques and top common features between each pair of techniques.

3.4 Class Imbalance

The arrhythmia class in the dataset was imbalanced. For a model to learn to better differentiate and classify the minority arrhythmia classes as well as correctly predict the majority 'normal' class, Synthetic Minority Over-sampling Technique (SMOTE) was applied on the training set [1, 3]. SMOTE was used for oversampling all minority classes with synthetic examples and undersampling the majority class [3]. Using different parameter values from the defaults for the SMOTE function improved performance measures of the models [1]. Therefore, SMOTE was applied with different sampling parameter values.

3.5 Training and Testing Classification Models

Figure 2 is a flowchart showing the three following steps: building classification models, model interpretation, and improving classification models with 'bellwether' training set.

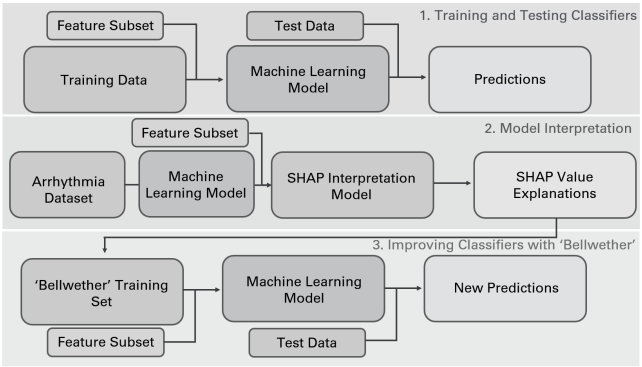


Figure 2: Building, Interpreting, and Improving Classification Models Flowchart

Four supervised learning classifiers, Naïve Bayes (NB), k-Nearest Neighbors (kNN), Random Forest (RF) and a heterogeneous stacking ensemble (Ensemble), were trained to predict normal heart rhythm and types of arrhythmia using 10-fold cross-validation on the training set. 10-fold cross-validation ensured that there was no overlapping of data used for learning and validation in the same runs [22]. More advanced models often require more computational time and are more difficult to interpret [13, 14, 20]. Also, traditional supervised models often perform well but can be overlooked. Classification models could be categorized into six main classes based on their underlying approaches: Bayesian, tree-based, support vector machine, neural network, boosting and other [14]. To study how different machine learning algorithms act on the

[4]https://archive.ics.uci.edu/ml/datasets/Arrhythmia

Arrhythmia dataset, different types of classification models were selected. Like Güvenir et al., this paper also explores the traditional NB and kNN models. RF was found to be one of the top-performing classifiers [13, 14]. Homogeneous ensemble models combine classifiers that act similarly as base models; whereas, heterogeneous ensemble models combine classifiers that perform differently as base models that have different perspectives on the same data and are complementary [13, 14]. Therefore, the simpler traditional NB and kNN models, top-performing homogeneous ensemble RF, and a stacking heterogeneous ensemble (Ensemble) built using NB, kNN, and RF algorithms were compared in the prediction of arrhythmia.

Using the 'caret' package [7], the optimal tuning parameters found for NB model were 'usekernel' = TRUE for non-parametric distribution and 'adjust' = 1 for bandwidth adjustment. Changing 'fL' for Laplace correction did not change the model's performance and was set to 1. For kNN model, it was necessary to preprocess the data using "center" and "scale" to reduce the more significant effect of large-value features on the predictions. The optimal tuning parameter for the number of neighbours was 'k' = 5. For the RF model, the optimal tuning parameter for the number of randomly selected predictors was 'mtry' = 2. The same optimal parameters were found for Ensemble's base RF model and the 'mtry' for the meta RF model was 'mtry' = 2 or 3. The trained classification models were then tested using the test set that was initially separated from the training set in data preparation.

To build the Ensemble model, the training set was split with a ratio of 50:50 for training and validation sets. The base models were trained using the training set and their predictions were made on the validation and test sets. The meta model using RF algorithm was trained using these predictions and the actual class of the validation set. Lastly, the meta model made predictions using the base models' predictions on the test set.

3.6 SHAP Value Explanations for Classification Models

Partial Dependence Plot (PDP) is a global interpretable model that illustrates the marginal effects of one or two features of a model's predictions [18]. Although heterogeneous effects in features are not distinguished using PDPs, Individual Conditional Expectation (ICE) curves and Accumulated Local Effects (ALE) plots find individual predictions of a classification model and heterogeneous effects become apparent [18]. However, ICE curves only show one feature and ALE plots show the differences in predictions, where a higher number of intervals produce less accurate explanations [18]. LIME uses local interpretable models for individual explanations and is additive [15, 18]. SHAP combines LIME and Shapley value to explain individual predictions of models. SHAP was applied to interpret our prediction models locally for individual predictions and globally for arrhythmia class subsets because the approximated Shapley values could be combined into global explanations [15, 18].

For each model, SHAP was run for each instance in the Arrhythmia dataset to obtain a data frame of SHAP values, feature values, prediction probabilities for every data instance, feature, and class. The Shapley value comes from coalitional game theory and aims to equitably distribute the payout among players in a game [18]. Here, the 'game' is the prediction of an instance, the 'payout' is the difference between the prediction of the arrhythmia classes minus the average predictions for the dataset, and the 'players' are the features used to train our classification models. Therefore, we want to determine how fairly distributed the contributions of the feature values were in predicting the arrhythmia class for all the applied classification models. For Ensemble, the 'features' were the base models' predictions and its interpretability explains how fairly distributed the contributions of the base NB, kNN and RF models' prediction values were in the prediction of the arrhythmia class. SHAP values were used to provide an approximation of Shapley values for models and observations using the R package 'iml' [19].

For each model and instance, SHAP values were calculated for the selected features for the 10 iterations performed. The parameter for the number of Monte Carlo samples was set to 20 to estimate the Shapley values for more efficient computation. All the SHAP values were combined into a data frame for each model. Since the meta RF model in Ensemble makes a final prediction for the arrhythmia classes using the base NB, kNN, and RF models' predictions, SHAP used these base models' predictions as 'features' for Ensemble model [19]. SHAP Feature Importance Plot uses the magnitude of feature attributions [18]. For Feature Importance Plots, we plotted the calculated mean SHAP values grouped by class and feature for the dataset and then features were ordered by decreasing importance [18]. SHAP Summary Plots illustrated the feature importance and feature effects [18]. Each point depicted the SHAP value for a feature and instance, where its colour shows its feature value from high to low [18]. Features were sorted by decreasing importance [18]. SHAP Dependence Plots illustrate the feature values and their corresponding SHAP values [18].

3.7 Finding 'Bellwether' Training Set to Improve Classification Models

Using the SHAP values data frame from the previous step for each classifier and selected features to train the models, a minimum absolute SHAP value threshold was chosen aimed in selecting around 70% of the dataset (or 323 instances) as 'bellwether' instances to retrain and improve each classification model. The maximum number of 'bellwether' instances chosen was 403. Because SHAP values determined the contribution of every feature for the individual predictions in each classification model, 'bellwether' instances were chosen for each model's new 'bellwether' training set.

SMOTE was applied to these 'bellwether' training sets with the same parameters and feature subset as before from feature selection and class imbalance to train the new classification models. The four applied classification models, NB, kNN, RF and Ensemble were retrained using these new 'bellwether' training sets to predict the arrhythmia class using 10-fold cross-validation. These models were evaluated using the same initial test sets that were partitioned during data preparation. This was performed for the 10 iterations.

3.8 Evaluation of Classification Models

To include multiple perspectives and to reduce potential bias, the models using the original and 'bellwether' training sets were evaluated using the following performance measures: accuracy, AUC, macro specificity, macro precision, macro recall, macro F1 score,

and Cliff's δ effect size. Accuracy ((TP+TN) / (TP+TN+FP+FP)) measures how accurately a classifier correctly predicts a class. The AUC measures recall (or sensitivity) (TP rate) versus 1-specificity (FP rate) for the classifiers. Macro precision (TP / (TP +FP)) is the overall number of correctly classified positive instances divided by the number of instances labelled as positive by the classifier. This is the percentage of instances predicted as normal or any arrhythmia class that were correctly predicted. Recall (TP / (TP + FN)) is the number of correctly predicted positive instances divided by the number of actual positive instances. Therefore, higher recalls have lower false negatives. F1 score is the harmonic mean of precision and recall and is effective when working with imbalanced datasets [12].

The Cliff's δ effect size from 'effsize' R package was used to measure how often the accuracy of one classification model was more significant than another model [17, 24]. Cliff's δ effect size is a non-parametric measure that estimates the magnitude of significant practical differences and determines the overlap between two groups [12]. It is an accurate and reliable measure. The magnitude thresholds for Cliff's δ is found in Table 1 [12].

Table 1: Cliff's δ Effect Sizes and Magnitude Thresholds [12].

Absolute Cliff's δ Effect Size	Magnitude Thresholds
$\delta < 0.112$	negligible
$0.112 \leq \delta < 0.276$	small
$0.276 \leq \delta < 0.427$	medium
$\delta \geq 0.427$	large

Cliff's δ effect size statistically checks whether the classifiers behave similarly to determine whether NB, kNN, and RF perform differently enough to incorporate into a stacking heterogeneous ensemble classification model. Additionally, the efficiency and stability of the classifiers were compared. Efficiency ensures classifiers perform at a reasonable time, especially considering the increasingly available ECG data. Increased stability of models provides more robust, reproducible, and accurate results, all important in diagnosing patients and ensuring classifiers do not overfit to the training set.

4 RESULTS

4.1 Feature Subsets

Using IG, RFE-RF, and AutoSpearman feature selection techniques resulted in feature subsets that varied across the 10 iterations. However, 14 features consistently appeared in the feature subsets for more than 5 of the 10 iterations: heart rate (bpm), AVF's average width of Q wave (msec), V3's number of intrinsic deflections, V1's area under the QRS complex (msec·mV), average QRS duration (msec), average duration between onset of Q and offset of T waves (msec), V6's amplitude of T wave (mV), average duration between two consecutive T waves (msec), V1's number of intrinsic deflections, V3's amplitude of R wave (mV), V4's amplitude of T wave (mV), AVR's amplitude of T wave (mV), V3's amplitude of the Q wave (mV), and V3's average width of the S wave (msec).

The recurrence of these features in the feature subsets suggests that V1, V3, V4, V6, AVF, and AVR leads helped predict normal heart rhythm and different types of arrhythmia. Additionally, the widths of Q and S waves and amplitudes of T, R, and Q waves were most common in the feature subsets. Heart rate, lead V1's area under QRS complex (msec·mV), and leads V1's and V3's number of intrinsic deflections were also often found as useful in predicting typical and different types of arrhythmia. Features involving leads V2, DII, AVL, V5, and DI, widths of R, Q, R', and S waves, and amplitudes of R', R, T, and S waves appeared less frequently in these subsets. Luz et al. mentioned that R-R intervals, amplitude and width of the T wave, and lead II were more important features in diagnosing cardiac diseases while leads V1, V2, and V4 were favoured for classifying ventricular related arrhythmias [16]. It is seen that features using leads V1 and V4 and amplitude of T waves were also found significant in our feature subsets for arrhythmia prediction.

4.2 Evaluation of Classification Models

Comparing the models' median accuracies over 10 iterations, RF performed the highest with 75.97% accuracy, Ensemble performed the second highest with 72.87% accuracy, NB had 67.83% accuracy, and kNN performed the lowest with 52.72% accuracy, as shown in Figure 5. However, RF and Ensemble models had statistically significant accuracies with p-values ($p < 0.0001$) for all iterations. The median AUCs shown in Table 3 were 0.7957 for NB, 0.7895 for RF, 0.7869 for Ensemble, and 0.7594 for kNN, where NB, RF, and Ensemble performed similarly and kNN slightly underperformed in terms of AUC.

Table 2: Cliff's δ Effect Sizes and the Correlation Coefficients Between the Pairs of Classifiers.

Classifier	Cliff's δ Estimate	Correlation Coefficient
NB-kNN	0.96	-0.2057
NB-RF	-1.00	-0.0149
kNN-RF	-1.00	0.0994
NB-En	-0.92	0.2335
kNN-En	-1.00	0.3878
RF-En	0.96	-0.5177

The absolute Cliff's δ estimates were very large (confidence level = 0.95) as shown in Table 2, indicating the classification models' accuracy distributions showed no to minimal overlap according to Kudjo et al. (Table 1) [12]. Since NB, kNN, and RF models' accuracy distributions show no to minimal overlap, this highlights a heterogeneous ensemble using the mentioned models could capture varying aspects of the data to increase its performance. However, we observe in this study that RF slightly outperformed Ensemble.

In terms of efficiency, the fastest classification model was kNN, then NB, and both RF and Ensemble had longer training times (Figure 6) (Microsoft Surface 2 Laptop with 16GB memory and Intel(R) Core(TM) i7 processor, R version 3.6.0 running on Windows 10). Figure 7 illustrates while NB and kNN performed the least consistently, RF and Ensemble performed the most consistently and therefore, were the most stable for arrhythmia prediction.

4.3 SHAP Value Explanations for Classification Models

Feature Importance Plots illustrate the average contributions of features to the prediction of the arrhythmia classes. Figure 3 shows Feature Importance Plots for the classifiers for the first iteration.

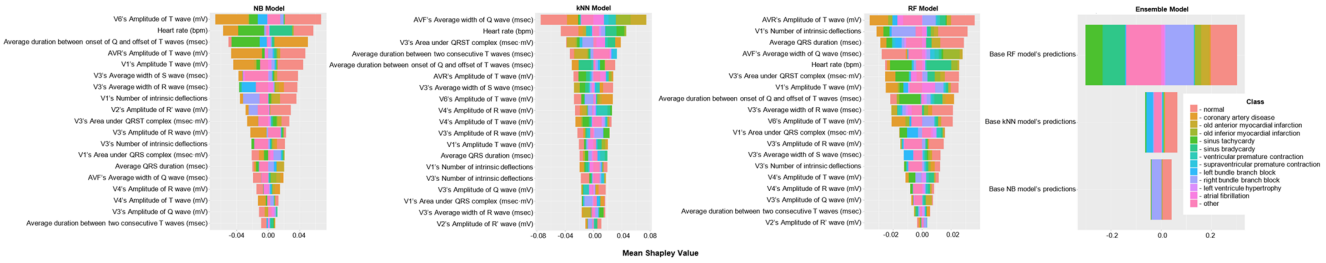


Figure 3: Feature Importance Bar Plots for NB, kNN, RF, and Ensemble Classifiers for Iteration 1.

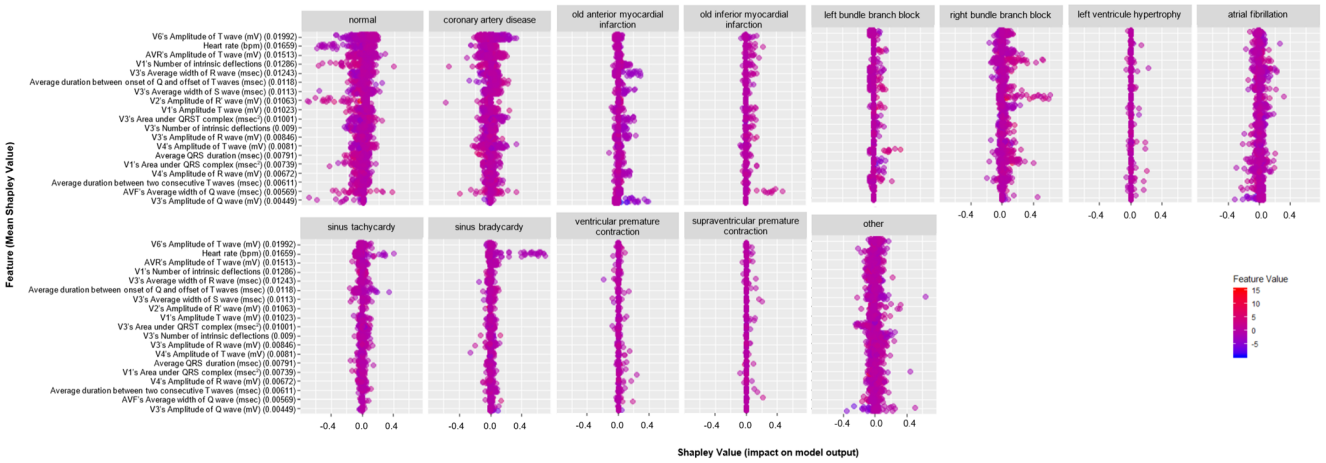


Figure 4: SHAP Summary Plot for NB Classifier for Iteration 1.

Concentrating on the 14 most frequently appearing features in the final feature subsets across the 10 iterations, heart rate (bpm) increased the prediction probability of 'other' class in NB model, 'sinus tachycardy' (ST) and 'sinus bradycardy' (SB) in kNN model, and 'normal' and 'coronary artery disease' (CAD) in RF model, while decreasing the prediction probability for CAD in NB and kNN models, and additionally 'normal', 'ventricular premature contraction' (VPC), and 'other' in kNN model, and often affecting 'normal' and 'atrial fibrillation' (AFib) in NB model, ST and SB in NB and RF models, and OIMI in all NB, kNN, and RF models. Average QRS duration (msec) often increased prediction of 'left bundle branch block' (LBBB), 'right bundle branch block' (RBBB), and 'other' in NB, VPC in kNN, and ST in RF, while decreasing the prediction probability of 'normal' in NB, and often influencing prediction of 'normal' in kNN and RF, and CAD, LBBB, RBBB, and 'other' in RF. Lead V1's area under QRS complex (msec·mV) increased prediction probability for ST in NB, RBBB in NB and kNN, and LBBB and 'other' in NB and RF. It often decreased prediction for 'normal' in NB, and 'other' in kNN while affecting predictions for AFib in NB, 'normal', SB, and RBBB in RF, and CAD in both NB and RF. The average duration between onset of Q and offset of T waves (msec) often increased probability for SB and VPC in NB, and ST in both NB and kNN, decreased prediction for 'other' in NB, SB in kNN, and 'normal' in both NB and RF, while influencing the predictions of CAD, LBBB, and AFib in NB, 'normal' in kNN, ST and SB in RF. The average duration between two consecutive T waves (msec) often increased the prediction probability of CAD and decreased prediction probability of 'normal' in NB and RF while often affecting the prediction probability of 'other' in kNN.

Lead AVF's average width of Q wave (msec) increased the prediction probability of OIMI in all models, decreased the prediction probability for 'other' in kNN and 'normal' in both NB and RF, and often influenced the predictions of 'normal' and VPC in kNN. V3's average width of S wave (msec) often increased prediction probability of OAMI in RF and affected prediction for 'normal' in NB. V6's amplitude of T wave (mV) often increased the prediction probability of 'normal' and OIMI in RF and affected the probability of CAD in all these models, 'normal' and SB in NB, and 'normal', VPC, RBBB, and 'other' in kNN. Lead V3's amplitude of R wave (mV) increased the prediction probability of OAMI in NB and RF, decreased 'other' in NB and 'normal' in kNN, and affected the probability of 'other' and 'normal' in RF. V4's amplitude of T wave (mV) often increased the prediction probability of 'normal' and CAD in NB, CAD, VPC, and RBBB in kNN, decreased the probability of RBBB in RF while affecting the probability of 'normal' and 'other' in kNN and RF and additionally CAD in RF. AVR's amplitude of T wave (mV) increased the probability of 'normal' and CAD in NB and CAD, VPC, and RBBB in kNN, decreased the probability of AFib in NB, while affecting CAD in NB, VPC and 'other' in kNN and 'normal', CAD, and 'other' in RF. V3's amplitude of Q wave (mV) increased the probability of CAD in NB and OAMI in RF while decreasing the probability of 'normal' in both NB and RF.

Lead V3's number of intrinsic deflections increased the prediction of OAMI in all models, CAD in both NB and RF, SB in kNN and

decreased the probability of SB in NB, 'normal' in kNN, AFib in RF while affecting 'normal' and 'other' in NB and RF, and RBBB in RF. V1's number of intrinsic deflections increased the probability of 'normal' in NB while decreasing the probability of RBBB in RF and often influencing the probability of OAMI and SB in NB, 'normal' in kNN and RF, AFib and 'other' in NB and RF.

For Ensemble, base NB's predictions often increased the prediction probability of CAD, decreased the probability of RBBB, and influenced the probability for 'normal'. Base kNN's predictions often increased the prediction probability for 'normal', CAD, and OAMI, decreasing the prediction for ST, SB, and 'other'. Base RF's predictions increased the prediction probability of 'normal' and OAMI similar to base kNN model, CAD similar to the base NB and kNN models, while decreasing the probability of RBBB similar to base NB model and AFib, and affecting the probability of OIMI, ST, SB, and 'other'. Overall, SHAP value explanations for the classifiers' predictions highlighted that different classification algorithms used different features from the feature subset to make predictions for the arrhythmia class. The SHAP values showed more significant contributions of features in NB, then kNN while RF had lower SHAP values. However, RF often affected many of the arrhythmia class predictions compared to NB and kNN.

SHAP Dependence Plot displays all data instances' feature values and its SHAP values for a feature. For example, for V6's amplitude of T wave (mV) in the NB model, there were nonlinear relationships between the SHAP values and feature values. Values between -1.5 to -3.1mV decreased the prediction probability for 'normal' class while values higher than 1.5mV increased its probability. Values less than 0mV increased the prediction probability for CAD; whereas, values greater than 0mV decreased its probability. 0 to 1.5mV values increased the prediction of ST and fluctuations were seen for 'other'. Interestingly, heart rate (bpm) only ranged from 0 to 65bpm in the Arrhythmia dataset, indicating it as erroneous because the normal heart rate is around 60bpm. However, the models appeared to have found patterns within this feature to predict 'normal' and different arrhythmias. For example, for NB, values less than 30bpm decreased the prediction probability of 'normal'. 0 to 17bpm sometimes increased prediction probability for ST, indicating some of these values could have been heart rates higher than 65bpm but possible constraints could have inhibited the recordings of these higher values. 15 to 25bpm increased the prediction probability of SB. Similar patterns were found for the RF model. For kNN, less than 30bpm showed decreased prediction probability of 'normal'. Values less than 17bpm increased prediction of ST and 17 to 30bpm increased prediction of SB. The same patterns were found for the same models and features across 10 iterations, depicting SHAP found consistent patterns in the models' predictions. Features could contain errors in the Arrhythmia dataset, as seen with heart rate (bpm), and could be determined with domain knowledge; however, finding better quality data subsets within this dataset could also reduce the effects of erroneous data present in a dataset.

SHAP Summary Plots show the distribution of SHAP values for the features and feature values for each model and class. For example, looking at Figure 4 for the first iteration of NB model, lower V6's amplitude of T wave (mV) values, lower heart rate (bpm) values, higher V1's number of intrinsic deflections values, and slightly higher V2's amplitude of R' wave values decreased the prediction probability for 'normal' class. Lower V6's amplitude of T wave (mV) also increased prediction probability of CAD with around 0.2 SHAP value. Lower values of V3's average width of R wave (msec) and V3's amplitude of Q wave (mV) increased prediction probability of OAMI with around 0.2 SHAP value. Lower heart rate (bpm) values increased prediction for ST and SB with SHAP values around 0.2 and 0.4, respectively. The lower average duration between onset Q and offset T waves (msec) increased prediction for ST. Higher average QRS duration (msec) values, lower V1's area under QRS complex (msec·mV), and lower V4's amplitude of R wave (mV) values increase prediction probability for LBBB with about 0.1 SHAP values. Higher V1's number of intrinsic deflections, V2's amplitude of R' wave (mV) values, and slightly higher V1's area under QRS complex (msec·mV) increase probability prediction for RBBB with $\geq$ 0.2 SHAP values. Lower values for V3's average width of S wave (msec) and lower V3's number of intrinsic deflections values increased prediction of other with about 0.2 SHAP value while lower values of V3's amplitude of Q wave (mV) decreased the prediction probability for other with around 0.2 SHAP value.

4.4 Evaluation of Improved Classification Models Using 'Bellwether' Training Instances

In terms of accuracy, RF+BW had the highest accuracy of 91.09%, followed by Ensemble+BW with 86.44%, NB+BW with 78.30%, RF with 75.97%, Ensemble with 72.87%, NB with 67.83%, kNN+BW with 62.79% and lastly, kNN with 52.72% as seen in Figure 5. As summarized in Table 3, using the 'bellwether' (BW) training set for the classification algorithms resulted in at least 10% improvement in accuracy for the models. RF+BW again had the highest AUC of 0.9415, second Ensemble+BW with 0.9065, closely followed by kNN+BW with 0.9041, NB+BW with 0.8644, NB with 0.7957, RF with 0.7895, Ensemble with 0.7869 and kNN with 0.7594. Interestingly, it was observed that training the models with 'bellwether' instances improved the AUCs of these models and found RF+BW performed well again in terms of AUC while Ensemble+BW and kNN+BW showed good AUCs above 0.90 as well, making kNN+BW appear stronger than NB+BW even though NB had a higher AUC than kNN and the kNN algorithm performed worst in terms of accuracy.

RF+BW and Ensemble+BW had the best macro specificity with 0.9898 and 0.9854, respectively. Macro F1 scores again indicated RF+BW performed the best with an F1 score of 0.9160 and then Ensemble with 0.8592 while RF, NB+BW, Ensemble, kNN+BW, NB, and kNN had 0.7614, 0.7561, 0.7336, 0.7102, 0.6804, and 0.5846 macro F1 socres, respectively. Looking at the macro precision and macro recall, all kNN, NB, RF, and Ensemble have low performances of less than 0.65. However, using 'bellwether' instances helped improve the classification algorithms' performances: RF+BW had 0.8962 precision and 0.9414 recall, Ensemble+BW had 0.8539 precision and 0.8949 recall, NB+BW had 0.7493 precision and 0.8039 recall, and kNN+BW had 0.5883 precision and 0.8862 recall.

Therefore, RF+BW was the top-performing model, followed by Ensemble+BW in arrhythmia prediction in terms of accuracy, AUC, macro specificity, precision, recall, and F1 score. Figure 6 shows the efficiencies of training these models. The fastest model was kNN classification algorithm that had training time under 5 seconds but

Table 3: Evaluation Metrics for the Classification Models Using the Original and 'Bellwether' (BW) Training Sets.

Models	Accuracy	AUC	Macro Specificity	Macro Precision	Macro Recall	Macro F1 score
NB	0.6783	0.7957	0.9630	0.5632	0.4930	0.6804
kNN	0.5272	0.7594	0.9575	0.4085	0.4922	0.5846
RF	0.7597	0.7895	0.9698	0.6425	0.5450	0.7614
Ensemble	0.7287	0.7869	0.9673	0.5749	0.5477	0.7336
NB+BW	0.7830	0.8644	0.9755	0.7493	0.8039	0.7561
kNN+BW	0.6279	0.9041	0.9692	0.5883	0.8862	0.7102
RF+BW	0.9109	0.9415	0.9898	0.8962	0.9414	0.9160
Ensemble+BW	0.8644	0.9065	0.9854	0.8539	0.8949	0.8592

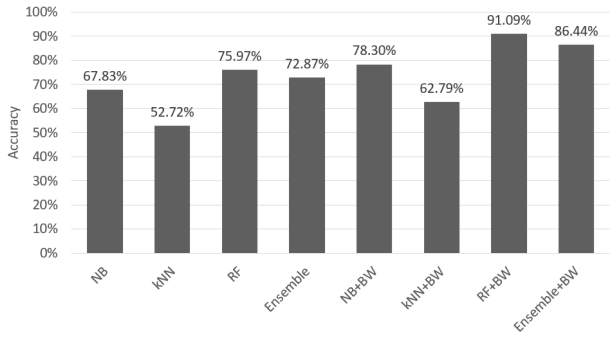


Figure 5: Median Accuracies for NB, kNN, RF, and Ensemble Classification Models Over 10 Iterations.

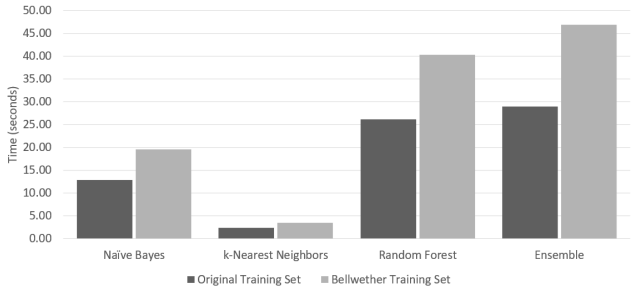


Figure 6: Median Computational Efficiencies of the Original and Improved NB, kNN, RF, and Ensemble Classifiers.

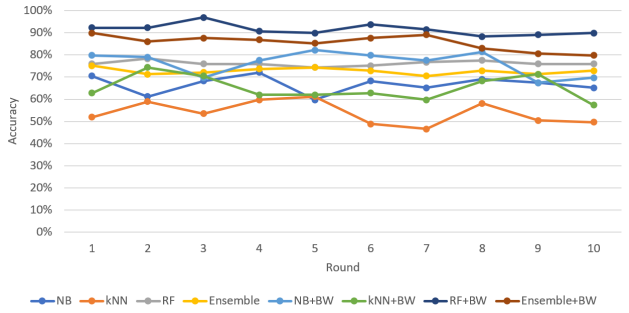


Figure 7: Stability of the Original and 'Bellwether' (BW) Classification Models

all models' training times were under one minute. The testing times for all the models were faster, within a couple of seconds. The most stable classification algorithms were RF and Ensemble, followed by NB and kNN as seen in Figure 7.

The Cliff's δ effect sizes between the new classification models' accuracies were still large, like the original models as seen in Table 4. However, it is evident that the improved NB, kNN, RF, and Ensemble models trained with 'bellwether' instances performed better than original NB, kNN, RF, and Ensemble models with Cliff's δ effect size magnitudes of 0.81, 0.89, 1.00, and 1.00, respectively. Therefore, finding 'bellwether' instances with SHAP values from the data was an effective approach in improving the performance and maintainability of the classification models.

Table 4: Cliff's δ Effect Sizes and the Correlation Coefficients Between the Pairs of Classifiers using Original and 'Bellwether' (BW) Training Sets.

Classifier	Cliff's δ Estimate	Correlation Coefficient
NB –NB+BW	-0.81	-0.1430
kNN –kNN+BW	-0.89	0.3215
RF –RF+BW	-1.00	-0.0976
En –En+BW	-1.00	0.1697
NB+BW –kNN+BW	0.8	-0.1590
NB+BW –RF+BW	-1.00	-0.1441
kNN+BW –RF+BW	-1.00	0.1948
NB+BW –En+BW	-0.90	0.4981
kNN+BW –En+BW	-1.00	-0.0930
RF+BW –En+BW	0.91	0.6136

5 THREATS TO VALIDITY

Internal Threats: The optimization of tuning hyperparameters, such as the maximum depth of trees in RF and Ensemble's base and meta RF models, were not investigated but could further improve these models' performances and highlight more differences among the models. Using SMOTE to oversample minority arrhythmia classes uses synthetic samples based on the dataset, creating potential bias and not accounting for varying samples that exist in reality. *External Threats:* More and current data for arrhythmia with ECG measurements would improve and allow for better assessment of the models, providing more generalizability. This would also provide more varying minority arrhythmia class examples to train the classification models and improve its prediction for these classes. *Construct Threats:* Other classification algorithms could also be compared for arrhythmia prediction. Using SHAP for model interpretability expects features to be not correlated with each other so that feature interactions could be further explored. Additionally, domain experts' knowledge would help validate and further improve feature selection techniques, classification models, and the interpretations of these models.

6 CONCLUSION AND FUTURE WORK

In conclusion, using three different feature selection techniques, IG, RFE-RF, and AutoSpearman, was effective in finding less than 20 features for building arrhythmia prediction models. Güvenir et al.'s VFI5 with feature weights obtained an accuracy of 68% while NB and kNN classifiers had 50% and 53% accuracy[8]. In answering RQ1, kNN classifier also had 53% accuracy (rounding to the nearest percent) in this study. NB classifier performed better with about 68% accuracy while Ensemble and RF had 72.87% and 75.97% accuracy, respectively. However, using the local model-agnostic interpretability method, SHAP, provided insights on how the prediction models predicted the arrhythmia class. To answer RQ2, NB and kNN showed features had higher SHAP values and therefore, more contribution for arrhythmia prediction than in RF model. However, features in RF model affected more arrhythmia classes' predictions than in NB and kNN models. Ensemble model often relied on base RF model's predictions but base NB and kNN models were still used to help make predictions, showing Ensemble still depended on base NB and kNN models in making predictions. SHAP showed classifiers used specific feature values for individual predictions. Domain knowledge would help evaluate whether models' explanations for the predictions were useful and reasonable. Lastly answering RQ3, it was seen that finding 'bellwether' instances using SHAP values improved the classification models by over 10% in terms of accuracy, macro precision, and macro recall. Similarly, kNN, RF, and Ensemble improved by over 10% for AUC and macro F1 score while NB improved by over 5% for these metrics. Using different evaluation metrics helped show the improvement of using 'bellwether' instances for prediction models (Table 3). Specifically, the magnitude of Cliff's δ were greater than 0.80, between the original models and their respective improved models, indicating 'bellwether' instances found using SHAP values were effective in improving arrhythmia prediction models. RF+BW, Ensemble+BW, NB+BW, and kNN+BW showed improved accuracies of 91.09%, 86.44%, 78.30%, and 62.79%, respectively. Additionally, RF+BW was the top-performing model, followed by Ensemble+BW, having 0.9415 and 0.9065 AUC, 0.9898 and 0.9854 specificity, and 0.9160 and 0.8592 macro F1 scores, respectively. In this study, we proposed using SHAP values as a reliable technique for model-agnostic interpretability for machine learning models and selecting 'bellwether' training instances using SHAP values to improve prediction model performances. This could help physicians and clinicians better understand the underlying explanations of models' predictions, allowing them to trust and further improve these models. Future work include comparing different classification models, like Gradient Boosting, in addition to the models from this study for arrhythmia prediction, investigating the effect of using feature selection on the 'bellwether' training set, exploring optimization of hyperparameters in RF and Ensemble models, and obtaining more samples and/or current datasets to better predict arrhythmias to increase the models' generalizability and for further interpretability of machine learning models using SHAP values.

REFERENCES

[1] Amritanshu Agrawal and Tim Menzies. 2018. Is "better data" better than "better data miners"?: On the benefits of tuning SMOTE for defect prediction. In *Proceedings - International Conference on Software Engineering*. 1050–1061.

[2] Raid Alzubi, Naeem Ramzan, Hadeel Alzoubi, and Abbes Amira. 2018. A Hybrid Feature Selection Method for Complex Diseases SNPs. *IEEE Access* 6 (2018), 1292–1301.

[3] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. 2002. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research* 16 (2002), 321–357. arXiv:1106.1813

[4] Shay Cohen, Eytan Ruppin, and Gideon Dror. 2005. Feature selection based on the shapley value. In *IJCAI International Joint Conference on Artificial Intelligence*. 665–670.

[5] Dheeru Dua and Casey Graff. 2019. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[6] Karim El Mokhtari, Ben Peachey Higdon, and Ayşe Başar. 2019. Interpreting Financial Time Series with SHAP Values. In *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering (CASCON '19)*. IBM Corp., USA, 166–172.

[7] Max Kuhn. Contributions from Jed Wing, Steve Weston, Andre Williams, Chris Keefer, Allan Engelhardt, Tony Cooper, Zachary Mayer, Brenton Kenkel, the R Core Team, Michael Benesty, Reynald Lescarbeau, Andrew Ziem, Luca Scrucca, Yuan Tang, Can Candan, and Tyler Hunt. 2019. *caret: Classification and Regression Training*. R package version 6.0-84.

[8] Halil Altay Guvenir, Burak Acar, Gulsen Demiroz, and Ayhan Cekin. 1997. Supervised machine learning algorithm for arrhythmia analysis. *Computers in Cardiology* (1997), 433–436.

[9] Zena M. Hira and Duncan F. Gillies. 2015. A review of feature selection and feature extraction methods applied on microarray data. *Advances in Bioinformatics* 2015 (2015), 1–13.

[10] Jirayus Jiarpakdee, Chakkrit Tantithamthavorn, and Christoph Treude. 2018. Autospearman: Automatically mitigating correlated software metrics for interpreting defect models. In *Proceedings - 2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018*. 92–103.

[11] Rahul Krishna and Tim Menzies. 2019. Bellwethers: A Baseline Method for Transfer Learning. *IEEE Transactions on Software Engineering* 45, 11 (2019), 1081–1105. arXiv:1703.06218

[12] Patrick Kwaku Kudjo, Jinfu Chen, Solomon Mensah, Richard Amankwah, and Christopher Kudjo. 2020. The effect of Bellwether analysis on software vulnerability severity prediction models. *Software Quality Journal* (2020), 1–34.

[13] Stefan Lessmann, Bart Baesens, Hsin Vonn Seow, and Lyn C. Thomas. 2015. Benchmarking state-of-the-art classification algorithms for credit scoring: An update of research. *European Journal of Operational Research* 247, 1 (2015), 124–136.

[14] Libo Li, Stefan Lessmann, and Bart Baesens. 2019. Evaluating Software Defect Prediction Performance: An Updated Benchmarking Study. *SSRN Electronic Journal* (2019).

[15] Scott M. Lundberg and Su In Lee. 2017. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, Vol. 2017-Decem. Neural information processing systems foundation, 4766–4775. arXiv:1705.07874

[16] Eduardo José da S. Luz, William Robson Schwartz, Guillermo Cámara-Chávez, and David Menotti. 2016. ECG-based heartbeat classification for arrhythmia detection: A survey. *Computer Methods and Programs in Biomedicine* 127 (2016), 144–164.

[17] Guillermo Macbeth, Eugenia Razumiejczyk, and Rubén Daniel Ledesma. 2011. Cliff's Delta Calculator: A non-parametric effect size program for two groups of observations. *Universitas Psychologica* 10, 2 (2011), 545–555.

[18] Christoph Molnar. 2019. Interpretable Machine Learning. A Guide for Making Black Box Models Explainable. (2019). https://christophm.github.io/interpretable-ml-book/

[19] Christoph Molnar, Bernd Bischl, and Giuseppe Casalicchio. 2018. iml: An R package for Interpretable Machine Learning. *JOSS* 3, 26 (2018), 786.

[20] Saman Parvaneh, Jonathan Rubin, Saeed Babaeizadeh, and Minnan Xu-Wilson. 2019. Cardiac arrhythmia detection using deep learning: A review. *Journal of Electrocardiology* 57 (2019), 70–74.

[21] Simon Perkins, Kevin Lacker, and James Theiler. 2003. Grafting: Fast, incremental feature selection by gradient descent in function space. *Journal of Machine Learning Research* 3 (2003), 1333–1356.

[22] Payam Refaeilzadeh, Lei Tang, and Huan Liu. 2009. Cross-Validation. In *Encyclopedia of Database Systems* (1 ed.), Ling Liu and M Tamer Özsu (Eds.). Springer US, 532–538.

[23] Dee Unglaub Silverthorn, Bruce R. Johnson, William C. Ober, Claire E. Ober, and Andrew C. Silverthorn. 2016. *Human Physiology: An Integrated Approach* (7 ed.). Pearson Education, San Francisco. 838 pages.

[24] Marco Torchiano. 2019. *effsize: Efficient Effect Size Computation.* R package version 0.7.6.

[25] Luis Torgo. 2010. *Data Mining with R, learning with case studies.* Chapman and Hall/CRC.

Efficient Location-Level Risk Analytics

Neil Burke
neil.burke@dal.ca
Faculty of Computer Science
Dalhousie University
Halifax, Nova Scotia

Oliver Baltzer
oliver@analyzere.com
Analyze Re
Halifax, Nova Scotia

Norbert Zeh
nzeh@cs.dal.ca
Faculty of Computer Science
Dalhousie University
Halifax, Nova Scotia

ABSTRACT

We propose a system for performing risk analytics of reinsurance portfolios at the resolution of individual insured locations. By using a graph-based portfolio representation, our system achieves the flexibility to represent arbitrarily complex reinsurance portfolios. In spite of this flexibility, which is not achieved by current risk analytics systems, neither commercial nor academic ones, our system is substantially faster than current risk analytics systems. Given that such a location-level portfolio analysis involves the processing of terabytes of data, the key to the efficiency of our system is the use of a scalable cloud-based architecture and the careful engineering of the data representation and algorithms to ensure that data processing happens entirely in memory of the compute nodes.

CCS CONCEPTS

• **Applied computing** → **Enterprise applications**; • **Information systems** → **Data analytics**.

KEYWORDS

risk analytics, cloud computing, algorithm engineering

1 INTRODUCTION

Insurance companies sell insurance to property owners and thereby expose themselves to the risk of financial losses when the insured files a claim. Natural disasters, such as earthquakes, floods or hurricanes, can expose an insurance company to *catastrophic* losses that result in the company's bankruptcy or, worse, its inability to reimburse its clients. *Reinsurance* companies act as insurers for insurance companies. Reinsurance *treaties* (contracts) between primary insurers as the insured and a reinsurance company as the insurer protect the primary insurer against such catastrophic losses. This is an industry capitalized at $500 billion per year and annual gross written premiums of more than $260 billion.

Both insurers and reinsurers aim to structure their portfolios of contracts and treaties so that the probability to make a profit—when premiums exceed claim payouts—is (significantly) higher than the probability of a loss—when claim payouts exceed premiums. This does not only ensure the profitability of the company but also reduces the risk that a reinsurance company cannot meet its insurance obligations toward its clients, which in turn reduces the risk that a primary insurer cannot meet its obligations toward its clients—property owners. The ability to model and quantify risk due to natural catastrophes to ensure an insurer's solvency has gained increasing importance over the last 25 years. Prior to the introduction of commercially available catastrophe modelling software, insurers relied on records of infrequent historical claims in order to extrapolate future loss potential [8, 23].

Reinsurance companies employ decision support systems to help with the management of their portfolios. This includes deciding whether to enter into a new treaty with a primary insurer, and on what terms; deciding whether it is safe to invest money or keep cash on hand to respond to seasonal fluctuations in risk exposure, for example due to hurricanes; and many more questions. The core problem is to compute a *loss distribution*, a probability distribution over the potential payouts to clients due to natural disasters.

The portfolio of a typical reinsurance company consists of thousands of reinsurance treaties that can cover several hundred million insured properties. The numbers of treaties and insured locations and the complex interactions between different treaties in the portfolio make it impossible to obtain a closed-form expression describing the loss distribution. Consequently, decision support systems used in the reinsurance industry rely on Monte Carlo simulation to obtain an accurate estimate of this distribution [6].

The estimation of the loss distribution using Monte Carlo simulation is computationally costly and involves the processing of dozens of terabytes of data when analyzing a typical reinsurance portfolio based on the risk exposure of individual insured locations. Performing such a *location-level analysis* on existing commercial risk analytics platforms is either infeasible or involves a combination of manual manipulation of the data with days of computation time (see Section 6.4). To reduce the computation cost, most commercial systems support only less fine-grained analyses based on preaggregated losses at the county or state level. This sacrifices accuracy and makes it impossible to model, for example, (quite common) reinsurance treaties that cover losses due to hurricane damage to properties within a certain distance from the coast.

Thus, there is a need for risk analytics platforms that are capable of performing a location-level risk analysis of a full reinsurance portfolio. Ideally, such a platform should be able to produce the loss distribution of a portfolio within minutes because this helps an underwriter to analyze the impact of a new treaty on the portfolio's risk exposure while negotiating the terms of the treaty. Fine-grained modelling of a portfolio's risk exposure at the level of individual locations also opens the door for detailed tailoring of reinsurance

CASCON'20, Nov 10-13, 2020, Toronto, Canada

treaties, for example by adjusting the coverage for losses incurred due to damage to high-risk locations. To support such fine-tuning, a location-level risk analytics platform should be flexible and support essentially arbitrary treaty terms and interactions between treaties.

In this paper, we propose a location-level risk analytics system that meets both requirements. It achieves the flexibility to model arbitrarily complex reinsurance portfolios by representing a portfolio as a directed acyclic graph structure built from a small number of simple building blocks. We propose a scalable cloud-based parallel evaluation engine that can compute the loss distribution of a typical reinsurance portfolio covering 70 million insured locations in around 30 minutes on 20 compute instances with 48 cores each. This is significantly faster than the performance of current commercial or academic risk analytics systems—many of these systems are unable to perform less fine-grained analyses in 30 minutes. Our system achieves faster performance in spite of its greater flexibility—existing commercial and academic risk analytics systems can manipulate only portfolios of a restricted structure.

Given that performing a portfolio risk analysis via Monte Carlo simulation involves evaluating a large number of independent *trials* (see Section 2), it is natural to parallelize the analysis by assigning different trials to compute nodes that evaluate these trials independently. This is efficient if each compute node can evaluate its assigned trials entirely in memory. Since several gigabytes of data must be processed to evaluate a single trial, this is non-trivial. The key technical contribution of this paper is to demonstrate that the evaluation of the portfolio graph can be organized so that the intermediate results that need to be held in memory use little space. This amounts to finding a topological ordering of the portfolio graph of low cut width (see Section 5). Minimizing the cut width is an NP-hard problem. Our solution employs a heuristic that exploits the structure of typical portfolio graphs to obtain a topological ordering of sufficiently low cut width quickly and then modifies the portfolio graph and its topological ordering to reduce the cut width further without changing the structure of the treaties in the portfolio represented by the graph.

The remainder of this paper is organized as follows: Section 2 gives a brief overview of the use of Monte Carlo simulation to obtain an accurate estimate of a portfolio's loss distribution. Section 3 discusses related work. Section 4 presents our graph-based portfolio representation. Section 5 discusses our implementation of a cloud-based risk analytics system. Section 6 discusses experimental results that demonstrate the performance of our system and presents a comparison against a major vendor's commercial system. We offer concluding remarks and a discussion of future work in Section 7.

2 PORTFOLIO RISK ANALYTICS

Monte Carlo simulation to estimate the loss distribution of a portfolio is based on evaluating a large number of *trials*, typically 10,000 or more. Each trial computes the sequence of payouts to the reinsurer's clients, given a particular sequence of catastrophic events (hurricanes, floods, etc.) in a given year. To compute a probability distribution over the total annual payouts by the reinsurer to its clients, it suffices to add up the total payouts in each trial and analyze the frequency distribution of the total payouts across all trials. More fine-grained analyses are possible that focus on the distribution of losses due to a particular type of peril or on seasonal loss distributions. This can be achieved by aggregating only subsets of loss values in each trial and again analyzing the frequency distribution of these aggregates across all trials.

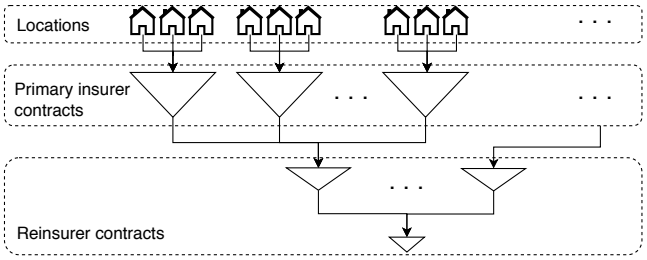


Figure 1: Typical structure of a reinsurance portfolio. Triangles represent embedded structures such as primary insurers' portfolios and the reinsurance treaties acting on them.

Each trial is based on a sequence of catastrophic events, obtained by sampling from catastrophe models developed by seismologists, meteorologists, and other scientists. Structural models developed by engineers are used to translate each event into an estimated amount of damage to each insured property affected by the event, quantified as a monetary *loss value*. The result is one *year event loss table* (YELT) per property (*location*) that records the sequence of losses for this property due to the sequence of events in this trial. Each entry in the YELT, called an *occurrence*, stores the loss value and the type and simulated date of the event that caused it. We call such a YELT recording the losses for one location a *location YELT*.

The input to a *location-level portfolio analysis* is a set of trials. Each trial is represented as a set of *location YELTs*, one per insured location. The payouts to the property owner by the primary insurer due to the losses recorded in each location YELT are determined by the contract between property owner and primary insurer. The totals of these payouts by a primary insurer to its clients in response to the events in the trial constitute the insurer's sequence of losses. Note that this sequence can itself be viewed as a YELT. The primary insurer's losses are covered by a complex network of reinsurance treaties that determine the payouts from the reinsurer to primary insurers. These payouts are the reinsurer's losses, and they form once again a YELT, the *portfolio YELT* for this trial. The portfolio analysis produces one portfolio YELT per trial. The structure of a typical reinsurance portfolio is illustrated in Figure 1.

This paper focuses on computing the portfolio YELTs for all trials from their location YELTs based on the insurance contracts and reinsurance treaties covering these losses. This is the computationally costly part of the analysis as it involves aggregating the losses across hundreds of millions of insured locations, millions of insurance contracts, thousands of reinsurance treaties, and thousands of trials. This requires processing several terabytes of input data. The statistical analysis of the portfolio YELTs to estimate the portfolio's loss distribution is comparatively trivial, as far as computation cost is concerned. We also do not consider the process of producing the input YELTs of each trial. They are provided as the input of the analysis. Commercial risk analytics platforms also start with YELTs representing raw losses as input. These YELTs

are provided by brokers who produce them based on catastrophe models or generated by licensing the necessary models and tools from a vendor.

3 RELATED WORK

3.1 Reinsurance Analytics

Location-level insurance analytics is a mostly unstudied topic.

Academic research on risk analytics systems has focused on creating a distributed risk analytics engine using Hadoop [25] and on using optimizations on specialized hardware to achieve fast, single-node running times [7, 10]. These solutions only support analyses at coarser granularities, with input YELTs aggregated to county or state level, and they assume a very restrictive portfolio structure, to simplify the implementation of efficient solutions. The portfolio is assumed to be composed of a flat list of "programs". Each program is composed of a sequence of "layers" or transformations that apply to all input losses in a user-specified order. The programs' outputs are combined to generate the portfolio's loss distribution. This means that the portfolio is modelled essentially as a directed tree, and that output from one program cannot be passed as input to another within the same analysis. Not only does this make location-level analysis impossible, it also limits the practical use of the system even for coarser-grained analyses, as actual reinsurance portfolios are rarely composed of programs with strictly delineated layers and the interactions between treaties rarely form a tree.

There are a number of commercial risk analytics platforms on the market that implement part of the functionality required to solve the problem of location-level reinsurance analytics.

Catastrophe modelling software [1, 27] can compute the loss distributions of a group of locations up to the primary insurer level, and is therefore typically sold to primary insurers. These systems are designed for modelling a much smaller number of locations than what would be seen in a reinsurance portfolio. The two leading products are built around Microsoft SQL Server as both their data storage and computational platform, and are consequently bound to the scalability limitations of SQL servers [34].

For analytics on reinsurance portfolios, the solutions on the market today [2, 4, 32] are only able to consume data at an aggregated geographic level (e.g., county level). The most flexible of these systems allow the user to "nest" contracts within certain other contracts. This allows for some flexibility in defining simple dependent relationships between contracts, but it does not offer the same flexibility and ease of expressing relationships between contracts as a graph-based portfolio representation.

Dynamic Financial Analysis (DFA) products [5, 14, 26, 33] allow for the modelling of complex cash flows and provide the greatest level of flexibility in terms of structuring and modelling features. However, they only consume data at very coarse levels of detail and are typically limited to a small number of trials.

3.2 Graph Modelling Frameworks

The modelling of complex data flow problems as directed graphs, as we do in our graph-based portfolio representation in Section 4, and the development of distributed systems to evaluate such graph-based data flow representations efficiently, as we do in Section 5, has been the focus of previous work [3, 12, 13, 16, 20–22]. In these systems, a graph represents a complex computation, vertices represent steps in this computation, and edges represent the flow of data from one step to another. Due to their data dependencies, steps connected by edges must be executed in sequence while steps not connected by edges may be executed in parallel. These systems are designed to schedule the steps of a processing pipeline (expressed as a directed graph) across multiple machines while managing communication between compute nodes. In spite of their effectiveness for such problems, their focus on sophisticated scheduling and communication strategies introduces overhead that is unnecessary in the context of portfolio analysis. Since a portfolio analysis consists of running tens of thousands of trials completely independently, the reinsurance analytics problem is trivial to parallelize (barring memory constraints; see Section 5).

Several works also exist on processing large graphs on a single compute node [15, 18, 28, 30, 35, 36]. These systems are more aligned with our reinsurance platform design, as each trial in our analysis is processed on an independent compute node. However, all of these single-machine graph processing systems are focused on iterative processing, where the amount of data flowing across each edge of the graph is relatively small (PageRank [24] is a common benchmark in these papers). Our problem is different in that the data flowing across edges is much larger and not uniform. This makes memory a scarce resource. Indeed, minimizing the amount of working memory necessary to evaluate a single trial is the core challenge we address in Section 5.

4 REINSURANCE PORTFOLIO AS A GRAPH

As described in Section 2, the output of a portfolio analysis is a list of portfolio YELTs, one per trial. Since trials can be evaluated independently, the core of the problem is to compute the portfolio YELT of a single trial from the location YELTs of the trial.

This process can be represented as a directed acyclic graph (DAG). The DAG has one source (vertex without in-neighbours) per location YELT. A single sink (vertex without out-neighbours) represents the output of the computation, that is, the portfolio YELT. Internal vertices represent terms and clauses of contracts and treaties, such as deductibles to be subtracted from claimed losses, the percentage of the remaining losses covered under a treaty or a limit up to which losses are covered. Deductibles and coverage limits may apply to individual claims or to the total of all claims throughout the year. These types of contract terms can be modelled using a small number of vertex types in the graph. Treaties are constructed by combining these vertices into subgraphs that capture which of these transformations apply to which losses and in which order.

In general, every vertex other than the sources and the sink takes one or more YELTs as input and produces a YELT of transformed loss values as its output. This output YELT forms (part of) the input of one or more other vertices.

For many transformations, the order in which occurrences are processed is important. For example, some contractual terms apply only to the first n occurrences. Therefore, YELTs must be in sorted order (by date). For the input YELTs of the portfolio, this can be guaranteed using a one-time preprocessing. To keep intermediate YELTs sorted, each vertex reads its input YELTs in order and writes the transformed occurrences in order. If a vertex has multiple input

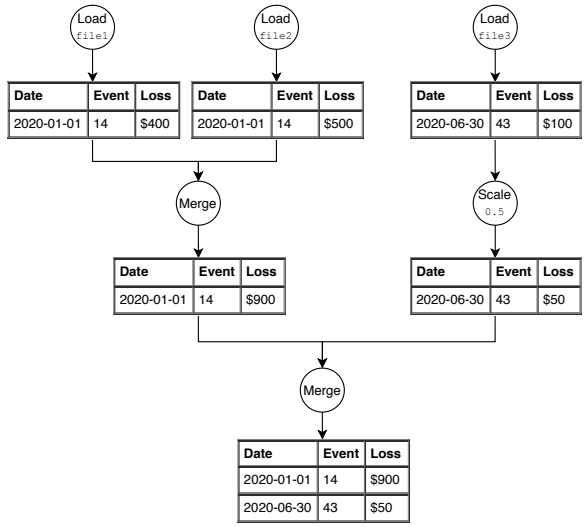


Figure 2: Processing YELT occurrences through a simple example graph for one trial

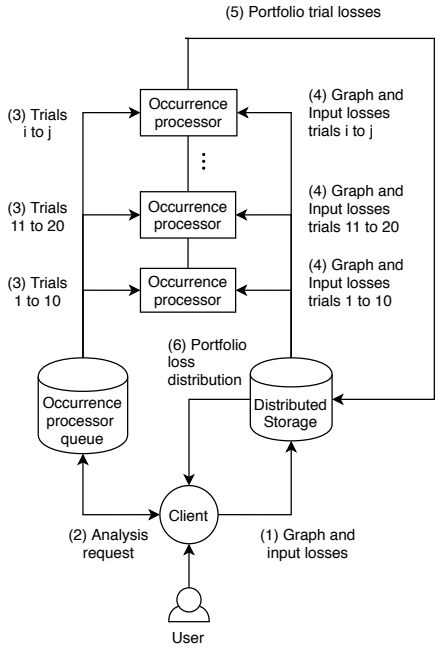


Figure 3: Occurrence processing architecture overview. Numbers reflect the order in which the steps are taken.

YELTs, these YELTs need to be merged by date before applying the vertex's transformation. Therefore, we refer to such a vertex as a *merge vertex*. If a merge vertex finds multiple occurrences for the same event and with the same date in its input YELTs, it combines them into a single occurrence whose loss value is the sum of the loss values of the combined occurrences.

Figure 2 illustrates a simple example portfolio modelled using our graph framework. The source vertices labelled "Load" read location YELTs from storage and send the occurrences across their output edges. "Merge" vertices merge their input YELTs without applying any transformations. In this example, the middle merge vertex combines the two occurrences for the same date and event (`Date=2020-01-01, Event=14`) and sums their losses. The "Scale" vertex scales the loss of each occurrence in its input YELT by 50%. This amounts to covering only 50% of the claimed losses under this treaty. The final vertex in the graph merges everything together, and outputs the portfolio's final YELT.

While using a directed graph for modelling a reinsurance portfolio is natural, it also is novel. As discussed in Section 3, previous reinsurance risk analytics systems are constrained to rigid or tiered portfolio structures. Many complex portfolios cannot be modelled directly using those systems. Directed graphs allow the construction of arbitrarily complex structures from elementary vertex types. A typical financial contract can be modelled using 5–10 vertices.

5 CLOUD-BASED SYSTEM FOR LOCATION-LEVEL RISK ANALYTICS

A typical reinsurance portfolio covers approximately 50–100 million locations; each location is represented by one location YELT per trial. These locations are covered by approximately 50 million insurance contracts, which are reinsured by thousands of reinsurance treaties. This is illustrated in Figure 1. Representing such a portfolio as a graph as in Section 4 results in a graph with hundreds of millions of vertices. A location-level analysis of such a portfolio based on Monte Carlo simulation involves processing several terabytes of input and intermediate data and requires significant computational resources.

We implemented a cloud-based solution that distributes the computation across a large number of compute nodes. This provides scalability, elasticity, and fault tolerance. Figure 3 gives an overview of the system's core architecture. The client (a front end through which the user interacts with the system) uploads the portfolio graph and input YELTs to a *distributed storage system*. To initiate an analysis, the client divides the trials into groups and submits each group to the *occurrence processor queue*. This queue is responsible for assigning each trial group to the next available *occurrence processor*. Each occurrence processor runs on its own compute node, independently of other occurrence processors. Occurrence processors read their input from and write their results to distributed storage, from where they can be retrieved by the client.

While this system design allows us to perform a portfolio analysis by evaluating individual trials independently on different occurrence processors, with *zero* communication between them, processing a single trial through a portfolio graph of hundreds of millions of vertices and edges can require more memory than is available on a commodity compute node if done naïvely. Occurrence processors process the vertices of the portfolio graph in an order that ensures that only few intermediate YELTs need to be held in memory at any point during the analysis. This allows us to process multiple trials in parallel so that CPU utilization is maximized, while doing all computation entirely in-memory using commodity compute nodes.

Determining the optimal evaluation order of the vertices in the portfolio graph is the responsibility of the *graph optimizer*. The

graph optimizer is run only periodically, whenever the portfolio changes substantially due to the addition of new insured locations or contracts. It is an offline task that is *not* part of the portfolio analysis itself. To re-optimize the graph, the client submits a request to the graph optimizer through a graph optimizer queue. The graph optimizer then reads the graph from distributed storage, optimizes its vertex ordering, and writes the resulting rearranged graph back to distributed storage, for use by future analysis runs.

The remainder of this section discusses our risk analytics system in greater detail. Section 5.1 discusses the design of the occurrence processor. Section 5.2 discusses the implementation of the graph optimizer. Section 5.3 offers some final remarks concerning the scalability, elasticity, and fault tolerance of our system design.

5.1 Occurrence Processor

The occurrence processor is responsible for computing the portfolio YELTs of a group of trials from the location YELTs of these trials. The occurrence processor starts one thread per trial. Typically, the number of trials assigned to an occurrence processor is at least the number of cores on the compute node running the occurrence processor, thereby allowing each core to run a thread.

The occurrence processor starts by loading the portfolio graph into memory in *Compressed Sparse Row* (CSR) format [29, pages 84–85], which has a small per-edge and per-vertex memory footprint.

Since the graph representation is static, it can be shared by all threads of the occurrence processor and concurrent accesses to the graph by different threads do not require locking. Once the graph is loaded, each thread begins processing a single trial, traversing the vertices in the portfolio graph and producing the output YELT of each visited vertex from its input YELTs based on the vertex type. Concurrently, an asynchronous I/O thread downloads the location YELTs of subsequent trials, with the goal of minimizing the amount of time worker threads are stalled waiting for input data.

Since each vertex u in the portfolio graph needs access to its input YELTs to produce its output YELT, the vertices producing these input YELTs need to be visited before u. Thus, the vertices of the portfolio graph need to be visited in topological order.

There are many valid topological orders. The chosen order can have a dramatic impact on the amount of memory the occurrence processor uses to evaluate a single trial. Consider a portfolio graph that is a complete binary tree. The YELT produced by each vertex u must be held in memory from the time we visit u—the time the YELT is produced—until we visit the last vertex that has this YELT as one of its inputs. Once this last out-neighbour of u has been visited, the YELT can be discarded and its memory reclaimed. If vertices are visited by decreasing distance from the sink, then the output YELTs of all source vertices have to be held in memory simultaneously because they are all evaluated before any of their out-neighbours. Since half of the vertices in a complete binary tree are source vertices, this means that half of all YELTs must be in memory simultaneously. For a portfolio with 50–100 million locations, this requires several gigabytes of RAM per trial. In contrast, if vertices are visited in postorder (all vertices in each subtree are visited consecutively), only $\lg n$ YELTs need to be in memory at any time, where n is the number of vertices in the graph. This significantly reduces the space needed to store intermediate YELTs.

5.2 Graph Optimizer

Choosing a space-efficient topological ordering of the portfolio graph is the responsibility of the graph optimizer. If the topological ordering of the portfolio graph arranges the vertices in the order $v_1, \ldots, v_n$, then the YELTs that need to be in memory immediately after processing the ith vertex v_i are the ones corresponding to edges $v_j v_k$ with $j \leq i$ and $k > i$: v_j has been visited and has produced its output YELT, while v_k requires this YELT as part of its input and has not been visited yet. The maximum number of YELTs to be held in memory simultaneously is thus

$$\max_{1 \leq i < n} |\{v_j v_k \in E \mid j \leq i < k\}|.$$

We call this the *cut width* of the topological ordering in analogy to the cut width of an undirected graph [17] and say that an edge $v_j v_k$ with $j \leq i < k$ "crosses the cut between v_i and v_{i+1}." Figure 4 illustrates that different topological orderings of the same graph may have different cut widths. Since the cut width of the topological ordering directly determines the maximum number of YELTs that need to be held in memory at the same time, our strategy to minimize the memory requirements of evaluating a single trial is to find a topological ordering of low cut width.

Finding a vertex ordering of *minimum* cut width is NP-hard even for undirected graphs [17]. Fixed-parameter algorithms [11, 31] and a polynomial-time approximation algorithm [19] for computing the cut width of an undirected graph have been proposed in the literature. However, the running times of these algorithms are far from linear. Thus, even if we were able to extend these algorithms to DAGs, they would not be efficient enough for portfolio graphs with hundreds of millions of vertices. Instead, we use a heuristic approach that exploits the structure of portfolio graphs to compute low-cut-width topological orderings for these graphs. This heuristic is not guaranteed to find a topological ordering of the *minimum* cut width, but it does find topological orderings of sufficiently low cut width to lead to low memory requirements of the occurrence processor, and it finds them quickly (in linear time).

The heuristic used by the graph optimizer proceeds in two phases. The first phase computes an initial low-cut-width ordering of the portfolio graph. The second phase modifies the graph and the ordering to reduce the cut width further while ensuring that the modified graph represents the same portfolio as the original graph.

The structure of a typical location-level portfolio. In a typical reinsurance portfolio, the insurance contracts covering individual locations do not interact with each other while the losses of primary insurers are covered by a network of reinsurance treaties. Thus, the portfolio graph can be viewed as a tree of subgraphs with a

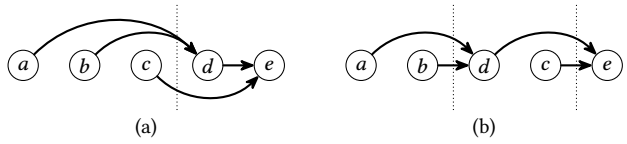


Figure 4: Two topological orderings of the same graph. As indicated by the dashed lines, the ordering in (a) has cut width 3, while the ordering in (b) has cut width 2.

densely connected subgraph representing the network of reinsurance treaties at the root and all other subgraphs representing primary insurance contracts. The subgraph representing the network of reinsurance treaties typically consists of a few thousand vertices. A primary insurance contract is modelled using 5–10 vertices.

Even if these subgraphs are densely connected, the portfolio graph remains very tree-like; it has a large block close to the sink and is otherwise composed of small blocks containing at most a few dozen vertices. A *block* or *2-edge-connected component* is a maximal subgraph that cannot be disconnected by removing a single edge.

While we hope that the flexible graph-based representation of reinsurance portfolios introduced in this paper will allow users to model more complex and fine-tuned portfolio structures than are in use today, we believe that the structure of reinsurance portfolios will remain largely hierarchical, so portfolios should continue to be composed of many fairly small blocks and *very* few larger blocks close to the sink. This is the portfolio structure we exploit.

The initial topological ordering. Recall the example of a low-cut-width ordering of a complete binary tree in Section 5.1. Given the tree-like structure of reinsurance portfolios, this example suggests the following simple strategy for computing a low-cut-width ordering of a portfolio graph: reverse the directions of all edges and perform a depth-first traversal (DFS) of the graph starting at the sink; arrange the vertices in postorder of the resulting DFS tree, that is, in the order the DFS backtracks from them.

The cost of computing a topological ordering in this fashion is linear in the size of the graph [9, Section 22.4]. If the tree of blocks is fairly balanced and most blocks are small, both of which tend to be true for reinsurance portfolios, then the edges crossing any cut in the ordering are the in-edges of roughly a logarithmic number of vertices. If these vertices have low in-degree, the topological ordering thus has low cut width. Some vertices, however, can have very high in-degree. The second phase of the graph optimizer modifies the graph to eliminate high-in-degree and high-out-degree vertices.

Degree reduction. Consider a vertex v of high out-degree t. Such a vertex must create a copy of its output YELT for each of its t out-neighbours. Instead of immediately creating t copies after visiting v, copies can be made in a tree-like manner, making few copies initially and replicating each copy further as needed. In other words, we introduce an "out-tree" of replicator vertices that simply make d copies of their input YELTs, for an appropriate parameter d used to tune the degree reduction (see Section 6). High-in-degree vertices can be reduced in a similar manner. A single high-degree merge vertex can be replaced with an "in-tree" of d-way merge vertices, for the same parameter d. Degree reduction increases the size of the graph by adding vertices in order to reduce the cut width of the graph. It thus trades a slight increase in the amount of computation to be performed for the ability to perform it entirely in memory and hence efficiently. For degree reduction to reduce the cut width of the ordering, however, the construction of the in- and out-trees needs to be informed by the current topological ordering.

To understand the construction of an in-tree (the construction of an out-tree is analogous), let v_i be a vertex with t in-neighbours $v_{j_1}, \ldots, v_{j_t}$, $j_1 < \cdots < j_t < i$ (see Figure 5). The in-edges of v_i cross the cut between v_h and v_{h+1} for all $j_t \le h < i$. The edges between $v_{j_1}, \ldots, v_{j_d}$ and v_i cross the cut between v_h and v_{h+1}

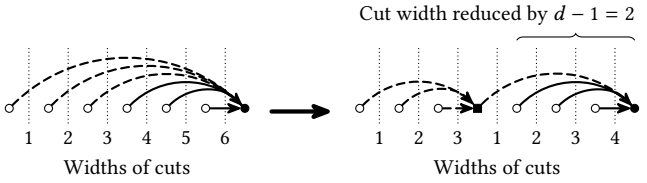


Figure 5: Cut width reduction by inserting a degree-d vertex (square) for $d = 3$. Modified edges are dashed.

for all $j_d \le h < i$. Now assume we introduce a merge vertex v' that merges the YELTs of $v_{j_1}, \ldots, v_{j_d}$ and then provides the merged YELT as one of the inputs of v_i. We remove $v_{j_1}, \ldots, v_{j_d}$ from the set of in-neighbours of v_i, make them in-neighbours of v', and make v' an in-neighbour of v_i. To obtain a valid topological ordering of the resulting graph, we can insert v' into the current topological ordering anywhere between v_{j_d} and v_i. The number of edges crossing any cut before v' or after v_i is not changed by this modification of the graph. The number of edges crossing any cut between v' and v_i is reduced by $d - 1$: before the addition of v', the d edges between $v_{j_1}, \ldots, v_{j_d}$ and v_i cross the cut; after the addition of v', a single edge between v' and v_i crosses the cut. This immediately suggests the following strategy to ensure that v_i's in-edges do not contribute more than d to the number of edges crossing any cut:

We divide the sequence of in-neighbours of v_i into $r = \lceil (t - 1)/(d - 1) \rceil$ groups $V_1, \ldots, V_r$ such that $V_2, \ldots, V_r$ contain $d - 1$ vertices each and V_1 contains up to d vertices. We add a new merge vertex v'_h immediately after the last vertex in each group V_h with $1 \le h < r$. We do not add a new vertex after the last vertex in V_r but refer to v_i as v'_r. We link the vertices $v'_1, \ldots, v'_r$ to form a path, by adding an edge from v'_h to v'_{h+1} for all $1 \le h < r$. We attach the in-neighbours of v_i to this path by adding an edge from every vertex in V_h to v'_h, for all $1 \le h \le r$. This results in v_i's in-tree having the shape of a caterpillar (see Figure 6).

While this shape ensures that after degree reduction, there is no vertex whose incident edges contribute more than d to the number of edges crossing any cut—and it is impossible to do better with degree-d vertices—we construct each in- and out-tree as a complete d-ary tree instead: We divide v_i's in-neighbours into $r = \lceil t/d \rceil$ groups $V_1, \ldots, V_r$ of size at most d, add a new merge vertex v'_h immediately after the last vertex in each group V_h, and add edges from the vertices in V_h to v'_h. The vertices $v'_1, \ldots, v'_r$ become the new in-neighbours of v_i. If $r > d$, we repeat this construction with this new set of in-neighbours until v_i has at most d in-neighbours (see Figure 7). This construction provides the weaker guarantee that the edges in v_i's in-tree contribute at most $d \log_d t$ to the number of edges crossing any cut, but it has the following advantage:

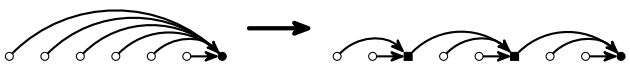


Figure 6: Transformation of a high-degree vertex into a caterpillar of degree $d = 3$. The inserted vertices are squares.

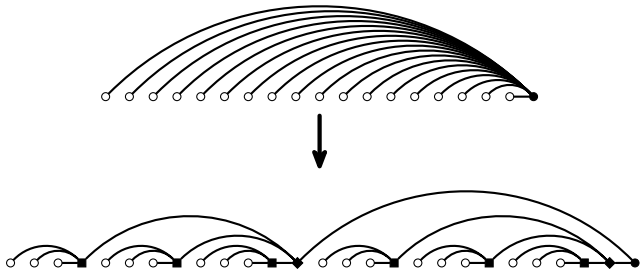


Figure 7: Transformation of a high-degree vertex into a balanced tree of degree $d = 3$. The parents of the leaves are squares. Their parents are diamonds. Edges are directed from left to right. For clarity, arrow tips are not drawn.

We use the cut width of the topological ordering as an approximate measure of the amount of memory needed by the occurrence processor to process one trial. This approximation is accurate if all YELTs have roughly the same size—in this case, it is the *number* of YELTs that need to be held in memory at any time, the cut width of the topological ordering, that determines the space requirements of the occurrence processor. This assumption is mostly satisfied if many occurrences are combined when a merge vertex merges YELTs, but this may not always be the case. In the extreme case when no occurrences are combined, the d input YELTs of v_i after degree reduction have the same total size as the t input YELTs of v_i before degree reduction. Degree reduction is completely ineffective as a means to reduce the space requirements of the occurrence processor in this case. However, the caterpillar structure would mean that the occurrences in half of the original input YELTs of v_i pass through at least half of the merge vertices in v_i's in-tree, which would negatively affect the computation cost. A balanced d-ary in-tree implements the merging process efficiently as a tree of d-ary merges while achieving only a slightly lower reduction in cut width. It is thus the reasonable defensive choice.

5.3 Scalability, Elasticity, and Fault Tolerance

The implementation of occurrence processors as stateless processes without communication between them supports scalability, elasticity, and fault tolerance. To improve system throughput (e.g., to scale to a larger portfolioor more trials) and to respond to changing system loads, it is easy to provision and deprovision occurrence processors. Unresponsive occurrence processors can be restarted and have their work rescheduled to a different compute node.

In *theory*, the scalability of our system is limited only by the number of trials to be evaluated. An important *practical* limiter is I/O. Provisioning more occurrence processors and allocating fewer trials to each occurrence processor increases the overall I/O bandwidth of the system and thus helps performance. However, decreasing the number of trials per occurrence processor beyond some point produces only minimal performance gains. There are two reasons for this. First, the number of trials per occurrence processor should be greater than the number of cores on a compute node so the I/O cost of loading most input YELTs can be hidden by loading the YELTs of later trials while evaluating ones that have already been loaded. Since the cost of loading the YELTs of the first batch of trials, one per core, cannot be hidden, a higher trials-to-cores ratio per occurrence processor—that is, a smaller number of occurrence processors—hides a greater fraction of the I/O cost. Second, the portfolio graph needs to be loaded in its entirety by each occurrence processor before the occurrence processor can start evaluating its first trial—the cost of loading the portfolio graph cannot be hidden. This cost is substantial because the portfolio graphs can be large (> 10GB). Since this graph needs to be loaded only once no matter how many trials each occurrence processor evaluates, this I/O cost can be amortized by allocating sufficiently many trials to each occurrence processor, but this limits scalability.

6 EVALUATION

We conducted a number of experiments to evaluate the effectiveness of our approach to location-level portfolio analysis. Since the feasibility of our approach depends on the ability of occurrence processors to evaluate multiple trials in memory simultaneously, Section 6.2 investigates the cut width of the topological ordering of a typical portfolio graph produced using our method, including the impact of degree reduction on the cut width and the amount of memory and time taken by the occurrence processor to evaluate a single trial. Section 6.3 performs a full 10,000-trial portfolio analysis using our system and demonstrates that it can carry out such an analysis efficiently. Section 6.4 compares the performance of our system against a commercial system on the market today.

6.1 Test Portfolio

Since current commercial systems are unable to perform a full portfolio analysis at the resolution of individual locations in a reasonable amount of time, there do not exist any real-world location-level portfolio data to date that could be used in experiments to evaluate our system. Therefore, we are limited to using synthetic data.

We constructed our test data set from the portfolios of primary insurers and from the portfolio of a reinsurer composed of treaties with primary insurers and insurance contracts for high-value individual properties.[1] This portfolio structure is illustrated in Figure 8.

High-value individual properties include bridges or office towers worth hundreds of millions of dollars. Such properties are insured directly by reinsurance companies. The contract insuring each such property is modelled using a subgraph of approximately 50 vertices, with fairly high connectivity near the sourceand sink of the subgraph. There are 59 such structures in our test portfolio.

Each primary insurer business unit covers 100,000 insured locations and is composed of approximately 400,000 vertices. In this structure, the contract for each location is modelled using a subgraph of 4–5 vertices. The losses from contracts are combined into the insurer's loss YELT using a high-degree merge vertex. Our graph models 700 different primary insurance portfolios, making these structures the bulk of our graph.

The "reinsurer's contractual terms" structure serves as the sink of the graph and takes the losses from the primary insurer business units and high-value properties as inputs. This structure contains approximately 5,000 vertices and models interdependent reinsurance contractual terms for the entire business of a real reinsurance group. The structure includes several large merge vertices, one with

[1] Due to confidentiality requirements, the specific companies cannot be identified.

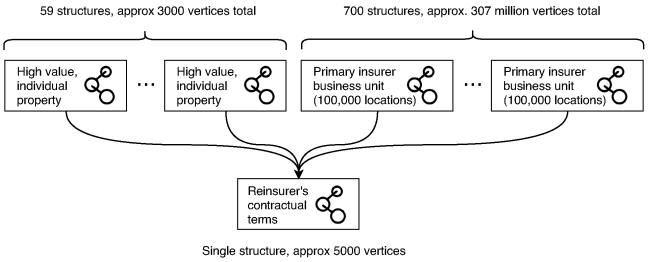


Figure 8: High-level structure of the location-level graph used in our experiments

in-degree over 1,000 and several with in-degree over 100, making it the most complex component of the graph in terms of connectivity.

Overall, our test graph had approximately 307M vertices and 377M edges. Its structure reflects the flow of risk from individual insured locations via primary insurance contracts to reinsurance treaties and thus should be representative of location-level reinsurance portfolios that we expect to emerge in the real world once systems such as ours make location-level portfolio analysis feasible.

6.2 Graph Layout, Single-Trial Memory Usage and Processing Time

Our first experiments concern the impact of the graph layout on memory usage and processing time. Since trials are evaluated independently in our system, we evaluated the impact of graph layout and degree reduction in single-trial runs on a Ubuntu 18.04 Linux workstation with an Intel i7-6700K CPU @ 4.0GHz, 64GB of DDR4 RAM @ 2400 MT/s, and with an SSD as the storage system.

Cut width vs graph size. Figure 9 shows how the choice of the maximum degree d during degree reduction affected the cut width and size of the optimized graph. Values of $d \geq 2^{14}$ resulted in no degree reduction and no modifications of the graph because the input portfolio graph had no vertices of degree greater than 2^{14}. The cut width of the topological ordering was around 10,000 in this case. A choice of $d = 2$ reduced the cut width to 157 but also added 75M vertices to the graph due the replacement of high-degree mergers with large binary trees. The sweet spot for our test graph was achieved for $d = 16$, which resulted in a cut width of 310 and increased the size of the graph by only around 2%.

Running time of the graph optimizer. Since the graph optimizer is run only periodically, its running time is a secondary concern as long as it is not excessive. For all choices of d in our experiments, the graph optimizer ran in under 4 minutes. Around 50s were spent on reading the input graph and writing the optimized graph back to disk. It took around 10s to compute the initial topological ordering of the input graph. Degree reduction was the costliest step, taking around 150s. For $d \geq 2^{14}$, the degree reduction cost dropped to almost zero since the graph is not modified by the degree reduction in this case. As a result, the running time of the graph optimizer dropped to around 60s (I/O time + topological sorting) for $d \geq 2^{14}$.

Running time of occurrence processor. The running time of the occurrence processor depends on the total size of all YELTs it needs

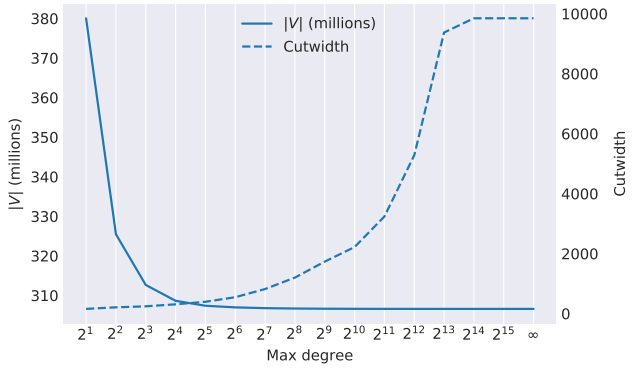


Figure 9: Number of vertices vs cut width for different values of the maximum degree parameter

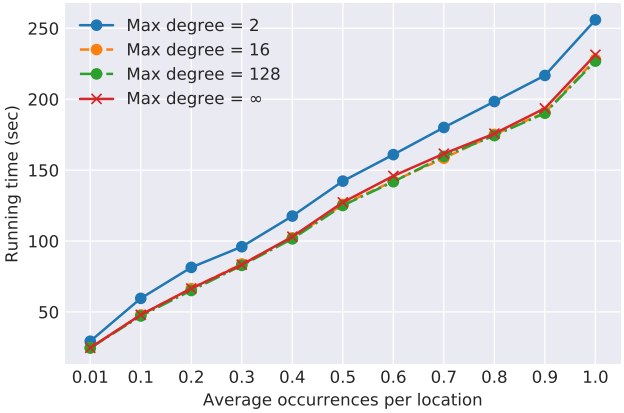


Figure 10: Running time of the occurrence processor as a function of maximum degree and average number of occurrences per location

to process, which is correlated with the size of the portfolio graph and the average size of the input YELTs. As discussed at the beginning of this section, the maximum vertex degree used during degree reduction impacts the graph size. We controlled the size of the input YELTs using the average number of occurrences per location as a parameter. We varied this parameter between 0.01 and 1. Since individual properties are unlikely to make an insurance claim every year, the average number of occurrences generated per location is typically no greater than 0.2 in practice (which resulted in 4TB of input YELTs for a 10,000 trial analysis for the test portfolio).

Figure 10 shows the time to process a single trial using the occurrence processor as a function of the average number of occurrences per location and for different levels of degree reduction. There is a baseline cost of about 25 seconds to load the graph into memory. In a multi-trial portfolio analysis, this cost is amortized over multiple trials evaluated by the same occurrence processor. Above this baseline, the running time scaled linearly with the average number of occurrences per location, which was to be expected because this number should have a roughly linear influence on the total size of

all YELTs in the portfolio graph. Degree reduction with $d \geq 16$ had an insignificant impact on the running time, as the size of the graph increases by no more than 2% for these parameters compared to no degree reduction. The running time for $d = 2$ was noticeably higher than for $d \geq 16$ due to the around 75M vertices added by the degree reduction.

Memory usage of occurrence processor. The occurrence processor used 112MB of memory to store the intermediate YELTs for the original graph without degree reduction ($d = \infty$, cut width 10,000), 31MB for the graph with maximum degree $d = 128$ (cut width 820), and 15MB for the graph with maximum degree $d = 16$ (cut width 310). These results are significant for two reasons:

First, the reduction in cut width for smaller values of d translates into a reduction in memory size, albeit not in a linear one: a factor of around 30 between the cut widths for $d = \infty$ and for $d = 16$ translate into only a factor of 8 between the amounts of memory used. This is because in the unreduced graph, the cuts crossed by many edges are close to the sources of the graph, but edges close to the sources carry only few occurrences; some of them do not carry any occurrences. As the cut width decreases with smaller values of d, the cuts causing peak memory usage shift closer to the sink where YELTs carry more occurrences.

Second, processing even the unreduced graph uses only a modest amount of memory, which shows that the initial topological ordering is good enough to allow processing multiple trials in memory. This is significant because, as mentioned in Section 5.2, if few occurrences are combined during merge steps, degree reduction provides little benefit to memory usage.

6.3 Evaluation as a Distributed System

To evaluate the feasibility of a full-scale location-level portfolio analysis consisting of 10,000 trials, we provisioned 20 m5.12xlarge compute nodes from Amazon EC2 to serve as occurrence processors and submitted a 10,000 trial job using our graph reduced to a maximum degree of 16. We used m5.12xlarge nodes for their high network bandwidth and because the high vCPU count (48) allowed us to reduce the number of times the graph had to be loaded into memory. We used an average of 14 million input occurrences per trial, an aggressively high estimate of what we would expect from a typical location-level analysis. Each compute node was issued 500 trials to process, an average 10–11 trials per vCPU. We used Amazon's Simple Storage System (S3) for distributed storage, as it scales well and has high throughput for Amazon EC2 nodes located in the same availability zone. We used Amazon's Simple Queue Service (SQS) for the occurrence processor queue.

Starting from a newly provisioned cluster of occurrence processors with no data preloaded onto it, the system was able to compute the portfolio YELTs for 10,000 trials in approximately 33 minutes. The dedicated I/O threads of the occurrence processors retrieved all data (approximately 4TB) in approximately 28 minutes. This cost was nearly perfectly hidden by downloading the input data for later trials while earlier trials were being processed. Since the rate of download exceeded the rate of occurrence processing, our current implementation is bound by computation speed. However, any further optimizations of the processing speed, without any I/O optimizations, will not reduce the running time below 28 minutes.

6.4 Comparison Against a Commercial System

The substantial licensing fees of commercial risk analytics systems make it infeasible to compare our system against a wide range of them. Due to a working relationship with one of the major vendors,[2] we were given access to a server running their platform.

The vendor's analytics suite offers two separate programs: an insurance client for modelling primary insurance structures and a reinsurance client for modelling reinsurance structures. Using these programs to model a reinsurer's portfolio at location-level requires using the insurance client to model the primary insurance contracts in the reinsurer's portfolio, manually exporting the resulting loss distributions to the reinsurance client, and running the reinsurance client to apply the portfolio's reinsurance treaties to the loss distributions generated by the insurance client.

We used this process to perform a location-level analysis on a real primary insurer's data set of 500,000 locations, representing hurricane risk exposures in a US state. Each location was covered by one primary insurance contract. As the reinsurance structure, we created a simple synthetic contract. A full reinsurance portfolio includes locations from many other states and countries. Thus, this data set represents only a small slice ($< 1\%$) of the amount of work required for a typical location-level analysis.

We evaluated the vendor's analytics suite on the vendor's hardware, a virtualized Windows Server 2016 machine running on a Xeon Gold 6154 processor with 16 virtualized cores and 64GB of memory, and another Windows Server machine running Microsoft SQL Server 2017 with 2 virtualized cores and 16GB of memory. With this configuration, it took the vendor's platform approximately 38 minutes to compute the portfolio's losses.

We ran the same experiment on our platform using comparable compute resources: one m5.4xlarge EC2 instance with 16 cores and 64GB of memory. We could not run on the vendor's hardware because the implementation of our platform is Linux-based. With this configuration, our platform took 35 seconds to perform the same analysis (plus an additional 11 seconds to topologically sort the graph and reduce its maximum degree). Due to nuances in the interpretations of some financial contracts, both systems generated different loss distributions in some instances. However, with detailed knowledge of the vendor's interpretation of such contracts, our system is capable of generating matching output.

In addition to being significantly faster, our system is also significantly more flexible. The vendor's system allows only one primary insurance contract per location. The contract itself only supports the three most common terms. The reinsurance client allows users to create portfolios containing multiple contracts of different types, but they are difficult to combine to model arbitrary dependencies between contracts. The system uses a referencing system to direct output from one reinsurance contract to another but only some contracts can be referenced by others and keeping track of the overall structure becomes difficult as more references are added.

On the small data set in this comparison, our system was over 50 times faster than the vendor's system. Therefore, while our system can perform a full-scale location-level analysis in around 30 minutes, we expect the vendor's system to take more than a day. This has a significant impact on the feasibility of location-level analyses in

[2] Again, confidentiality agreements prevent us from disclosing the name of the vendor.

the reinsurance industry. Moreover, we expect that the vendor's system's use of a single SQL server for processing and retrieving data introduces a significant bottleneck that severely hampers its scalability to the size of a full-scale location-level portfolio.

7 CONCLUSION

We presented a system for processing complex reinsurance portfolios at location-level resolution. By employing a flexible graph representation, our system can model arbitrary dependencies between reinsurance contracts. In contrast, many commercial systems on the market impose significant restrictions on the type of portfolio structures they can model. In spite of this greater flexibility, our system is over 50 times faster than at least one commercial system by a major vendor we were able to use for comparison. Moreover, it is unclear whether current commercial systems can scale to the size of a full-size location-level portfolio, an input our system can process in 33 minutes using a scalable cloud-based architecture.

To support interactive use of our system, future work should focus on reducing the time of a portfolio analysis further by supporting incremental updates to the graph and caching intermediate results. Repeated runs of portfolio analyses on the same portfolio are necessary mostly when investigating the impact of adding new contracts or renegotiating the structure of existing contracts on the portfolio's risk exposure. Caching a well-chosen subset of intermediate results should enable the computation of an updated portfolio YELT after each change in a matter of seconds because most changes are local to only a small part of the graph and only YELTs "downstream" from these changes need to be recomputed.

ACKNOWLEDGMENTS

The research of Neil Burke and Norbert Zeh was supported by the Natural Sciences and Engineering Research Council of Canada.

REFERENCES

[1] AIR Worldwide. 2019. Touchstone. https://www.air-worldwide.com/Software-Solutions/Touchstone/. Accessed: 2019-10-16.
[2] AIR Worldwide. 2019. Touchstone Re. https://www.air-worldwide.com/Software-Solutions/Touchstone-Re/. Accessed: 2019-10-16.
[3] Tyler Akidau, Alex Balikov, Kaya Bekiroğlu, Slava Chernyak, Josh Haberman, Reuven Lax, Sam McVeety, Daniel Mills, Paul Nordstrom, and Sam Whittle. 2013. MillWheel: fault-tolerant stream processing at internet scale. *Proceedings of the VLDB Endowment* 6, 11 (2013), 1033–1044.
[4] Analyze Re. 2019. Write More Profitable Reinsurance. https://analyzere.com/. Accessed: 2019-10-16.
[5] Aon. 2019. ReMetrica. https://www.aon.com/reinsurance/analytics-(1)/remetrica.jsp. Accessed: 2019-10-16.
[6] Aman Kumar Bahl, Oliver Baltzer, Andrew Rau-Chaplin, and Blesson Varghese. 2012. Parallel simulations for analysing portfolios of catastrophic event risk. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. IEEE, 1176–1184.
[7] Neil Burke, Andrew Rau-Chaplin, and Blesson Varghese. 2016. Computing probable maximum loss in catastrophe reinsurance portfolios on multi-core and many-core architectures. *Concurrency and Computation: Practice and Experience* 28, 3 (2016), 836–847.
[8] Karen M. Clark. 2002. The Use of Computer Modeling in Estimating and Managing Future Catastrophe Losses. *The Geneva Papers on Risk and Insurance. Issues and Practice* 27, 2 (2002), 181–195. http://www.jstor.org/stable/41952626
[9] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, Third Edition* (3rd ed.). The MIT Press.
[10] Frank Dehne, Glenn Hickey, Andrew Rau-Chaplin, and Mark Byrne. 2009. Parallel catastrophe modelling on a cell processor. In *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research*. IBM Corp., 24–31.
[11] Archontia C Giannopoulou, Michał Pilipczuk, Jean-Florent Raymond, Dimitrios M Thilikos, and Marcin Wrochna. 2019. Cutwidth: Obstructions and algorithmic aspects. *Algorithmica* 81, 2 (2019), 557–588.
[12] Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. 2012. PowerGraph: Distributed graph-parallel computation on natural graphs. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. 17–30.
[13] Joseph E Gonzalez, Reynold S Xin, Ankur Dave, Daniel Crankshaw, Michael J Franklin, and Ion Stoica. 2014. GraphX: Graph Processing in a Distributed Dataflow Framework. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, Vol. 14. 599–613.
[14] Guy Carpenter & Company. 2019. MetaRisk. http://www.guycarp.com/managing-risk/analytics/metarisk.html. Accessed: 2019-10-16.
[15] Wook-Shin Han, Sangyeon Lee, Kyungyeol Park, Jeong-Hoon Lee, Min-Soo Kim, Jinha Kim, and Hwanjo Yu. 2013. TurboGraph: A fast parallel graph engine handling billion-scale graphs in a single PC. In *Proc. of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 77–85.
[16] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. 2007. Dryad: Distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS Operating Systems Review*. ACM, 59–72.
[17] Ephraim Korach and Nir Solel. 1993. Tree-width, path-width, and cutwidth. *Discrete Applied Mathematics* 43, 1 (1993), 97–101.
[18] Aapo Kyrola, Guy Blelloch, and Carlos Guestrin. 2012. GraphChi: Large-Scale Graph Computation on Just a PC. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. 31–46.
[19] Tom Leighton and Satish Rao. 1999. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM* 46, 6 (1999), 787–832.
[20] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph Hellerstein. 2010. GraphLab: A New Framework for Parallel Machine Learning. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*. 340–349.
[21] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: A system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*. ACM, 135–146.
[22] Derek G Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martín Abadi. 2013. Naiad: A timely dataflow system. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles*. ACM, 439–455.
[23] American Academy of Actuaries Extreme Events and Property Lines Committee. 2018. Uses of Catastrophe Model Output. https://www.actuary.org/sites/default/files/files/publications/Catastrophe_Modeling_Monograph_07.25.2018.pdf.
[24] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The pagerank citation ranking: Bringing order to the web*. Technical Report. Stanford InfoLab.
[25] Andrew Rau-Chaplin, Blesson Varghese, Duane Wilson, Zhimin Yao, and Norbert Zeh. 2013. QuPARA: Query-driven large-scale portfolio aggregate risk analysis on MapReduce. In *IEEE International Conference on Big Data*. IEEE, 703–709.
[26] Reynolds Porter Chamberlain. 2019. Software | RPC. https://www.rpc.co.uk/services/rpc-consulting/software://www.rpc.co.uk/services/rpc-consulting/software/. Accessed: 2019-10-16.
[27] Risk Management Solutions. 2019. Additional Software Products. https://www.rms.com/software/additional-software-products. Accessed: 2019-10-16.
[28] Amitabha Roy, Ivo Mihailovic, and Willy Zwaenepoel. 2013. X-stream: Edge-centric graph processing using streaming partitions. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles*. ACM, 472–488.
[29] Yousef Saad. 2000. *Iterative methods for sparse linear systems* (second ed.).
[30] Julian Shun and Guy E. Blelloch. 2013. Ligra: A Lightweight Graph Processing Framework for Shared Memory. In *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 135–146.
[31] Dimitrios M Thilikos, Maria Serna, and Hans L Bodlaender. 2005. Cutwidth I: A linear time fixed parameter algorithm. *Journal of Algorithms* 56, 1 (2005), 1–24.
[32] TigerRisk Partners. 2019. The leading risk-to-capital advisor worldwide. https://tigerrisk.com/. Accessed: 2019-10-16.
[33] Ultimate Risk Solutions. 2019. Leading Provider of Dynamic Financial Analysis DFA Software. https://www.ultirisk.com/. Accessed: 2019-10-16.
[34] Sitalakshmi Venkatraman, Kiran Fahd, Samuel Kaspi, and Ramanathan Venkatraman. 2016. SQL versus NoSQL movement with big data analytics. *International Journal of Information Technology and Computer Science* 8 (2016), 59–66.
[35] Da Zheng, Disa Mhembere, Randal Burns, Joshua Vogelstein, Carey E Priebe, and Alexander S Szalay. 2015. FlashGraph: Processing billion-node graphs on an array of commodity SSDs. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*. 45–58.
[36] Xiaowei Zhu, Wentao Han, and Wenguang Chen. 2015. GridGraph: Large-scale graph processing on a single machine using 2-level hierarchical partitioning. In *2015 USENIX Annual Technical Conference (USENIXATC 15)*. 375–386.

Investigation of Encrypted and Obfuscated Network Traffic Utilizing Machine Learning

Kay Boldt
University of New Brunswick
Fredericton, Canada
kay.boldt@unb.ca

Kenneth B. Kent
University of New Brunswick
Fredericton, Canada
ken@unb.ca

Rainer Herpers
University of Applied Sciences
Bonn-Rhein-Sieg
Sankt Augustin, Germany
rainer.herpers@h-brs.de

ABSTRACT

This paper utilizes machine learning to investigate the classification of encryption applied to network traffic and the underlying activities. It is firstly motivated by the difficulty of traditional traffic classification caused by additional encryption as ports and headers are hidden. Secondly, the results also present the effectiveness of currently available privacy-enhancing technologies. A new dataset is created, containing Pure (without additional encryption), Tor, Tor with obfuscation, VPN and VPN+Tor network traffic. Additionally, there are five different activities performed during each kind of traffic recording, namely audio streaming, browsing, P2P/SFTP file transfers and video conferencing. The traffic is classified by extracting features based on flows calculated by ARGUS and CICFlowMeter, combining three classifiers with seven feature selection algorithms. The results for the classification of the encryption clearly indicate the possibility of using this detection system in a modified fashion within a practical application. For the detection of activities inside encrypted network traffic, the results show that the disguise is ineffective. Overall, this reveals the need to improve the resistance of commonly used techniques for the protection of network traffic against machine learning.

CCS CONCEPTS

• **Security and privacy**; • **Computing methodologies** → *Machine learning*;

KEYWORDS

VPN, Tor, machine learning

ACM Reference Format:
Kay Boldt, Kenneth B. Kent, and Rainer Herpers. 2020. Investigation of Encrypted and Obfuscated Network Traffic Utilizing Machine Learning. In *Proceedings of 30th Annual International Conference on Computer Science and Software Engineering (CASCON'20).* IBM Corp., Riverton, NJ, USA, 10 pages.

1 INTRODUCTION

Nowadays, the use of virtual private networks (VPN) or Tor can prevent or complicate the classification of network traffic. A VPN creates a virtual tunnel between the client and an endpoint by utilizing encryption of the network traffic, while Tor creates a layered tunnel through a network of nodes to the desired destination (see Section 2.2). Traditional approaches for network classification fail because they rely mostly on network ports or headers of network packets and both are hidden under encryption. Therefore, machine learning might be a possible solution. Further, this also gives an indication of how well the currently available privacy-enhancing technologies work.

CASCON'20, November 10–13, 2020, Toronto, Canada

1.1 Objectives

In this research, different machine learning methods were investigated and examined to evaluate how well the detection and classification of the additionally encrypted network traffic works, as well as the hidden activity performed within the network traffic. Detection means that the use of an encryption method is discovered, while classification means that the specific encryption is recognized (e.g., VPN). To achieve this, a new dataset was created, containing several types of additional encryption, including VPN, Tor, Tor with obfuscation and VPN+Tor (VPN through Tor). The activities performed were audio streaming, browsing, P2P/SFTP file transfer and video conferencing. A new dataset was created to have full control of the comprising encryptions and applications, which in turn contains less noise than naturally occurring network data.

To classify the network traffic with machine learning, statistical information/features about the recorded network flows were calculated, utilizing ARGUS [2] and CICFlowMeter [3]. For machine learning, selected methods from the machine learning framework Scikit-learn [21] were tested using preliminary network data. Based on those results the most suitable candidates for the analysis of network traffic, with and without additional encryption, were further utilized. To classify the network traffic, several steps were necessary. The first step was the calculation of the statistical features based on flows. Secondly, several feature selection algorithms were utilized to create different sets of features. Finally, several classifiers were used with each of those sets to classify the used encryption and, based on the result, the activity performed within the network traffic.

In addition to the comparison of the two tools for feature calculation and the different machine learning algorithms, another goal was determining whether the detection of the encryption and the activities within can be done in practice, as this would pose a significant threat towards privacy. Overall, this assesses the effectiveness of the investigated privacy-enhancing techniques and can be used to discern which techniques may need improvement.

1.2 Paper Organization

This paper is organized as follows. Section 2 covers the background knowledge and reviews the related work. Section 3 describes the planning of the dataset created within this research as well as the extraction of features from network traffic and preparations to utilize it for machine learning. Further, it includes planning for feature selection and classification methods as well as the evaluation. Section 4 is about the implementation and contains the setup for the dataset, the tools for feature creation and the algorithms for feature selection and classification. Section 5 comprises an analysis of the feature selection and classification results. Section 6 presents the conclusion and Section 7 is about future work.

2 BACKGROUND

Network traffic is usually encrypted or obfuscated using a commercial VPN and/or Tor, which are explained in the following sections. In addition, machine learning, which was used to analyze network traffic in this research, is also introduced.

2.1 VPN

When using a VPN (virtual private network tunnel) [17], the network traffic of a client is encrypted locally and forwarded to a VPN server. This server decrypts the data and forwards it to the original destination such as internal systems of a company or a website. The services used or performed activities are protected by the encryption of the VPN.

2.2 Tor

Tor [12] is an anonymization network consisting of several thousand nodes or relays operated by volunteers around the world using onion routing [14]. The applied layered encryption is the reason for the name onion routing. A user who wants to use this network will be connected to three randomly selected nodes [12]. With each of these three nodes, the user negotiates its own encryption/decryption key. The data to be transmitted is then encrypted three times on the user's system with the individual keys of the nodes and then transmitted to the first node. This removes the outermost encryption and forwards the traffic to the second node, which decrypts the second level. At the last node, the final encryption level is removed and the network traffic is forwarded to its actual destination, which only sees the IP address of the last node and therefore not the one of the actual user.

Additionally, Tor also supports so-called "pluggable transports" [8], which is a wrapper around the regular Tor traffic and is designed to circumvent censorship and obfuscate network traffic. Currently (version 9.0.1), the Tor browser supports meek-azure and obfs4.

Unfortunately, the bandwidth provided by Tor+meek is far too low to successfully load modern web pages, connect to Spotify for audio streaming, or start a video conference using Hangout.

Obfs4, the second currently supported obfuscation protocol, [6] is the successor of obfs3 [5] and ScrambleSuit [7]. It encrypts the Tor traffic in a way that it looks uniformly random. The interesting feature of obfs4 for this research is the capability to disguise flow signatures by offering protection against some protocol fingerprint attacks, especially based on the packet size, and optionally on the packet timing.

Obfs4 achieves this by implementing a protocol polymorphism [6, 7], consisting of two steps. The first one manages the packet length obfuscation. As long as enough data is in the send buffer, all packets are as large as the maximum transmission unit (MTU). If the send buffer no longer contains enough data, a random packet length is chosen, and the last packet will be padded to this length, resulting in an obfuscated packet size.

The second step is the optional obfuscation of the inter-arrival times. As long as there is data to be sent, it will pick a random delay and pause the transmission accordingly. This should protect network traffic from being classified based on packet timings.

2.3 Machine Learning

Machine learning can be used to obtain information automatically from data [22]. This allows structures, patterns and information to be recognized in large amounts of data. Usually, this would be very difficult or even impossible with conventional approaches [15]. This research uses algorithms that belong to the class of supervised learning, which needs labelled data.

2.3.1 Training and Testing. Through testing and validation, the error of the trained model in the generalization can be determined [15]. A part of the existing data is retained during training and used for testing. In order to not waste large amounts of data, but still be able to achieve a trustworthy score for the model, the so-called k-fold stratified cross-validation can be used [22]. Usually, 10 folds are used, which means that the data is split into 10 folds, maintaining the original distribution of samples per class, using nine folds for training and the last one for testing. The same is now repeated until every fold has been the test fold. This gives 10 scores on 10 different testing sets that can now be averaged in order to get a trustworthy score of the performance of the model.

2.3.2 Evaluation. To evaluate the performance of the different algorithms and feature subsets for each classification task the true positive rate (TPR—also called recall), false positive rate (FPR), precision and F1 score were calculated per class and on average. The calculations are as follows:

- TPR or Recall [15]: $\frac{TP}{TP + FN}$
- FPR [22]: $\frac{FP}{FP + TN}$
- Precision [15]: $\frac{TP}{TP + FP}$
- F1 [15]: $2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$

TP: true positives—the sample is positive as well as the classification of it.
FN: false negatives—the sample is positive but the classification is negative.
FP: false positives—the sample is negative but the classification is positive.
TN: true negatives—the sample is negative as well as the classification of it.

The TPR and FPR can be used to see how good or bad the predictions of a certain classifier are (e.g., the FPR indicates how many

false alarms a classifier generates). The precision combines the TP and FP to describe how precise the prediction of a certain classifier is. F1 is a combination of precision and recall and will be used as a value to rank the performance of all classifiers.

2.3.3 Decision Tree. A decision tree consists of nodes with learned conditions based on the training data. New samples are checked against the condition of the root node and will travel all the way down to a leaf node where it is finally classified. A tree has the advantage that it can be visualized, which helps to get an insight into the classification progress and the importance of certain features [22].

2.3.4 Random Forest. A random forest is a combination of multiple decision trees. To build those trees, the training data is split into random subsets, where each subset is used to create a single tree. One difference with a decision tree is that not all features are used in a specific tree. Instead, a random subset is used. The classification starts the same way as with a normal decision tree. However, in the end, a majority vote of all trees of the forest will determine the class of a new sample [22].

2.3.5 K-Nearest Neighbours. K-nearest neighbours (KNN) is a simple algorithm, which is based on similarity [18]. It stores the training data and uses this to classify new elements by choosing the closest k elements, which are used for a majority vote to determine the class [22]. To measure how close/far two objects are, the Euclidean distance can be used [18].

2.4 Related Work

A few groups have previously conducted research in VPN or Tor traffic detection [11, 13, 16, 23] by using different machine learning techniques.

Draper-Gil et al. [13] focused on time-related features in order to distinguish between VPN and non-VPN traffic. For this, they created flows of regular encrypted network traffic from different classes, like browsing, email, streaming, file transfer, VoIP and P2P. Moreover, they did the same for VPN traffic. The flows were created by the ISCXFlowMeter using different flow timeout values (15, 30, 60 and 120 seconds). The machine learning part was performed with Weka using C4.5 and KNN. Overall, the C4.5 algorithm achieved better results and some traffic types like VoIP had a good classification rate while Chat is hard to classify. As the recognition of VPN is a binary decision in this work, the results have to be interpreted carefully.

Lashkari et al. [16] also focused on time-based features, but this time they tried to distinguish between Tor and non-Tor traffic. For the generation of the network traffic for the different classes (browsing, VoIP, P2P, etc.) they captured the traffic without any additional encryption at a virtual machine. This traffic was subsequently routed through a gateway in order to send it through Tor. At the gateway, the traffic was captured a second time, but now with the additional Tor encryption. This capture approach ensures that exactly the same network data is used for the encryption as captured before this step. However, one problem is that, especially in the case of time-based features, the timing of the packets is heavily affected by the slower Tor network. In a real-world scenario a user who does not use Tor will have different timings than in this case. From the captured data they extracted flows using the ISCXFlowMeter with different flow timeout values. The resulting dataset is unbalanced (e.g., 969 Tor flows and 38,285 non-Tor flows). Based on this they created two different scenarios. In the first one they used their data from this paper and merged it with the former paper [13] and labelled it as non-Tor. The resulting classifier should tell if a given sample belongs to Tor or not. In the second case, they only used the data generated for this paper where they tried to classify the used application in Tor traffic. For the classification itself, they used Weka with the algorithms Zero R, C4.5 and KNN for the first scenario and Random Forest, C4.5 and KNN for the second scenario. Overall they achieved a precision of 99% for the differentiation between Tor and non-Tor traffic and 84.1% for the classification of the Tor traffic type. Similar to their previous research [13] they have some classes like chat or email, which have a low classification rate while P2P has a good one. As with the previous work, the recognition of Tor is a binary decision in this case and the results have to be interpreted carefully again.

Cuzzocrea et al. [11] used different machine learning algorithms to detect and classify applications within Tor network traffic. They also captured Tor and non-Tor network traffic in one session like Lashkari et al. [16] did and used the ISCXFlowMeter to generate the flows afterwards. The compared machine learning algorithms are J48, J48Consolidated, BayesNet, jRip, OneR and REPTree from Weka. In the end, J48 and jRip performed best overall, but the results for different traffic classes are strange, because they are steady. All classes have more or less the same results (e.g., email and P2P have a true positive rate of 99.8% and 99.6%) while all other papers have a huge difference between those two.

Shahbar and Zincir-Heywood [23] tried the Tor traffic classification with two different approaches. One uses the Tor circuits and cells, the other focuses on the Tor network flows. The main difference between those two approaches is where the analyzer needs to be. To get the information about the Tor circuits and cells, the analyzer needs access to the Tor relay. For the Tor network flows the data can be captured on the host, in the local network or at the ISP level. They focused on P2P, streaming and browsing and tested different machine learning algorithms with Weka. For circuit level classification, random forest achieved a cross-validated accuracy of 94.9%. For flow level classification, they managed to achieve a cross-validated accuracy of 99.2% using a bayesian network classifier. However, their dataset is unbalanced with 60% browsing, 20% streaming and 20% BitTorrent data.

Montieri et al. [20] went a step further. They tried to distinguish between different anonymity tools (Tor, I2P, JonDonym) and classified the traffic inside of them. To create the flows from the network traffic, the researchers in this paper used the tool Tranalyzer2 [9]. Because the dataset is highly biased (e.g., 6,335 JonDonym samples and 645,708 I2P samples) they downsampled the set to 10% and 5% but nevertheless, their dataset is still very unbalanced. For the classification itself, they used naive bayes, bayesian networks, C4.5 and random forest. They achieved an accuracy of about 99.99% for the classification of the different anonymity tools and 98.03% for the applications.

This work contributes to this research by creating a new dataset consisting of traffic created by multiple activities recorded without additional encryption and with VPN, Tor, Tor with obfuscation, as

well as the combination of VPN and Tor. Further, several different feature selection algorithms and different machine learning classifiers are used on balanced data. Moreover, different tools for the calculation of flow-based features are used and compared.

3 APPROACH

For the investigation of encrypted and obfuscated network traffic, first a dataset was created that contains the corresponding samples. Secondly, these data were prepared for machine learning algorithms, the results of which were evaluated in Section 5.

3.1 Dataset

The dataset for this research contained network traffic generated by several activities like audio streaming, browsing, P2P/SFTP file transfer and video conferencing. Furthermore, all of the above-mentioned activities were recorded several times:

- Without additional encryption (Pure)
- While a VPN was active (VPN)
- While Tor was in use (Tor)
- While Tor with obfs4 was in use (Tor+obfs4)
- While a VPN running through Tor was active (VPN+Tor)

Each activity within the traffic classes (e.g., browsing in Tor) was performed until the recorded network data sufficed to generate at least 1,000 samples using ARGUS and CICFlowMeter (see Section 3.1.1). The VPN traffic was generated by utilizing a commercially available solution from AirVPN [1] while the Tor traffic is recorded on a Whonix [10] gateway (Section 4.1).

3.1.1 Preparation of the Network Data. To use recorded network data for machine learning, it needed to be prepared. For this, the tool ARGUS [2] in version 3.0.8.2, and CICFlowMeter [3] in version 4.0 was used. They extracted and calculated statistical features based on the network flows. A network flow consists of all the packets with the same source/destination IP address, source/destination port and the same protocol (UDP/TCP). Using flows has several advantages. First, it reduces the amount of data significantly as a lot of information like the encrypted payloads are removed. Second, the extracted features can be calculated on all kinds of network traffic, if Tor/VPN is running or not. Lastly, statistical information about the network traffic ignores all randomness from the encryption.

3.2 Machine Learning

Scikit-learn [21] (version 0.21.3), a collection of different machine learning algorithms, was used within this research.

3.2.1 Preprocessing of the Flow Data. To scale the data to appropriate ranges Scikit-learn's built-in functions were used. The necessary conversion of any non-numerical feature or entry into a numerical feature was coded separately. To achieve perfectly balanced classes, the Python toolbox Imbalanced-learn [19] in version 0.5.0 was used, which is fully compatible with the Scikit-learn framework. It offers different methods like down- and up-sampling of classes or even a combination of both in order to balance the dataset again.

3.2.2 Feature Selection Algorithms and Classifiers. During preliminary research on this problem, a dataset containing Pure and VPN+Tor network data had been created. For both cases, certain activities were performed during network traffic recordings. For the calculation of flow-based features, CICFlowMeter was used with different flow timeout values (15, 45 and 75 seconds). This means that a flow, which is longer than e.g., 15 seconds, was split into separate flows. Feature selection was performed using the correlation between features and classes. For classification the algorithms J48, random forest and multi-layer perceptron were used.

These preliminary results showed that random forest performed best by far, while correlation is not suited to select features. Utilizing different flow-timeouts led to contradictory results.

Feature Selection. Based on the data and results of the preliminary research, several additional feature selection algorithms were tested to choose the ones for this research. This included variance, select from model and cross-validated recursive feature elimination. The latter two used decision tree, random forest and extremely randomized trees as base-classifiers. For the tests, the old data was used to compare the results.

Model-based means to use a base-classifier like decision tree or random forest, which by itself calculates the importance of the given features, and uses the provided selection [22].

Recursive feature elimination is a computationally expensive approach, that uses again such a classifier, and recursively eliminates the least important feature [22]. This also enables the observation of the performance of the model in each step, which can be used to select the best performing feature subset.

As a result, variance was used to remove features that are below a certain threshold. This keeps all features that could possibly be of use for machine learning. Further, model-based feature selection, as well as recursive feature elimination based on decision tree, random forest and extremely randomized trees, were used as feature selection methods.

Classifiers. Based on the data and results of the preliminary findings, further classification algorithms were tested, namely support vector machine with a non-linear kernel, k-nearest neighbours, naive bayes, decision tree, extremely randomized trees and bagging with k-nearest neighbours and decision tree. Bagging utilizes multiple instances of the provided base-classifier to improve the overall performance.

As a result, the classification algorithms k-nearest neighbours, random forest and extremely randomized trees were used in this research.

3.2.3 Classification. The task for each classifier was first, to recognize what kind of additional encryption was used. Secondly, the classifier needed to detect the activity performed, while the network traffic was recorded.

For the first case, ARGUS and CICFlowMeter calculated the statistical features based on the recorded network data containing all traffic labelled as Pure, VPN, Tor, Tor+obfs4, Tor+meek or VPN+Tor. Next, each feature selection algorithm created their representative feature set on the provided data. Afterwards, all combinations of classifiers and feature sets were trained and tested using stratified 10-fold cross-validation. To evaluate the results, the scores mentioned in Section 2.3.2 were calculated as well as the confusion matrices.

The second case was approached in a similar fashion as the recorded network data now was split to only contain activities performed during one type of additional encryption (e.g., VPN). This data was labelled as, e.g., "browsing - VPN". ARGUS and CICFlowMeter calculated their statistical information as before and all feature selection algorithms calculated their representative feature sets. Finally, all classifiers were trained and tested with all feature sets as before. This was now repeated until the activities performed during each type of encryption were classified.

Since the choice of algorithms for feature selection and classification was performed on a different dataset than the feature selection and classification itself, no statistical correction procedure (e.g., Bonferroni correction) was used. Additionally, feature selection and classification were compared on two different datasets.

4 REALIZATION

In order to create the dataset, it was necessary to set up the environment with the needed software to create and collect the network data. Furthermore, flows were extracted from the recorded network data, preprocessed and finally used by machine learning classifiers.

4.1 Setup for the Dataset

To generate the dataset, two virtual machines (Alice and Bob) with Ubuntu Desktop 18.04.03 as the operating system were used as clients.

The setup used to generate and capture Pure data consisted of two virtual machines in different networks. For VPN and VPN+Tor (illustrated in Figure 1) traffic, the provided client Eddie (based on OpenVPN) from AirVPN [1] was used in version 2.16.3. Additionally, the Whonix [10] gateway in version 15.0.0.6.6 was used to redirect Tor and Tor+obfs4 traffic from the client into the Tor network, as shown in Figure 2. This gateway is a ready-to-use virtual machine, which acts as a gateway for the client Alice. It is built to send all data into the Tor network and can be further modified to use obfuscation like obfs4 and meek. The software Wireshark in version 2.6.10 was used to record the network data and stored it in pcap files for later use. The recording took place on the client Alice for the cases Pure, VPN and VPN+Tor. For the Tor and Tor+obfuscation cases, the recording took place on the Whonix gateway. For the virtualization, the software Virtual Box in version 6.0.14 was used.

For audio streaming, the software Spotify in version 1.1.10.546 was used. To generate browsing traffic, the browser Firefox in version 71.0 was used on various websites. For P2P traffic, the software qBittorrent in version 4.0.3 was used to download Linux images. To generate SFTP traffic, the client software Filezilla in version 3.28.0 was used, as well as an external SFTP server. For the traffic itself, a small number of generated, incompressible, binary files ranging from 100 to 800 MB were down- and uploaded, while at least twice the amount of data was downloaded. Additionally, several thousand small files ranging from 6 to 250 KB were also down- and uploaded. Lastly, for video conferencing, the Chrome browser in version 79.0.3945.79 and Google Hangout were used. In all cases, the clients Alice and Bob were on separate networks.

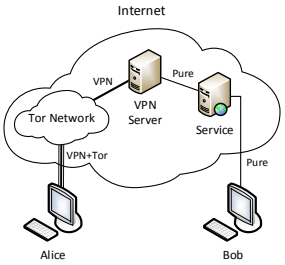


Figure 1: Setup to record VPN+Tor network data on the VM Alice.

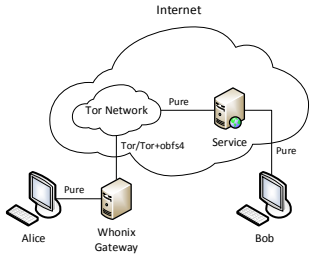


Figure 2: Setup to record Tor network data on the VM Alice.

4.2 Extraction of Flows

The extraction of statistical information about flows was the first step to prepare the network data for machine learning. The following sections describe this process utilizing the software ARGUS and CICFlowMeter.

4.2.1 ARGUS. The pcap files containing the recorded network data, captured by Wireshark, needed to be read by ARGUS to extract flows. First, the pcap files were converted into an ARGUS-specific format with the command `argus -r packet.pcap -w packet.argus`. The second step was to use the created file and extract the flows with their statistical information using the ARGUS module "ra" with the command `ra -F rarc.print.modified.conf -nn -r file.argus`. All project-specific fields like IPs, Ports and IDs were excluded with one exception. The source net was kept in order to filter IPv6 samples later, as all IPv6 related flows were from local communication and had nothing to do with the research. "-nn" was used to prevent *ra* from converting protocols to their names (e.g., 6 to TCP). The output of *ra* was stored in a CSV file. Continuous flows were split into five-second slices.

In the first post-processing step, all IPv6 samples were removed. After the removal, the column "source net" was dropped, as the addresses are project-specific.

The second step was to convert the TCP options field into a usable format for machine learning. Initially, the options were stored in a 12-character long string with a specific character representing the presence of a specific TCP option. As everything else was numeric, this string was converted to binary columns, one for each possible TCP option. After the values of the string were transferred to the new columns, the old column with the TCP options was dropped.

The third step was to transfer the TCP flags in a similar way. One problem is that the field containing the TCP flags is not as thoroughly documented as the TCP options field. But as the string in the TCP flags field was always 10 characters long, using spaces if a TCP flag was not present, a conversion for every character to its own binary column was used.

Finally, the columns "sEnc" and "dEnc" were converted from single characters to binary numeric values, empty cells were filled with a zero and the label column was added.

4.2.2 CICFlowMeter. The second tool used for the extraction of flows was the CICFlowMeter. The flow-timeout was configured

to 5,000ms to be equal to ARGUS's flow-timeout. After the flows were generated and stored in the CSV file, project-specific features were removed (flow id, IPs, ports and timestamp), the data type of the two columns "Flow Byts/s" and "Flow Pkts/s" was changed to numeric, empty cells were filled with zeros and the label column was filled with the filename.

4.3 Machine Learning

After the data is recorded and the CSV files with the statistical information of the flows were generated, several merged CSV files were created. The first file contained all flows, where the labels were exchanged for the type of encryption used. This CSV file was used for the task of classifying the used type of encryption.

The next five files only contained the data of one specific encryption, which were used to detect the type of performed activity, like browsing. The following steps were using those merged CSV files.

4.3.1 Environment to run the Algorithms. For the usage of Scikit-learn, an Ubuntu 18.04.03 server was used. In order to run it, just Python3.6 and Pip3 (version 9.0.1) were needed to install the Scikit-learn package with its dependencies: NumPy (version 1.17.4) and SciPy (version 1.4.0). Additionally, the software library and data structure pandas (version 0.24.2) was installed and used.

4.3.2 Preprocessing of the Flow Data. In order to use the data within the CSV files for machine learning, some preprocessing was necessary. A label encoder from Scikit-learn was used to encode the labels from strings to numbers. In the next step, redundant columns were removed.

Subsequently, the classes were balanced with a random undersampler function provided by imbalanced-learn. It chose samples at random from all classes, but the smallest one, until the size of the classes were equal.

4.3.3 Feature Selection Algorithms. After the preprocessing and the downsampling, feature selection algorithms were used to create different feature subsets. The first algorithm was "VarianceThreshold" provided by Scikit-learn. It removed all features without any variance and all subsequent feature selection algorithms used only this filtered data.

The second feature selection algorithm, called "SelectFromModel", is a classifier-based algorithm, where the classifier calculates the importance of the provided features by itself. The algorithm was used once with a decision tree, once with a random forest and lastly with extremely randomized trees. By default, the feature selection algorithm will select all features whose importance is greater than the mean importance of all the features. For random forest and extremely randomized trees, the parameter "n_estimators" was used to specify that 100 trees were used.

The third algorithm, called recursive feature elimination (RFE) with cross validation (RFECV) is also a classifier-based algorithm. Again a decision tree, a random forest and extremely randomized trees were used, with the same parameters as before. RFE recursively trains the classifier with all features, evaluates the importance of the features and removes the least important one. After this, the training starts again. RFECV uses RFE in a cross-validation loop to obtain the optimal number of features on its own.

4.3.4 Classification Algorithms. For classification, the first algorithm used was k-nearest neighbours. Scikit-learn provides an implementation called "KNeighborsClassifier". Except for the "weights" parameter, all were set to default. For weights, the distance based approach resulted in better results. It simply assured, that closer elements have more influence during the voting of a new element. K is five by default and changing this value did not improve the results. K-nearest neighbours was evaluated using stratified 10-fold cross-validation. The function "StratifiedKFold" was used, which only calculates lists of indices for the training and test data. Using this function ensured that in each step of the 10-fold cross-validation, all results can be retrieved and then used to calculate the evaluation scores. Additionally, it was possible to fit and use a scaler ("MaxAbsScaler") on the training data for each run individually. The same fitted scaler was also used for the test data of the corresponding run. Since the data values in this research were sometimes sparse, the selected scaler was specially designed to handle sparse data and keep the structure of the data. It scales each feature value by the maximum absolute value of this feature, which will be set to 1 [4]. For all following classifiers, training and evaluation were done in exactly the same way as for the k-nearest neighbours classifier.

The second and third classifiers used were random forest using the function "RandomForestClassifier" and "ExtraTreesClassifier", which implements extremely randomized trees. The parameter "n_estimators" was set to 100 and specified the number of trees. Additionally, the seed value for the random state was set to a fixed value.

5 ANALYSIS

The results of the feature selection and classification algorithms, based on the flows provided by ARGUS and CICFlowMeter, are analyzed in the following sections. For the classifiers, the average F1 score was utilized as the primary indicator to rank the results to select the best classifiers. Average means the average of the F1 scores of the classes, which resulted from the 10-fold stratified cross-validation. The true positive rate, false positive rate and precision (on average and per class) as well as the confusion matrix were used as needed.

5.1 Feature Selection

The number of features selected by the variance algorithm represents the upper limit, as all non-static features were included. All other algorithms were based on these features and selected the most relevant ones based on their individual criterion. This led to significant differences as, for example, the model-based approach with a decision tree selected seven features of the Tor dataset, while using it with random forest results in 14 selected features. But these numbers on their own are meaningless, as the results of the different machine learning classifiers based on those feature sets are of interest.

All classifiers were able to achieve better performance with a reduced feature set in comparison to the variance algorithm utilizing features calculated by ARGUS. Besides, all feature selection algorithms were able to reduce the number of features, in most cases significantly.

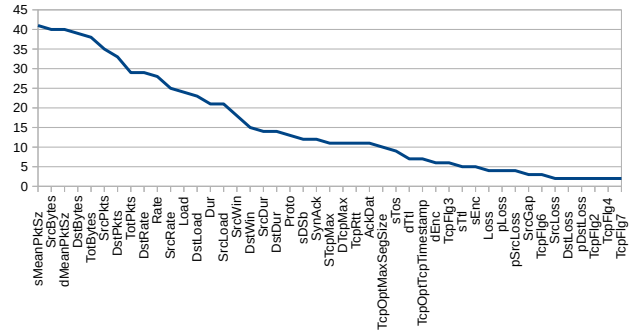


Figure 3: Total frequency of features calculated by ARGUS. The x-axis lists all used 45 features provided by ARGUS while the y-axis displays how often a feature was chosen by a feature selection algorithm.

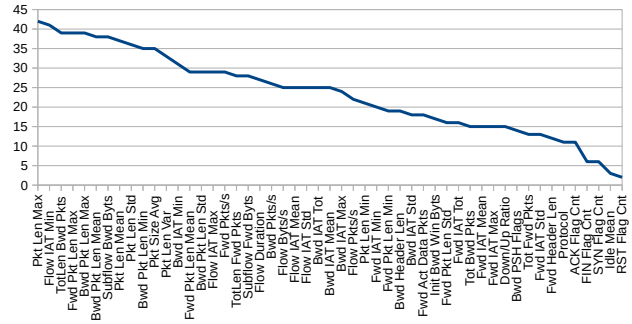


Figure 4: Total frequency of features calculated by CICFlowMeter. The x-axis lists all used 51 features provided by CICFlowMeter while the y-axis displays how often a feature was chosen by a feature selection algorithm.

The usage of the CICFlowMeter resulted in a slightly different picture. Feature selection was most of the time still able to reduce the number of features, but some algorithms only eliminated a few. In addition, the performance improvement from the reduced feature sets in comparison to the full feature set was not as significant as with ARGUS.

The reason for this can be derived from Figure 3 and Figure 4. Those two graphics show rankings of the features provided by ARGUS and CICFlowMeter based on how often a feature was selected by a feature selection algorithm. In the case of ARGUS, there is a leading group of seven features, where each was selected more than 30 times. For CICFlowMeter, 13 features were selected more than 30 times. Together with the fact that the graph for ARGUS is steeper, this clearly shows that the feature selection algorithms were able to reduce the number of features better by using those provided by ARGUS. Overall this indicates, that the ARGUS features were more significant than those from CICFlowMeter. Further analysis to evaluate the performance of this leading group was not necessary, as given the task, the best performing classifier and feature set can always be utilized.

Classifier	Task	Best F1	Feature Sel. Algo.	# of Features
KNN	Major Class	88.62%	Model-based RF	19
	Pure	89.45%	Model-based ERT	23
	Tor	88.94%	RFECV DT	5
	Tor+obfs4	91.78%	RFECV DT	9
	VPN	90.3%	Model-based RF	12
	VPN+Tor	87.85%	Model-based RF	14
RF	Major Class	91.97%	RFECV ERT	13
	Pure	94.61%	Model-based RF	20
	Tor	91.84%	RFECV DT	5
	Tor+obfs4	94.43%	RFECV DT	9
	VPN	92.37%	RFECV DT	6
	VPN+Tor	89.09%	Model-based/RFECV DT	8
ERT	Major Class	92%	RFECV ERT	13
	Pure	97.63%	Model-based DT	10
	Tor	92.28%	RFECV ERT	9
	Tor+obfs4	94.55%	RFECV DT	9
	VPN	92.37%	RFECV DT	6
	VPN+Tor	88.97%	RFECV RF	8

Table 1: Best F1 scores from all classifiers for all datasets based on ARGUS flows along with the used feature selection algorithm and the number of features. If the best F1 score is reached multiple times, the one achieved with lesser features is used.

5.2 Machine Learning Classifiers

To evaluate the machine learning classifiers, the best average F1 score of each classifier for each dataset are listed in Table 1 for ARGUS and in Table 2 for CICFlowMeter along with the used feature selection algorithm and the number of features. The marked rows are the overall best results per task and per feature calculation software.

Based on those results extremely randomized trees was the overall best classifier utilizing flows provided by ARGUS. The classifier was able to achieve the best F1 scores in all tasks but one. Only for the classification of the activities within VPN+Tor traffic random forest performed slightly better. Using the flows provided by CICFlowMeter the results were different as random forest is the overall best classifier for all tasks.

However, when comparing the results between the two tools, overall the classifiers were able to perform better with fewer features when using flows provided by ARGUS. The only task where the flows provided by CICFlowMeter resulted in a higher F1 score was the detection of the used encryption. Therefore, a combined approach using the flows provided by CICFlowMeter to detect the type of encryption and the flows from ARGUS to detect the used application afterwards is the best option.

5.2.1 Classification of the used Encryption. The best F1 score for the classification of the used encryption based on flows provided by ARGUS was achieved by extremely randomized trees utilizing 13 features selected by the extremely randomized tree-based recursive feature elimination function. Table 3 shows the detailed results based on 5,734 samples per class. It reveals that the classifier was able to detect Pure traffic best followed by Tor+obfs4, VPN, Tor and lastly, VPN+Tor. The most surprising result was certainly, that the detection rate of Tor+obfs4 traffic surpasses the one of plain Tor. As obfs4 is designed to disguise the traffic pattern, this was out of place. As this could be simply because of the presence of VPN+Tor

Classifier	Task	Best F1	Feature Sel. Algo.	# of Features
KNN	Major Class	88.16%	Model-based RF	21
	Pure	78.39%	RFECV DT	4
	Tor	64.12%	RFECV RF/ERT	48
	Tor+obfs4	68.61%	RFECV RF	40
	VPN	79.86%	Model-based RF	16
	VPN+Tor	71.76%	Model-based DT	11
RF	Major Class	93.05%	RFECV RF	25
	Pure	87.63%	RFECV RF	23
	Tor	72.78%	RFECV RF	43
	Tor+obfs4	76.57%	RFECV RF	40
	VPN	88.41%	RFECV DT	38
	VPN+Tor	80.79%	RFECV ERT	15
ERT	Major Class	92.3%	RFECV RF	25
	Pure	87.02%	RFECV RF	23
	Tor	71.26%	RFECV RF/ERT	43
	Tor+obfs4	74.75%	RFECV DT	42
	VPN	86.95%	RFECV ERT	36
	VPN+Tor	80.24%	Model-based DT	11

Table 2: Best F1 scores from all classifiers for all datasets based on CICFlowMeter flows along with the used feature selection algorithm and the number of features. If the best F1 score is reached multiple times, the one achieved with lesser features is used.

Class	TPR	FPR	Precision	F1
Pure	98.55%	0.21%	99.14%	98.85%
Tor	88.73%	2.90%	88.46%	88.59%
Tor+obfs4	93.48%	2.95%	96.33%	94.88%
VPN	92.15%	1.31%	94.61%	93.37%
VPN+Tor	86.78%	3.92%	81.99%	84.32%
Average	86.78%	2.26%	92.11%	92%

Table 3: Detailed results for the best cross-validated classification for the major class utilizing extremely randomized trees and the feature set created by the extremely randomized tree-based recursive feature elimination function using 5,734 flows per class provided by ARGUS.

traffic, which consists of Tor traffic at the outer layer, another test was conducted. This time the VPN+Tor traffic was excluded but, nevertheless, the detection of Tor+obfs4 traffic still surpassed the one of plain Tor. Therefore, the only remaining option is, that obfs4 has some specific characteristics, which enabled the classifier to detect it better than plain Tor.

Despite this, the combination of multiple layers of encryption techniques was still able to improve the resistance against detection by machine learning proved by the fact, that the detection result of VPN+Tor traffic was far below the detection of VPN traffic alone and also inferior to Tor.

Table 4 shows the confusion matrix for this classifier. One interesting point is that Tor traffic is mostly misclassified as VPN+Tor and vice versa, which is most likely because the outer layer of the network traffic in both cases was Tor. Additionally, the matrix reveals that the classifier easily discerns VPN traffic from the other classes save VPN+Tor, which has shared characteristics.

The detailed results for the best classifier using the flows provided by CICFlowMeter are shown in Table 5 and are based on 150,000 samples per class. The random forest classifier was used,

Class | classified as →	Pure	Tor	Tor+obfs4	VPN	VPN+Tor
Pure	5,651	3	6	52	22
Tor	2	5,088	120	0	524
Tor+obfs4	1	241	5,360	0	132
VPN	35	0	0	5,284	415
VPN+Tor	11	420	78	249	4,976

Table 4: Confusion matrix for the best cross-validated classification for the major class utilizing extremely randomized trees and the feature set created by the extremely randomized tree-based recursive feature elimination function using 5,734 flows per class provided by ARGUS.

Class	TPR	FPR	Precision	F1
Pure	96.84%	1.02%	95.95%	96.40%
Tor	85.22%	3.95%	84.35%	84.78%
Tor+obfs4	98.20%	5.11%	98.46%	98.33%
VPN	99.46%	0.13%	99.47%	99.46%
VPN+Tor	85.56%	2.18%	87.02%	86.29%
Average	93.06%	2.48%	93.05%	93.05%

Table 5: Detailed results for the best cross-validated classification for the major class utilizing random forest and the feature set created by the random forest-based recursive feature elimination function using 150,000 flows per class provided by CICFlowMeter.

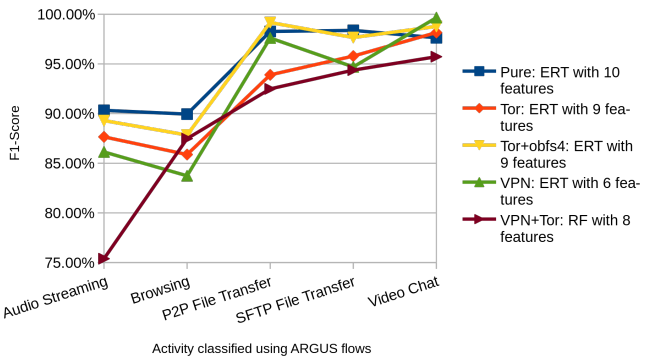


Figure 5: Best F1-Scores for the classification of the performed activity. All flows were provided by ARGUS.

while the feature set was provided by the random forest-based recursive feature elimination function selecting 25 features. It is slightly different from the results of ARGUS as the recognition of Tor+obfs4 and VPN is better to an extent, that both surpassed the one of Pure traffic. This leads to an average F1 score over all classes, which is superior to the one achieved with ARGUS. Besides those two, the detection of all other classes was inferior to ARGUS.

5.2.2 Classification of Activities. The best classification of activities were all achieved by classifiers utilizing ARGUS flows. The corresponding F1 scores are displayed in Figure 5.

The results for the best classification of activities within Pure network traffic using the flows provided by ARGUS are based on 1,096 samples per class. Extremely randomized trees was used as a

classifier, utilizing 10 features selected by the model-based approach based on a decision tree. The results show that the classes P2P file transfer, SFTP file transfer and video chat are the ones that can be classified best with an F1 score above 97%. Audio streaming and browsing are the two classes with slightly worse results, with about 90%. Those two classes were often classified as each other.

The results for the classification of activities within Tor traffic based on ARGUS flows are based on 1,006 samples per class. The classifier extremely randomized trees was used, utilizing nine features selected by the extremely randomized tree-based recursive feature elimination function. Still, the three classes P2P/SFTP file transfer and video chat were classified best (the latter even better than in the Pure case), while audio streaming and browsing were worst. Overall, all classes, other than video chat, were classified worse than before, thanks to the additional layer of encryption. This difference is particularly noticeable for P2P file transfer where the F1 score dropped from 98.27% to 93.9% as it often was misclassified as audio streaming or browsing.

Configuring Tor to use obfs4 should increase the difficulty upon the classification of activities within the network traffic. But as the graph shows, based on 1,023 ARGUS flows per class, this was not the case. Extremely randomized trees was again used for classification utilizing nine features selected by the decision tree-based recursive feature elimination function. All scores for every class improved in comparison to the classification of plain Tor activities. P2P traffic was recognized especially well with an F1 score of 99.17%, which is a bad thing, as this counteracts the purpose of Tor and especially obfs4. It clearly indicates, that the deployed techniques of obfs4 to disguise the traffic pattern (see Section 2.2) are either not useful, or are not used at all.

The best results for the classification of activities within VPN traffic are based on 1,025 ARGUS samples per class. Extremely randomized trees was used for classification while utilizing six features selected by the decision tree-based recursive feature elimination function. It shows that the classification of the classes, other than video chat, was worse than within Pure traffic thanks to the used VPN. However, video conferencing was classified with perfect precision. Despite this, the classes audio streaming, browsing and SFTP file transfer had a lower F1 score as with the usage of Tor. But on average, the classification of activities in Tor or VPN traffic was done with a very similar success rate.

The final stage is the classification of activities within VPN+Tor traffic, based on 1,170 ARGUS flows per class. The random forest classifier utilized eight features selected by the decision tree-based recursive feature elimination function. Other than browsing, which was classified with an F1 score of 87.48%, the results were worse than for VPN alone, Tor or Tor+obfs4. Within Tor+obfs4 traffic the classification of browsing was similarly good with 87.84%. Overall, this indicated clearly, that utilizing two different encryption techniques resulted in a better disguise of the network traffic. Nevertheless, the classification rates were still too high considering that the applied multiple layers of encryption are supposed to hide the used activities inside.

6 CONCLUSION

Summarizing, feature selection improved the performance of the classifiers as well as the runtime, although the recursive feature elimination functions themselves were quite time-consuming. As the feature selection is rarely performed, this can be ignored. RFECV offered most of the overall best-performing feature sets, but no base algorithm could be relied upon in all cases, which was not necessary anyway. The framework outlined in this research can always choose the optimal feature set.

Features of ARGUS had a higher significance as the classifiers achieve better results with fewer features compared to the results based on flows provided by CICFlowMeter. Using ARGUS flows resulted in the usage of fewer features, fewer samples and better results in all activity classification tasks. Only for the detection of the used encryption were the results of the classifiers better by a small margin, when using flows calculated by CICFlowMeter. Contrary, using CICFlowMeter flows resulted in the usage of significantly more features, a vast amount of samples and the best result for the recognition of the used encryption only.

For classification, extremely randomized trees was the overall best performing classifier for all tasks, save one, based on ARGUS flows. As this refers to the recognition of the application within the traffic, the best performing classifier, along with the optimal feature set, can always be chosen based on the results of the detected type of encryption. For CICFlowMeter, this is not necessary. For all tasks, random forest was the best performing classifier, and overall, it would only make sense to use the flows of the CICFlowMeter for the classification of the used encryption.

The impact of this research on encryption techniques like VPN or Tor utilized to disguise the performed activity or even the usage of the encryption altogether is huge. For example, the lowest average FPR for the classification of the used encryption was 2.26% in this framework. This is too high for a regular approach because when classifying thousands of samples in a couple of hours, there would be numerous false classifications leading to many false alarms. However, with additional information, this can be reduced significantly. If it is known that certain network traffic is generated by a specific machine, the consecutively collected samples can be treated differently. As it is highly unlikely that the used encryption will change within 30 seconds, at least six samples can be used to vote for the encryption, which significantly reduces the FPR. As this enables the classifier trained in this thesis to precisely recognize the encryption, the following application detection can always be performed with the optimal classifier and feature set. This is a serious issue, as especially Tor+obfs4 is designed to avoid detection and, as a last resort, should hide the activities of a user. The first objective is crushed and the second is weakened severely.

7 FUTURE WORK

Future work involves building a real-time detection model for the used encryption based on the before mentioned approach by implementing a majority vote. This could be based on six or more samples that are collected and calculated during 30 seconds of network traffic. If only one continuous flow is present during the 30 seconds, ARGUS would slice it into 5-second samples, resulting in at least six samples. If more flows are present, more samples can be obtained.

Additionally, a sensitivity-function, to prevent false classifications due to switching the type of encryption (e.g., turning on/off a VPN) would be needed. It could be implemented as a controller to adjust when an alarm should be sent, e.g., only when all samples were classified the same or only one is classified different etc.). Further, the 30-seconds approach can be implemented as a sliding window (once 30 seconds of network traffic were recorded) to continuously monitor the network traffic for changes, while keeping the false alarm rate very low.

As this is already a practical approach towards the detection of the used encryption, the next step is to investigate to what degree it is possible to discern multiple activities performed at the same time, as this is a common scenario nowadays. Further, the possible disturbance induced by the network traffic of updates performed by the used operating system needs to be considered as well.

Finally, to investigate countermeasures to this machine learning-based classification of network traffic, it is of interest to examine whether enforcing the optional obfs4 time-based obfuscation yields an improvement in regards to privacy. Additionally, to counteract traffic recognition, the obfuscation performed by a VPN can be improved by configuring the usage of padding and the utilization of dummy traffic with various patterns. As dummy traffic generates expenses without a direct benefit, it is quite unpopular and is usually not an option within commercially available VPNs. Nevertheless, it can be configured when the VPN is built on private servers, as a company would do.

8 ACKNOWLEDGEMENTS

The authors thank Markus Ullmann for technical consultation and Stephen MacKay for his help with editing. The authors would also like to acknowledge the financial support of both the Natural Sciences and Engineering Research Council (NSERC) as well as the New Brunswick Innovation Foundation (NBIF) for their support of the research.

REFERENCES

[1] [n.d.]. *AirVPN.* Retrieved 2019-06-05 from https://airvpn.org
[2] [n.d.]. *ARGUS – Auditing Network Activity.* Retrieved 2019-09-17 from https://qosient.com/argus
[3] [n.d.]. *CICFlowMeter.* Retrieved 2019-02-20 from http://www.netflowmeter.ca/
[4] [n.d.]. *MaxAbsScaler – scikit-learn 0.22 documentation.* Retrieved 2019-09-11 from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MaxAbsScaler.html
[5] [n.d.]. *obfs3-protocol-spec.txt_obfs3_doc – pluggable-transports_obfsproxy – Pluggable transport for obfuscated traffic.* Retrieved 2019-11-14 from https://gitweb.torproject.org/pluggable-transports/obfsproxy.git/tree/doc/obfs3/obfs3-protocol-spec.txt
[6] [n.d.]. *obfs4_obfs4-spec.txt at master Âů Yawning_obfs4 Âů GitHub.* Retrieved 2019-11-14 from https://github.com/Yawning/obfs4/blob/master/doc/obfs4-spec.txt
[7] [n.d.]. *scramblesuit-spec.txt_doc - user_phw_scramblesuit – Philippś ScrambleSuit repository.* Retrieved 2019-11-14 from https://gitweb.torproject.org/user/phw/scramblesuit.git/tree/doc/scramblesuit-spec.txt
[8] [n.d.]. *Tor Project: Pluggable Transports.* Retrieved 2019-10-01 from https://2019.www.torproject.org/docs/pluggable-transports.html.en
[9] [n.d.]. *Tranalyzer – About.* Retrieved 2019-06-25 from https://tranalyzer.com
[10] [n.d.]. *Whonix.* Retrieved 2019-12-30 from https://www.whonix.org/
[11] A. Cuzzocrea, F. Martinelli, F. Mercaldo, and G. Vercelli. 2017. Tor traffic analysis and detection via machine learning techniques. In *2017 IEEE International Conference on Big Data (Big Data).* 4474–4480. https://doi.org/10.1109/BigData.2017.8258487
[12] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. Tor: The Second-generation Onion Router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13* (San Diego, CA) *(SSYM'04).* USENIX Association, 21–38. http://dl.acm.org/citation.cfm?id=1251375.1251396
[13] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. 2016. Characterization of Encrypted and VPN Traffic using Time-related Features. In *Proceedings of the 2nd International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,*. INSTICC, SciTePress, 407–414. https://doi.org/10.5220/0005740704070414
[14] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. 1996. Hiding Routing information. In *Information Hiding*, Ross Anderson (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 137–150.
[15] AurÃllien GÃŕron. 2018. *Praxiseinstieg Machine Learning mit Scikit-Learn und TensorFlow: Konzepte, Tools und Techniken fÃ¼r intelligente Systeme.* O'Reilly.
[16] Arash Habibi Lashkari, Gerard Draper Gil, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. 2017. Characterization of Tor Traffic using Time based Features. In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,*. INSTICC, SciTePress, 253–262. https://doi.org/10.5220/0006105602530262
[17] Martin Kappes. 2013. *Netzwerk- und Datensicherheit: Eine praktische Einführung.* Springer Fachmedien Wiesbaden.
[18] Miroslav Kubat. 2017. *An Introduction to Machine Learning, Second Edition.* Springer. https://doi.org/10.1007/978-3-319-63913-0
[19] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. 2017. Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. *Journal of Machine Learning Research* 18, 17 (2017), 1–5. http://jmlr.org/papers/v18/16-365
[20] A. Montieri, D. Ciuonzo, G. Aceto, and A. PescapÃŕ. 2017. Anonymity Services Tor, I2P, JonDonym: Classifying in the Dark. In *2017 29th International Teletraffic Congress (ITC 29)*, Vol. 1. 81–89. https://doi.org/10.23919/ITC.2017.8064342
[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[22] S. Raschka and V. Mirjalili. 2018. *Machine Learning mit Python und Scikit-Learn und TensorFlow: Das umfassende Praxis-Handbuch für Data Science, Deep Learning und Predictive Analytics.* mitp.
[23] K. Shahbar and A. N. Zincir-Heywood. 2014. Benchmarking two techniques for Tor classification: Flow level and circuit level classification. In *2014 IEEE Symposium on Computational Intelligence in Cyber Security (CICS).* 1–8. https://doi.org/10.1109/CICYBS.2014.7013368

An Approach to Represent and Transform Application-Specific Constraints for an Intrusion Detection System

Ayesha Babar, Fahim Imam, Thomas R. Dean
Queen's University
Kingston, Ontario, Canada
{ayesha.babar,fahim.imam,tom.dean}@queensu.ca

Jose Fernandez
Ecole Polytechnique
Montreal, Quebec, Canada
jose.fernandez@polymtl.ca

ABSTRACT

While the need for newer and more efficient network security techniques is increasing, refining the existing and proven techniques can also have potential benefits. One of the aspects of such improvements in the existing systems is making them flexible to modify. Currently, we have an intrusion detection system (IDS) that defines the normal patterns of a network behaviour using constraints. The IDS dissects the network packets into network information to evaluate the constraints. In this research, we extend the existing IDS to validate constraints defined on application data. We extend the IDS to further dissect the data within the incoming network packets. We define the data constraints to identify possible malicious inconsistencies in the application data of a closed network such as the Air Traffic Control (ATC) as an example. We use an ATC ontology for the ATC domain data representation and threat evaluation. We modify an existing ATC simulator and use it to generate both clean and malicious data. Rules and queries are then developed for these data using the ontology to represent detectable threats. The queries are then transformed into application data constraints readable by the IDS. While the transformation is defined as a manual process, the IDS will be updated with automated transformation in the future. The data constraints are written in the same domain-specific language (DSL) already used for the IDS that ensures real-time performance. In this paper, we present our approach to represent and transform application-specific constraints for our IDS along with examples.

CCS CONCEPTS

• **Security and privacy** → **Network security**; • **General and reference** → **General conference proceedings**; • **Networks** → Network reliability.

KEYWORDS

Intrusion Detection, Data Constraints, Program Transformation

ACM Reference Format:
Ayesha Babar, Fahim Imam, Thomas R. Dean and Jose Fernandez. 2020. An Approach to Represent and Transform Application-Specific Constraints for an Intrusion Detection System. In *CASCON '20: 30th Annual International Conference on Computer Science and Software Engineering, Nov 10–13, 2020, Toronto, Canada.* IBM Corp., Riverton, NJ, USA, 10 pages.

1 INTRODUCTION

Cybersecurity is a vital concern as networks surround all aspects of our lives. While the internet may put our information and identity at risk, closed safety-critical network systems such as air traffic control systems (ATC) or nuclear plants may put lives at risk.

Intrusion detection systems (IDS) monitor networks for malicious behaviour. IDSs emphasize the detection of malicious data at the network level. However, malicious data can also occur at the application layer. For example, the ATC systems have come to rely more on Automatic Dependent Surveillance–Broadcast (ADS–B) to extend radar coverage. However, ADS-B has no authentication, and anyone with a software defined radio can transmit false ADS-B data.

We have previously described an IDS designed for network integrity of closed networks such as ATC [31]. This IDS detects abnormal behaviour at the network level using a constraint engine. In this research, we leverage the IDS to detect the presence of attacks at the application layer. It may be true that the application logic is equipped to deal with possible corruption of data. However, application logic is complex due to the fact that it must both validate and operate on the data. Malicious external data is not always obvious. Adding an additional check on the application level data provides in-depth defense to the system.

We use a simulated ATC system to produce simulated air traffic data. This data is parsed and translated into resource description framework (RDF) [14] graph database, using an ATC ontology. We use SPARQL [15] to develop queries that represent the integrity of the information. We then manually translate the domain level threats to the low-level constraints used by our IDS. This allows us to prototype the transformation, and identify changes needed in the implementation of constraint engine to support application level constraints.

The main contribution of our work are:

- Extension of existing constraint based IDS to identify data integrity.
- A specification of transformation of a threat from natural language to a domain specific language, used to generate a custom IDS.
- Testing and evaluation of data constraints with the existing IDS framework.
- Proposing required extensions in the existing framework for new proposed data constraints.

The structure of the rest of the paper is as follows. Section 2 provides a description of the existing framework, ATC simulation and ATC ontology. Section 3 discusses selected threat scenarios of the research. Section 4 describes the transformation process, followed by Section 5 to illustrate the transformation process. The evaluation of IDS and results are presented in Section 6 followed by the related work in Section 7. We conclude the paper and discuss the future work in Section 8.

2 BACKGROUND

To model the cyber threats we use an ATC simulator developed by Morel [26] to generate the ATC data for our IDS. Originally developed by Hasan et al. [17], the IDS detects intrusions based on anomalous network behaviour. The IDS is based on constraints capable of detecting anomalies in a limited access, closed networks such as ATC. Such networks are characterized by a limited number of protocols which makes it possible to define the normal network behaviour as constraints. The current version of the IDS detects intrusions based on protocol-specific constraints. One of the goals of this research is to extend the IDS to specify application-specific data constraints. The IDS can then detect anomalies in the data carried in the network packets. Figure 1 shows the IDS architecture along with the application level extensions. The modified IDS now implements two kinds of constraints: a) protocol-specific network packet constraints, b) application-specific data constraints. We refer to the former as the network constraints and the later as the data constraints.

Application Data. The application layer supports application and end-user processes. It provides application services for file transfers, e-mail, and other network software services [1]. In our research, ATC application data is generated by the air traffic control simulation. This data is embedded in the captured network packets and is parsed by an application data parser. Examples of application data are the speed of an aircraft or the position of an aircraft detected by radar. An example of a constraint on the data is that the speed of an aircraft is within a given range. The constraints that ensure integrity of the application domain data are referred as application data constraints or data constraints.

Network Data. The data captured by the IDS framework that deals with the network layer is referred to as 'network data' for the purpose of our research. This data is parsed by a network parser, and the constraints that check the integrity in this data are referred to as network constraints. Network constraints are already implemented and evaluated by the IDS. An example of this constraint is that the a publisher in the Real-Time Publish-Subscribe protocol (RTPS) [13] has previously declared that it is a participant.

2.1 The Intrusion Detection System

The input to the IDS framework is a network protocol specification written in the Structure and Context-Sensitive language (SCL) [23]. SCL describes the syntax and the semantic constraints of a given protocol. Since SCL supports both context dependant parsing and specifying general constraints, it is used to generate the two main components of the IDS: the *parser* and the *constraint engine*. The generated custom parser reads the network packets and converts them in a format readable by the constraint engine. The constraint

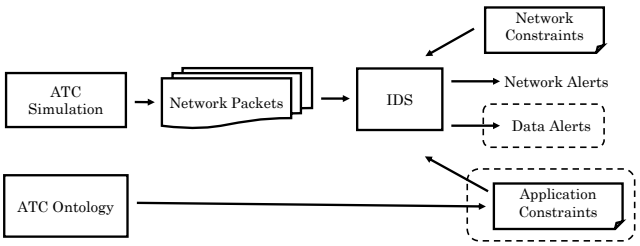


Figure 1: The IDS architecture with extensions.

engine validates these packets against the defined network constraints and generates alerts.

The constraints in SCL are first transformed into an intermediate DSL which describes the constraint tree life-cycle along with memory management. The constraints in the intermediate DSL are then used to automatically generate the constraint engine in C. The DSL describes the constraint tree life-cycle defined by Hasan et al. [16] and has the following four phases: Instantiate (I), Bind (B), Evaluate (E), and Destroy (D). We refer to the DSL as the IBED DSL based on the life-cycle phases. The first, instantiate, occurs when an initial packet of a constraint is encountered. This causes an instance of the internal data structure to be allocated for a constraint tree. The bind phase is used when additional packets are encountered that add information to a constraint tree. The evaluate phase adds the final data to the constraint and evaluates it. Since a constraint may be evaluated multiple times, the destroy phase is used when a packet is encountered that indicates that particular instance of the constraint is redundant. Details about the IBED DSL can be found in Rakha et al. [31].

In this approach, the constraints are intended to validate the last packet in the constraint. The previous packets in the constraint are used to provide needed information to validate the evaluation packet.

2.2 The ATC Simulation

The ATC Simulation is designed and developed by Morel [26] and generates the data used in our research. While the simulation is not a complete representation of an ATC system it provides the necessary components for our research [48]. The ATC is simulated over a closed Data Distributed Service (DDS) network using the RTPS protocol. The main components of the ATC simulation are shown in Figure 2. An existing ATC simulator, Euroscope [8], is used to generate and visualize ATC data using the FSD protocol (FSD and Euroscope in the figure). We use a multiplexer to split the data and transform it to DDS representations of Primary Surveillance Radar (PSR), Secondary Surveillance Radar (SSR), Automatic Dependant Surveillance-Broadcast (ADS-B) data.

2.3 The ATC Ontology

The ATC ontology defines the domain with the help of a controlled and precise vocabulary. When describing the ATC domain, we define the concepts that are present in the domain. For example, *speed* is a concept and it has a meaning and context in our domain. Some concepts can be explained using relations between other

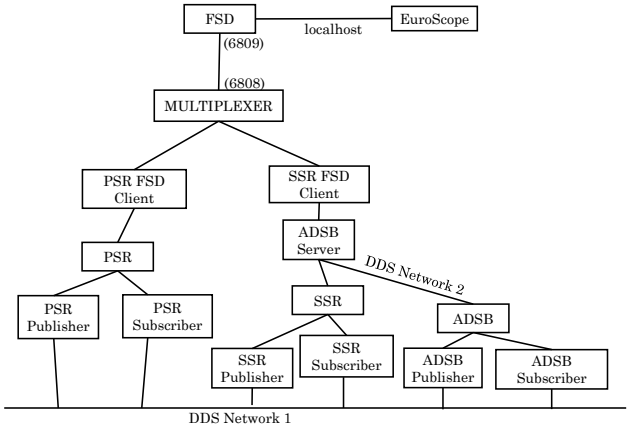


Figure 2: ATC Simulator Architecture Model, adapted from Morel [26]

Figure 3: ATOM detection process, adapted from Corriveau [21].

concepts and/or objects. Some concepts can be described as data structures. The vocabulary used for the ATC domain is concise, and includes classes, sub-classes and relations between them.

The ATC ontology research by Morel [26] is a practical application of the ATOM [21] process. Abstractions-Translation-Ontology-Method (ATOM) is a step-wise method to develop an ontology for a domain specific system.

The ATOM process produces three artifacts: the final ontology, the translation diagram, and the specification document. The final ontology is in the form of a Resource Description Framework (RDF) graph [30]. RDF is the most common way of representing ontologies. In RDF, an ontology is represented as a set of (Subject, Predicate, Object) triplets. The subjects and objects are the nodes of the graph, and the predicate is the property or relation between them. For example, the instance of a concept an aircraft that has a specific speed can be expressed in (PlaneA, hasSpeed, 370).

The ATOM process shown in Figure 3 is used to develop an approach to anomaly detection in the ATC domain. After examining the application data in the network packets, the concepts are identified, and the RDF model is created. The RDF model is then used to initiate the ontology. The nodes are the entities (e.g. airplane, radar, speed) and the edges are the relations between the entities. Further reasoning and flexibility can be added to the ontology by applying rules. The final stage is a querying mechanism, which is used to retrieve and update the information in the ontology. The query language we use is SPARQL (SPARQL Protocol and RDF Query Language) [27]. We have extended Morel's initial ATC ontology[1] for our research.

[1] Available at http://pyxis.ece.queensu.ca/graph/atc/ontologies/atc.owl

New modules have been added to the original ontology to support the representation of Flight Plan data, PSR report, and SSR report. The core ontology is modified to provide better organization to navigate its classes and properties. The current ontology provides the logical framework to consistently describe, query, and reason about different ATC attacks, including the types of attacks described in this paper. The ontology currently includes 72 classes, 39 object properties, 40 data properties, and 250 logical axioms.

2.4 IDS and the ATC Simulator Extensions

To support the evaluation of application-specific data constraints we have extended the existing IDS architecture as shown in Figure 4. An application protocol specification is used to generate a parser for the application specific data encoded in the network data. The ontology from the ATOM Process is shown in the upper right. It is used to initialize the graph database and also to derive a mapping specification that identifies the relationship between the low level data in the packets and the primitive entities and relations present in the data. The RDF mapping is used to automatically generate an RDF translator that populates the graph database with primitive entities and relations. This database can be enhanced with rules and a set of queries are identified that should be continuously evaluated by the constraint engine. These extensions in the upper box have been completed previously.

This paper describes the extensions in the lower box. We transform the queries to a set of application level constraints which is used to generate the application level constraint engine. This is currently a manual transformation and we are working to automate this transformation in the future.

The ATC Simulation was updated to take flight plan information from EuroScope and model as flight strips in the simulated ATC network. It was also updated to allow scripts that inject fake ADS-B data into the simulation.

3 THREAT SCENARIOS

The nature and requirements of command and control systems such as ATC differ from traditional IT systems. Cerchio et al. [9] identify the primary requirements of ATC systems as Integrity and Availability. Cerchio et al. also claim that airborne and seaborne environments are not often considered in security research. While the ground part of an ATC system is a closed network, it still receives outside information without verification. Threats against open communication networks are related "mainly to message insertion (confidentiality), modification (integrity) or suppression (availability)" [34]. Thus, ATC systems are vulnerable to potential attacks some of which are targeted directly at message integrity.

Automatic Dependent Surveillance-Broadcast (ADS-B) has become a key component of ATC systems. The U.S. Federal Aviation Administration (FAA) has required certain aircraft to have installed ADS-B by January 2020 [29]. The threats we consider in this paper are based on this mandate and are information attacks on ADS-B. Balduzzi et al. [2] identify several threats against Automated Identification System (AIS) a system similar to ADS-B for ships. AIS and ADS-B are examples of security critical networks. Both transmit information periodically and are enhance the situational awareness of entities in the system. ADS-B and AIS are subject to attacks

of the same nature, which are to intercept, modify, or delete the messages [22]. We use the same categorization as Balduzzi et al. [2] due to similarities between AIS and ADS-B. Their categorization shows that the attacks can be done at two levels: software and radio frequency (hardware). Among the software identified threats, spoofing and hijacking are two major categories, and both can be modelled under the data related attacks.

We implemented three threat models: Ghost Plane, Physical Law Violation and Spoofed Location. They represent breach to the integrity, confidentiality and authentication of the system. The ghost plane threat scenario simulates malicious ADS-B data of an aircraft that doesn't exit. The threat can be detected if it is within the range of primary radar, is at an altitude that is not in the radar shadow, but is not detected by the radar. This plane can have SSR or ADS-B updates. If outside the range of radar, a ghost plane can be detected if it violates the law of physics. That is, if it descends or ascends faster than the aircraft category, or turns too fast or too slow, or has a speed outside the range of the aircraft type.

Another attack is to monitor existing ADS-B broadcasts for an aircraft and immediately broadcast a new position that overrides the real position. This attack can be detected based on the time intervals of the ADS-B messages.

4 TRANSFORMATION PROCESS

Figure 5 shows the artifacts involved in both the IDS framework and the ATOM process. For both, the network packets generated by the ATC simulator are first parsed into useful data structures. The IDS uses the constraints defined in IBED DSL and protocol specification in SCL, and auto-generates the *C* code for the constraint engine. The constraint engine uses the generated *C* code to evaluate the constraints and ensures network integrity in real-time. The ATOM process translates the parsed network packets into RDF triples using a RDF translator. The resulting RDF is be stored in a graph database. SPARQL [15] queries are used to analyze and understand different aspects of data. The main purpose for these queries is to diagnose and investigate the packet data for constraints that can be used to assess the health of the data. We first transform the SPARQL queries from the ATOM process into SCL constraints and

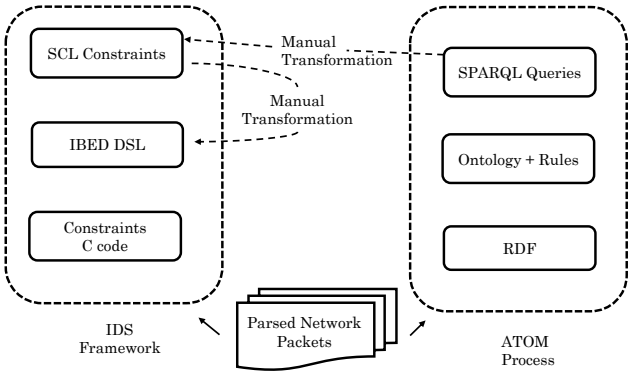


Figure 5: Transformation of SPARQL queries to IBED DSL constraints.

then transform the SCL constraints into the IBED DSL for our IDS framework.

SPARQL queries are used for exploratory purposes by domain experts to formulate the constraints at a high level. The IBED DSL constraints are used to detect intrusions at run time by the constraint engine. Both SPARQL and IBED DSL queries represent the threat in the application data domain which can be expressed in First Order Logic (FOL). The complete transformation process consists of six artifacts as shown in Figure 6.

The first artifact is a query specification in Natural Language. Each of the steps between the artifacts up until artifact 5 (IBED DSL of constraints) are currently manual in nature. The final step, used to generate the *C* code is partially automated. We now describe each step below and explain the involved representations.

Step 1: Query Specification in Natural Language.

We start by naming the queries that represent the respective threat. We define them as a concise statement in natural language. We try to remove as much syntactic or lexical ambiguities as possible. This definition helps in the true representation of the query.

This definition provides a basis for the FOL representation of the queries in the next step.

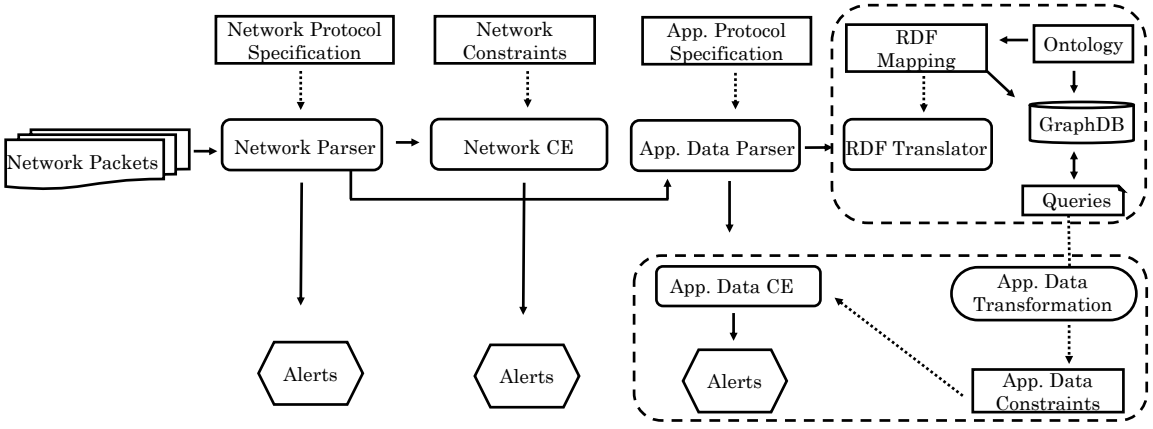


Figure 4: IDS runtime framework architecture.

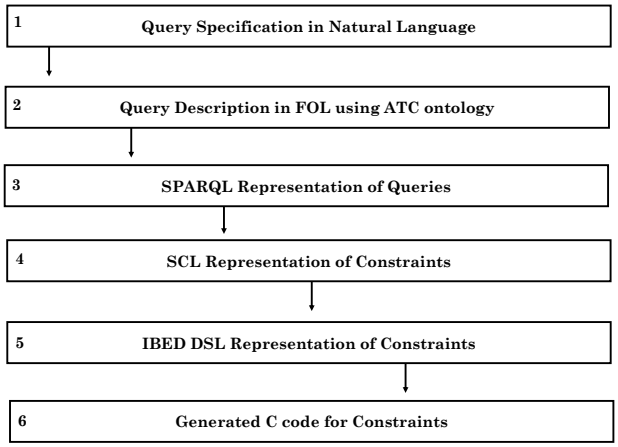


Figure 6: Transformation process from Natural Language to low-level constraint engine code.

Step 2: Query Description in FOL using ATC Ontology. In this step we decompose the natural language description of the queries into concepts and relations using the ATC ontology. Once a query is broken down into basic concepts and relations, we can represent it in FOL. For example, the FOL representation of the statement "if an aircraft has an SSR report and that SSR report has some reported speed s then s is the speed of that aircraft" is:

$$\forall x : Aircarft\ \exists r : SSRReport\ \exists s : ReportedSpeed$$
$$hasSSRReport(x, r) \wedge hasReportedSpeed(r, s) \rightarrow hasSpeed(x, s)$$

The FOL statements are used to construct the SPARQL queries as part of the next step.

Step 3: SPARQL Representation of Queries.

The FOL description use the ATC ontology vocabulary which gives context to the concepts and relations of the queries. The context forms the basis for the SPARQL query representation. We use RDF to represent and store application data. We translate the raw data of the packets to RDF and store in a graph database. After translating the FOL queries to SPARQL, the graph database provides an executable environment to refine the queries and test them against the simulated data.

Step 4: SCL Representation of Constraints.

This step maps the queries from the concepts in the ontological space as expressed by RDF and SPARQL to the network protocol space as expressed by SCL. This moves representation of the queries closer to the network level. For example, the concept `speed` is mapped to the protocol data `SSRModeSType.airspeed`.

The Structure and Context-Sensitive Language (SCL) is an extension of ASN.1 (a network specification language widely used by network engineers). SCL provides a higher abstraction compared to the IBED DSL, but is still attached to the representation and organization of the data given by the protocol specification.

SCL specifies the behavior of the IDS for the incoming packets. Each packet must have one or more constraints that specify the validity of the packet [18]. The SCL constraints specify the how the information in the specified packet depends on information in previous packets. For example, the maximum reasonable speed of an aircraft in an ADS-B packet depends on the type of the aircraft which was a field in an earlier flight strip packet.

```
1 <constraints>
2  <constraint>
3   TYPE: SINGLE-PACKET-ENV
4   VALID-ENV: @RTPS.DATA_P (SrcIP, DstIP, DstPort)
5 </constraint>
6
7  <constraint>
8   TYPE: MULTI-PACKET
9   VALID-SEQ: (1)RTPS.DATA_W, @RTPS.GAP
10  {
11    @RTPS.GAP.SrcIP == (1)RTPS.DATA_W.SrcIP
12    @RTPS.GAP.writerEntityID ==
13        (1)RTPS.DATA_W.writerEntityID
14  }
15  </constraint>
16 </constraints>
```

Listing 1: Syntax convention of SCL SINGLE and MULTI packet constraints [18].

Listing 1 shows an example of two constraints in SCL. The keyword `TYPE` indicates if the constraint is on a single packet (the value `SINGLE-PACKET-ENV`), or if it involves multiple packets (the value `MULTI-PACKET`). The `TYPE` is followed by the sequence of the packets required for the and the type of the target packet. The target packet is prefixed with the symbol '`@`'. For single packet constraints, there is only one packet involved, the target packet.

Listing 1 has a single-packet environment constraint (lines 2-4). Environment constraints refer to entities in the particular environment. Our constraint engine has two modes. When first run on a new system, environmental constraints record the information in the constraints, such as the IP addresses of RTPS participants (DATA_P), or the publishers of particular data topic. As such, environmental constraints list the fields to be memorized as part of the constraint.

For a multi-packet constraint, the target packet is always the last or second last packet in the sequence, as the constraint is written from the point of view of the last packet that triggers the constraint. It may by optionally followed by the packet type that indicates that the instance of constraint is no longer needed (prefixed with the symbol '`~`'. In Listing 1, a GAP submessage in the RTPS protocol must be proceeded by a publisher packet (DATA_W) that introduces the entity id in the gap packet.

Step 5: IBED DSL Representation of the Constraints. Transformation specification of constraint from SCL to an IBED DSL representation is the final step our transformation process. IBED DSL code maps the packets to the constraint tree life-cycle: instantiate, bind, evaluate and destroy. The IBED DSL constraint trees can perform real-time evaluation and provide efficient memory management for the constraint engine [31]. As part of this research the IBED DSL was extended to handle concepts that had not previously been used for constraints at the network infrastructure level. The details of the extension are described in section 6.2

Step 6: Generated C code for Constraints. The IBED DSL is the final step of the manual transformation. The IDS framework

uses the IBED DSL to generate low-level code for constraint engine evaluation. The generated code for the data constraints validate the application data integrity for incoming packets and raises application alerts. The IDS framework uses TXL (a language designed for source code transformation [6]) for auto transformation of DSL code to the C code. As part of this research, the translator was extended for the new concepts identified in step 5. Other than the extensions in the auto-generated code, IBED DSL representation also requires some extensions for correct transformation to c code. For our application domain constraints we manually fixed the generated code for testing.

5 TRANSFORMATION EXAMPLE

In this section, we illustrate the process with one of the threats we identified in section 3

5.1 Violation of Physical Law

This example represents one of the queries that might reveal malicious ADS-B data for an aircraft that doesn't exit. If the reported position of an aircraft is outside the range of a radar antenna, then there is no independent confirmation of the data. A malicious actor might not fully check the data for consistency before broadcasting it, particularly if they are modifying an existing attack. They may get the speed or other characteristics of an aircraft type wrong. This query checks that the speed of an aircraft is consistent with the category of the aircraft.

5.1.1 Step 1 - Query Specification in Natural Language. .

Query Title: The Speed violation of an aircraft at cruising altitude.

Query Definition: The speed of an aircraft is too slow or fast while flying at the cruising altitude, based on the speed range of the aircraft category given by the SSR reports.

Query Description: We identify the ontological concepts such as `Aircraft`, `SSRReport`, `Speed` and `AircraftCategory` along with their relations. Every aircraft has an aircraft category. In the simulation, ADS-B reports are an instance of an SSR data message. They are distinguished from SSR radar reports by the equipment field in the packet. The ADS-B packets for some aircraft may be received (relation `hasSSRReport`). Some SSRReports (there are several type) contain the speed of the aircraft (relation `hasReportedSpeed`).

5.1.2 Step 2 - Description of the Query in FOL using ATC ontology. An aircraft has SSR report and the aircraft is identified with a unique number called `target ID` in these reports. The SSR report has information about the aircraft. Information such as the speed of an aircraft in these reports can be expressed as:

$$\forall x : Aircarft\ \exists r : SSRReport\ \exists s :\ ReportedSpeed$$
$$hasSSRReport(x,\ r)\ \wedge hasReportedSpeed(r,\ s) \rightarrow hasSpeed(x,\ s)$$

The SSR reports for aircraft have other information about the aircraft such as the category of the plane. For example, a Boeing A380 belongs to the aircraft category `C` [19]. Each of these categories has a known minimum and maximum speed.

The ADS-B reports contain the speed of the aircraft, as well as the category of the aircraft. The following query that identifies aircraft whose speed is outside of the range of the category:

$$\forall x : Aircarft\ \exists r : SSRReport\ \exists c : AircraftCategory$$
$$\exists s : ReportedSpeed\ \exists m : MaxSpeed\ \exists l : MinSpeed$$
$$hasSSRReport(x,\ r) \wedge hasReportedSpeed(r,\ s) \wedge$$
$$hasReportedAircraftCategory(r,\ c) \wedge hasMaxSpeed(c, m)$$
$$\wedge hasMinSpeed(c, l) \wedge ((s\ >\ m)\ \vee (s\ <\ l))$$
$$\rightarrow hasViolatingSpeed(x,\ s)$$

Table 1 lists the ATC ontology relations used for the query. The relations `hasMaxCSpeed` and `hasMinCSpeed` refer to the maximum and minimum speed and are not part of the ontology vocabulary. For these queries added as extra relations in the graph database. The Table 1 shows the types of the domain and range for each the properties.

Domain	Predicate	Range
Aircraft	hasSSRReport	SSRReport
SSRReport	hasAirspeed	xsd:integer
SSRReport	hasAircraftCategory	xsd:string
AircraftCategory	hasMaxCategorySpeed	xsd:integer
AircraftCategory	hasMinCategorySpeed	xsd:integer

Table 1: Step 2 - Violation of the Physical Law and Ontology Vocabulary.

5.1.3 Step 3 - The SPARQL Query. The SPAQRL Query in listing 2 is the translation of the FOL query from the previous section. This query is expressed in the same RDF framework as the data mapping that was used to map the application data in network packets to the graph database. This is the first point in time that we can test the query against data from the simulation. This query was successfully run against both clean data from the simulation, and data that contained simulated malicious data.

```
1  SELECT ?assignedTargetID  ?ssrReport
2         ?reprotedSpeed
3  FROM FastInjectedData:
4  WHERE {
5  ?ssrReport ssr:hasTargetID ?assignedTargetID;
6    ssr:hasAircraftCategory ?reportedCategory;
7    ssr:hasAirSpeed ?reprotedSpeed;
8    st:hasMaximumCategorySpeed ?maxCategorySpeed;
9    st:hasMinimumCategorySpeed ?minCategorySpeed.
10 FILTER((?reprotedSpeed < ?minCategorySpeed)
11    || (?reprotedSpeed > ?maxCategorySpeed))
12 }
```

Listing 2: Step 3 - SPARQL Query for the Violation of the Physical Law.

5.1.4 Step 4 - SCL Representation. We transform the SPARQL query to a corresponding SCL constraint. The mapping of RDF elements of the SPAQL query to network fields used in SCL is given in Table 2.

Listing 3 gives a SCL representation of the SPARQL query. This is an extension to the SCL language to allow a logical constraint on the fields of a single single packet. the target packet of the constraint is an SSR Mode S packet. The same packet destroys the instance of the constraint that is created. The second extension to the language is the addition of the domain element that allows the constraint

Ontology Relations	SCL Field Name
ssr:	SSRModeSType
ssr:hasTargetID	SSRModeSTyp.target_id
ssr:hasAirSpeed	SSRModeSTyp.airspeed
ssr:hasAircraftCategory	SSRModeSType.category
st:hasMaximumCategorySpeed	used as a scalar value
st:hasMinimumCategorySpeed	used as a scalar value

Table 2: SCL fields to the ontology relations mapping.

writer to reference elements of the domain. The constraint simply says that the speed must be between minimum and maximum value for the category.

```
1 <constraint>
2  TYPE: SINGLE-PACKET
3  VALID-SEQ: @SSRModeSType, ~SSRModeSType.
4  {(@SSRModeSType.airspeed
5           > Domain.CategoryMinSpeed)||
6   (@SSRModeSType.airspeed
7           < Domain.CategoryMaxSpeed)}
8 </constraint>
```

Listing 3: Step 4 - SCL Representation of the Violation of the Physical Law.

5.1.5 Step 5 - IBED DSL Representation. In the SCL representation of the constraint, it is a single packet constraint and requires comparison of only one value, `airspeed`, for each incoming SSR packet and after comparison it can be destroyed. The IBED DSL is shown in Figure 4.

The constraint starts with the validation tree, that has three values, the speed of a plane, the min and max speed for a category (`categoryMaximumSpeed` and the `categoryMinimumSpeed`).

The DSL requires both an instantiate phase and and an evaluate phase. Nominally these are triggered by different packets and a hash table on values shared between the packets are used to transfer the instance of the constraint tree from one packet to the other. The code for each packet type is generated first for instantiate, bind second, evaluate third and last for destroy. We take advantage of this when generating code to evaluate a predicate on a single packet.

In the instantiate phase on line 6 through line 15 is triggered by an SSRModeSType packet and the values required from the incoming packets are copied to the tree. The notation has been extended with two domain information functions, `DomainLookUpMax`, line 11, and `DomainLookupMin`, that provide external information based on information in the packet. In this case, we use the field `aircarft$category` to find the maximum and minimum speeds of the aircraft. We store the tree instance in the hashtable for use in the evaluate phase.

In evaluate, line 17 through line 24, we recover the tree instance and evaluate it. In destroy we find the tree and destroy it, line 27. As the needed extensions to the constraint engine are in the process of being implemented, a simplified version of the DSL was implemented and after the code was generated, was hand patched to add the needed operators to the evaluation of the tree and in code for the instantiate phase.

```
1  CONSTRAINT AD42
2
3  V( AND( LT(speed, categoryMaximumSpeed),
4         GT(speed, categoryMinimumSpeed) ) )
5
6  INSTANTIATE
7     AppData PDU_AppData.Type is SSRModeSType
8     if not SEARCH Protocol~target$id :Hash=hashIAD42
9         Tree.targetId = Protocol~target$id
10        Tree.category = Protocol~aircraft$category
11        Tree.categoryMaxSpeed = DomainLookUpMax(Tree.
      category)
12        Tree.catgeoryMinSpeed = DomainLookupMin(Tree.
      category)
13        Key = Protocol~target$id
14        HashInstantiate = hashIAD42
15    endif
16
17 EVALUATE
18    AppData PDU_AppData.Type is SSRModeSType
19    HashBind = hashIAD42
20    if SEARCH Protocol~target$id :Hash=hashIAD42
21        Tree.category = Protocol~aircraft$category
22        Tree.speed = Protocol~groundspeed
23        EVAL Protocol~target$id , Protocol~speed
24    endif
25
26 DESTROY
27    if SEARCH Protocol~target$id :Hash=hashIAD42
28        Key=Protocol~target$id
29        HashBind =  hashIAD42
30    endif
31 END
```

Listing 4: Step 5 - IBED DSL Code for Violation of the Physical Law.

6 EVALUATION AND RESULTS

The evaluation shows that constraint engine can be extended to handling not only network constraints but application data constraints as well. It also shows that the ontology and SPARQL can be used to evaluate potential threats in the domain/ It shows that we can implement the SPARQL queries in the constraint engine and enforce them at the network level. In addition to the constraint detecting the violation of physical laws, we applied the process to the other two threats identified in section 3.

6.1 Evaluation of SPARQL

All three threats were expressed as SPARQL queries on our ATC ontology. We generated four data sets. One contains only the clean data from a Euroscope scenario file. We created three scripts that injected malicious data for each of the three threats. Three graph databases were created, each with one set of data. Each query was run against the clean graph database and the malicious data set for that threat. The result of the SPARQL queries is shown in table 3.

In one case, the SPARQL query successfully detected the malicious data, and processed the clean data without incident. In the ghost plane attack scenario, the range of the primary radar was set to the range used by EuroScope. However, this ended up with an edge condition in which an aircraft came into range and broadcast

Threat	Normal Trace	Attack Trace
Physical Law Violation	No alerts	Correct detection
Ghost Plane	1 false alert	Correct detection
Spoofed Location	2 false alerts	Correct detection

Table 3: SPARQL Query Results.

an ADS-B message before the simulated radar detected the aircraft. Revising the query to use a slightly smaller range of radar to ensure that the radar picks up a legitimate aircraft before an ADS-B message is considered malicious.

The third attack scenario is that the position of an existing aircraft is altered by immediately following a legitimate ADS-B message with a malicious ADS-B message with a false position. This query detects this attack by examining the period between ADS-B messages. However, there were two cases in the clean scenario where EuroScope generated legitimate updates that were closer than threshold used in the query.

6.2 Evaluation of IDS

The results of evaluation the application data constraints with the constraint engine are summarized in figure 4. The ghost plane and the violation of the physical law constraints have the same results as the SPARQL queries. The IBED DSL features needed for the spoofed location query were not available, even using the approach of using a placeholder and manually correcting the generated code.

Threat	Normal Trace	Attack Trace
Physical Law Violation	No alerts	Correct detection
Ghost Plane	1 Case	Correct detection
Spoofed Location	Not complete	Not Complete

Table 4: IBED DSL Results.

One of the contributions of this research is to identify the extensions required in the IBED DSL to implement application domain data constraints. The two needed extensions are:

- All three application data constraints rely on external information to be evaluated successfully, such as the 'range of a radar station'. But this information is not available in any packet. We added domain functions such as DomainLookupMax in line 11 of listing4. These allow facts about the real world to be added to constraints.
- The current IBED DSL does not support a constraint on a single packet, as most single packet issues at the network level are handled in the protocol parser. We constraints on single packets that aren't limited to the parsing of the packet.
- The existing IBED DSL implementation has a limited number of logical and relational operators, and no arithmetic operators. These are needed if more general constraints are to be implemented. We generalized the constraint trees to include arithmetic, logical and relational operators.

7 RELATED WORK

There are three areas of related research. The first is redundancy checking and correlation of data. The second is related research in intrusion detection, first order logic and data integrity. The last is work related to the types of threats we investigate.

7.1 Redundancy Checking

Co-relating available information is one of strategies that can be effective in the existing security of any system. This co-relation can be done between different types of data, between data of different systems or between data from different layers of same system. Choo et al. [5] propose that the cyber attacks are 'coordinated' and are 'interconnected'. The main defense of such attacks requires an infrastructure that includes data analytics. Choo et al. suggest that a research challenge is the intelligent analysis of data that is collected from different layers of network security.

Every detection system has the potential to raise false alarm. Ducharme [10] notes that most of the time the consequences of false alarms are resources and time. He notes that to avoid the consequences, it is important to understand the false alarms and be able to co-relate them. Eschelbeck et al. [12] note the importance of the assessment and correlation of data between different systems. They identify the need for correlation of information and used a correlation engine with *Snort IDS* to reduce and validate alerts.

Parnas et al. [28] suggest a "triple redundancy" approach for safety critical systems. The main system of any critical system must perform reliably. Any backup systems must be independent. Parnas et al. suggest that double or triple failure in a disjoint infrastructure is less likely. We do not claim data integrity checking in the IDS is a replacement for data integrity checking in command and control systems such as ATC. Using an IDS to validate application data adds redundancy and more confidence in the overall security of the system.

7.2 Related IDS

Many organizations use security information and event management (SIEM) systems to get an overall view of the information security activity and enforce data integrity [24]. In general, SIEM systems are designed to process security events which are generated by network security solutions [3]. SIEM systems gather a considerable amount of data for analysis from different sources in various formats. SIEM has many advantages, but there are limitations. To make any sense of this data it must be converted into a consistent format [35]. Security reports and dashboards provided by SIEM systems are useful for security staff and management, because they show several security metrics and the general state of information security within organizations [25].

But these reports, logs and alerts contain a significant amount of data. SIEM rules are used to correlate this information. Majeed et al. [20] suggest that many SIEM systems are incapable of giving the status of these rules in real time. Our approach may be adapted to allow critical rules to be validated in real time.

Andrea et al. [4] investigate using an IDS that represent the states of the system using a rules language for Industrial Control Systems (ICS). Like our approach they work with a domain specific network. We focus on data in command and control systems such as ATC. Elfaki et al. [11] also based their intelligent rules on first order logic to better detect inconsistencies. We use FOL for representation of our threats, which are then transformed to IBED DSL constraints.

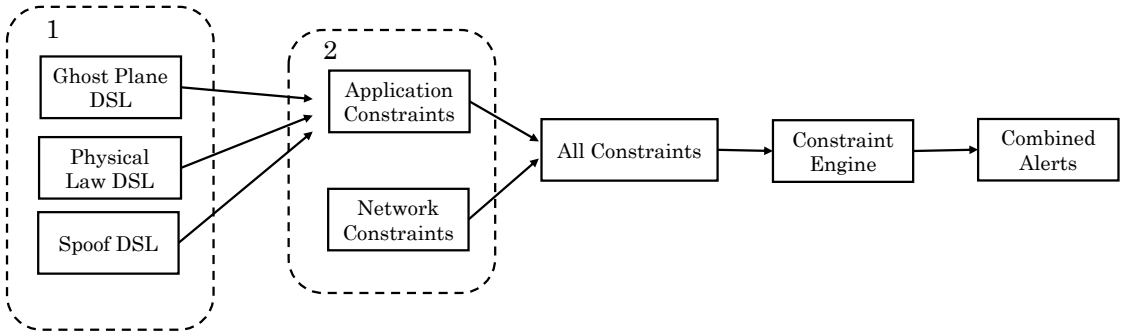


Figure 7: Contributions

7.3 Threats and Attacks

Costin et al. [7] in *Ghost in the Air* identify security issues in ADS-B and show that attacks on the ADS-B are not only possible, but easy. In ATC systems, data provided by ADS-B is trusted and lacks security as a key feature [7] [33]. Costin et al. emphasize adding the most basics authentications. Our physical law violation threat model is a demonstration of the lack of such basic authentication. Abnormal behaviour can be indicative of something that needs to be further investigated.

Ray et al. [32] propose using an ontology for threat models. They suggest starting by familiarizing oneself with the domain by interviewing domain experts before building a threat model. Balduzzi et al. [2] provided a categorization of attacks on AIS. AIS and ADS-B share many of the same vulnerabilities and threats.

8 CONCLUSIONS AND FUTURE WORK

In this research we extend an existing constraint based IDS to identify data integrity at the application level. We demonstrate the extensions in the domain of air traffic control. Figure 7 is a representation of the contributions. We specify a set of transformations from natural language to SPARQL queries to IBED DSL constraints, that can be used to generate a custom IDS which are shown on the left of figure 7. We test our proposed application data constraints with our current IDS framework. The evaluation demonstrates some elements of the DSL and generator that must be extended to fully support application data constraints as shown in region 2 of Figure 7. We show that with the extensions, application data constraints can use the same life-cycle as our network constraints. We propose and present a set of application domain data constraints for the ATC domain, using the same auto-generated framework.

The future work for our research will focus on extensions to the SCL and the IBED DSL. More application data constraints should be evaluated and the work on mutli-packets constraint will be completed. The IDS framework is currently generated semi-automatically. The extensions identified in this research are in the process of being integrated into the constraint engine generator.

The IDS is now capable of working on ensuring integrity in two different aspects of a system, network and data. One interesting dimension would be to explore defining constraints on another aspect or working layer, to see if that adds further security.

In conclusion, some application domain data can be evaluated at the network level. Industrial control systems and command and control applications are often complex, and while security is a critical component, it is one of many components for critical systems. Our approach adds a redundant check of the integrity of application data in the intrusion detection system, where the sole focus is on the system security.

We also provide an example of using the ATOM process to use an Ontology to evaluate application integrity in the air traffic control domain using queries. We then transform them to a low level constraint representation that can be validated in real time.

9 ACKNOWLEDGMENTS

We would like to acknowledge funding from the Department of National Defense.

REFERENCES

[1] ISO/IEC JTC 1. 1994. *ISO/IEC 7498-1:1994 Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model.* International Standards Organization, Geneva, Switzerland.

[2] Marco Balduzzi, Alessandro Pasta, and Kyle Wilhoit. 2014. A Security Evaluation of AIS Automated Identification System. In *Proceedings of the 30th Annual Computer Security Applications Conference* (New Orleans, Louisiana, USA) *(ACSAC '14)*. ACM, New York, NY, USA, 436–445. https://doi.org/10.1145/2664243.2664257

[3] S. Bhatt, P. K. Manadhata, and L. Zomlot. 2014. The operational role of security information and event management systems. *IEEE Security & Privacy* 12 (2014), 35 – 41.

[4] Andrea Carcano, Igor Nai Fovino, Marcelo Masera, and Alberto Trombetta. 2010. State-Based Network Intrusion Detection Systems for SCADA Protocols: A Proof of Concept. In *Critical Information Infrastructures Security*, Erich Rome and Robin Bloomfield (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 138–150.

[5] Kim-Kwang Raymond Choo and Ali Dehghantanha. 2018. Introduction to the Minitrack on Cyber Threat Intelligence and Analytics: A Conceptual Three-Pronged Approach and Future Research Agenda. In *Proceedings of the 51st Hawaii International Conference on System Sciences*. 5521 – 5523. https://doi.org/10.24251/HICSS.2018.688

[6] James R. Cordy. 2006. The TXL source transformation language. *Science of Computer Programming* 61, 3 (2006), 190 – 210. https://doi.org/10.1016/j.scico.2006.04.002 Special Issue on The Fourth Workshop on Language Descriptions, Tools, and Applications (LDTA âĂŹ04).

[7] Andrei Costin and Aurélien Francillon. 2012. Ghost in the Air(Traffic): On insecurity of ADS-B protocol and practical attacks on ADS-B devices. In *BLACKHAT 2012, July 21-26, 2012, Las Vegas, NV, USA*. Las Vegas, UNITED STATES. http://www.eurecom.fr/publication/3788

[8] Gergely Csernak. [n.d.]. EuroScope User Guide, for Version 3.0a. https://www.euroscope.hu/documents/EuroScopeUsersGuide30.pdf. Accessed: 2019-10-29.

[9] R. De Cerchio and C. Riley. 2012. Aircraft systems cyber security. In *2012 Integrated Communications, Navigation and Surveillance Conference*. 1–12. https://doi.org/10.1109/ICNSurv.2012.6218454

[10] É. Ducharme. 2017. *Détection d'intrusion à l'aide d'un système expert basé sur l'ontologie*. Master's thesis. École Polytechnique de Montréal.

[11] Abdelrahman Osman Elfaki, Somnuk Phon-Amnuaisuk, and Chin Kuan Ho. 2009. *Investigating Inconsistency Detection as a Validation Operation in Software Product Line*. Springer Berlin Heidelberg, Berlin, Heidelberg, 159–168. https://doi.org/10.1007/978-3-642-05441-9_14

[12] Gerhard Eschelbeck and Michael Krieger. 2003. Eliminating noise from intrusion detection systems. *Information Security Technical Report* 8 (04 2003), 26 – 33. https://doi.org/10.1016/S1363-4127(03)00004-9

[13] Object Management Group. [n.d.]. The Real-time Publish-Subscribe Protocol (RTPS) DDS Interoperability Wire Protocol Specification. https://www.omg.org/spec/DDSI-RTPS/2.3/Beta1/PDF. Accessed: 2019-05-22.

[14] RDF Working Group. 2014. Resource Description Framework (RDF). https://www.w3.org/RDF/. (2014).

[15] SPARQL Working Group. 2008. SPARQL Query Language for RDF. https://www.w3.org/TR/rdf-sparql-query/. (2008). Accessed: 2020-06-12.

[16] MD Siam Hasan, Thomas Dean, Fahim T. Imam, Francisco Garcia, Sylvain P. Leblanc, and Mohammad Zulkernine. 2017. A Constraint-based Intrusion Detection System. In *Proceedings of the Fifth European Conference on the Engineering of Computer-Based Systems* (Larnaca, Cyprus) *(ECBS '17)*. ACM, New York, NY, USA, Article 12, 10 pages. https://doi.org/10.1145/3123779.3123812

[17] M. S. Hasan, A. ElShakankiry, T. Dean, and M. Zulkernine. 2016. Intrusion detection in a private network by satisfying constraints. In *2016 14th Annual Conference on Privacy, Security and Trust (PST)* (Aukland, New Zealand). 623–628. https://doi.org/10.1109/PST.2016.7906997

[18] Fahim Imam. 2020. *Specifying Constraints in SCL5 for Intrusion Detection*. Technical Report. http://pyxis.ece.queensu.ca/papers/compasstr20-1.pdf/ [Online; Accessed: 2020.02.13].

[19] Legal Information Institute. [n.d.]. Aircraft approach category. https://www.law.cornell.edu/cfr/text/14/97.3 [Online; accessed 14-June-2020].

[20] Abdul Majeed, Raihan ur Rasool, Farooq Ahmad, Masoom Alam, and Nadeem Javaid. 2019. Near-miss situation based visual analysis of SIEM rules for real time network security monitoring. *Journal of Ambient Intelligence and Humanized Computing* 10, 4 (01 Apr 2019), 1509–1526. https://doi.org/10.1007/s12652-018-0936-7

[21] Simon Malenfant-Corriveau. 2017. *PROPOSAL FOR A METHOD OF DEVELOPING ONTOLOGY FOR A SYSTEM EXPERT IN SECURITY*. Master's thesis. École Polytechnique de Montréal.

[22] Mohsen Riahi Manesh and Maima Kaabouch. 2017. Analysis of Vulnerabilities, Attacks, Countermeasures and Overall Risk of the Automatic Dependent Surveillance-Broadcast (ADS-B) System. https://doi.org/10.1016/j.ijcip.2017.10.002. *Int. J. Crit. Infrastruct. Prot.* 19, C (Dec. 2017), 16âĂŞ31. https://doi.org/10.1016/j.ijcip.2017.10.002

[23] Sylvain Marquis, Thomas R. Dean, and Scott Knight. 2005. SCL: A Language for Security Testing of Network Applications. In *Proceedings of the 2005 Conference of the Centre for Advanced Studies on Collaborative Research* (Toranto, Ontario, Canada) *(CASCON âĂŹ05)*. IBM Press, 155âĂŞ164.

[24] Pal Michelberger and Sandor Dombora. 2016. A Possible Tool for Development of Information Security- Siem System. *Ekonomika, Journal for Economic Theory and Practice and Social Issues* 1350-2019-2051 (2016). https://doi.org/10.22004/ag.econ.288703

[25] Raydel Montesino, Stefan Fenz, and Walter Baluja GarcÃŋa. 2012. SIEM-based framework for security controls automation. *Information Management & Computer Security* 20 (10 2012). https://doi.org/10.1108/09685221211267639

[26] L.-P Morel. 2017. *Using Ontologies to Detect Anomalies in the Sky*. Master's thesis.

[27] Ontotext. 2019. What is SPARQL. https://www.ontotext.com/knowledgehub/fundamentals/what-is-sparql/. (2019). Accessed: 2020-02-13.

[28] David Parnas, Jan Madey, and G. Asmis. 1991. Assessment of safety-critical software in nuclear power plants. *Nuclear Safety* 32 (04 1991).

[29] CFR Part. 91. Automatic Dependent Surveillance–Broadcast (ADS–B) Out Performance Requirements to Support Air Traffic Control (ATC) Service. *Final Rule* 91 (91).

[30] Y. Raimond and G. Schreiber. 2014. RDF 1.1 primer. http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/. (2014).

[31] Mohamed Sami Rakha, Fahim T. Imam, and Thomas R. Dean. 2019. Generating a Real-Time Constraint Engine for Network Protocols. In *12th IFIP International Conference on Information Security Theory and Practice (WISTP) (Information Security Theory and Practice, Vol. LNCS-11469)*, Olivier Blazy and Chan Yeob Yeun (Eds.). Springer International Publishing, Brussels, Belgium, 44–60. https://doi.org/10.1007/978-3-030-20074-9_5 Part 2: Real World.

[32] Cyril Ray, Romain Gallen, Clement Iphar, Aldo Napoli, and Alain Boujou. 2015. DeAIS project: Detection of AIS spoofing and resulting risks. IEEE, OCEANS 2015 - Genova, Genoa, Italy. https://doi.org/10.1109/OCEANS-Genova.2015.7271729

[33] SC-186. 2009. *DO-282B, Minimum Operational Performance Standards for Universal Access Transceiver (UAT) Automatic Dependent Surveillance-Broadcast (ADS-B)*. Technical Report. 1150 18th NW, Suite 910 Washington, DC 20036 USA.

[34] Lucio Vismari and JoÃčo Junior. 2011. A safety assessment methodology applied to CNS/ATM-based air traffic control system. *Reliability Engineering & System Safety - RELIAB ENG SYST SAFETY* 96 (07 2011), 727–738. https://doi.org/10.1016/j.ress.2011.02.007

[35] Peter Zegzhda, Dmitry Zegzhda, Maxim Kalinin, Alexander Pechenkin, Alexander Minin, and Daria Lavrova. 2016. Safe Integration of SIEM Systems with Internet of Things: Data Aggregation, Integrity Control, and Bioinspired Safe Routing. In *Proceedings of the 9th International Conference on Security of Information and Networks* (Newark, NJ, USA) *(SIN '16)*. ACM, New York, NY, USA, 81–87. https://doi.org/10.1145/2947626.2947639

Blockchain-based Security for Heterogeneous IoT Systems

Kale Yuzik
Department of Computer Science,
University of Saskatchewan
Saskatoon, SK, CANADA
kay851@usask.ca

Dwight Makaroff
Department of Computer Science,
University of Saskatchewan
Saskatoon, SK, CANADA
makaroff@cs.usask.ca

ABSTRACT

The Internet of Things (IoT) is being deployed in industry, public services, and even homes. These devices are making information more available and allow for greater automation and efficiencies. With the rapid growth this industry is experiencing, the security of IoT devices has not been given the attention it needs. Many of these devices leave sensitive information exposed or may allow for malicious actors to take control of them. The Internet of Things uses a vast range of hardware which has led to many different approaches to security. Administering a network with such variability makes it easy for insecure configurations to be overlooked.

This paper proposes the use of blockchain technology as the backbone to a security framework to unify IoT devices of varying resource constraints under one system. Ethereum is used to create a secure system that is Denial of Service resistant, store encryption keys, store encrypted data, and manage trust of devices. Using the Proof-of-Authority consensus method instead of the more common Proof-of-Work, allows for more efficient use of resources. This system features mechanisms to include the use of LoRa LP-WAN technology, which is often used in IoT. Tests were run on a small network of devices while recording processor utilization. Latencies were also measured, showing that devices with fewer resources showed significant latencies, and suggestions as to how these latencies can be reduced are proposed.

CCS CONCEPTS

• **Information systems** → *Information systems applications*; • **Computer systems organization** → **Peer-to-peer architectures**; • **Security and privacy** → **Key management**; **Security services**; • **Networks** → *Network services*;

ACM Reference Format:
Kale Yuzik and Dwight Makaroff. 2020. Blockchain-based Security for Heterogeneous IoT Systems. In *Proceedings of CASCON 2020 (CASCON'20).* IBM Corp., Riverton, NJ, USA, 10 pages.

1 INTRODUCTION

The Internet of Things (IoT) is experiencing a rapid expansion in growth. ARM predicts that one trillion IoT devices will be manufactured between 2017 and 2035, and world will see a $5 trillion boost in G.D.P. due to the industrial use of IoT technologies by 2035 [27]. The Internet of Things offers great value for uses such as monitoring critical infrastructure, which will inevitably lead to the deployment of these systems throughout cities. Manufacturers are driven by economic factors and those fastest to market benefit the most. This encourages manufacturers to cut corners and take calculated risks and there is no exception when it comes to the security of these products. Many of these IoT devices are deployed in remote or inaccessible locations and use low bandwidth connections. This makes servicing or updating them far more challenging than conventional computer networks. As the use of IoT systems expands, the risks involved with failure or security breaches become increasingly severe.

IoT traffic lights have been developed to synchronize with other traffic lights within road networks to minimize delays and reduce congestion [1]. While the benefits of smart road infrastructure are considerable, if targeted by an attacker, traffic collisions could be caused, putting lives at risk. Commercially available, internet connected cardiac implants were found to contain a critical security vulnerability [28]. This not only exposed data collected by the implants, but the administration of shocks by pacemakers and defibrillators could be altered. This documented vulnerability is irrefutable evidence that with the growing adoption of IoT technologies the benefits are immense, but the cost of breaches will be financially expensive and may endanger lives. For these reasons, it is critical these systems be secure at the time they are deployed.

Current approaches to IoT networks employ cloud-based services to collect and process data from IoT devices. These cloud-based IoT services (such as The Things Network[1]) introduce a single point of failure by means of an external agency. Should the cloud service become compromised, all guarantees of data confidentiality, integrity, and availability are lost. This exposure may be acceptable for some applications, but for critical services for which society may come to depend upon, minimizing/eliminating these exposures is vital. A blockchain-based security framework is proposed to address these issues with cloud-based IoT services.

Due to the inexpensive and low-power hardware used for IoT systems, five categories of constraints apply: compute power, memory capacity, persistent storage capacity, connectivity bandwidth, and power source. Some limitations that may exist for one IoT device may not be an issue for another. Given this broad range of devices, the question of how to design a security framework that caters to the needs of these heterogeneous devices arises. Using dissimilar solutions for devices of varying hardware resources is not only cumbersome, but introduces security concerns itself. With more solutions come greater potential for configuration errors and complexity of administration.

CASCON'20, Nov. 10–13, 2020, Toronto, Canada

[1] https://www.thethingsnetwork.org/

This paper explores the application of blockchain technology to create a unified security framework for IoT devices with heterogeneous compute resources. The remainder of this paper is organized as follows. Section 2 describes the component technology of the problem domain, while Section 3 gives a brief overview of similar previous work. Section 4 outlines the implementation and configuration of the test environment, Section 5 provides proof-of-concept results for the test network, and Section 6 draws some insight and analysis. Section 7 provides a summary and suggests future directions.

2 BACKGROUND

When assessing information security there are three fundamental qualities that must be considered. These qualities form what is referred to as the CIA triad: Confidentiality, Integrity, and Availability [18]. Confidentiality refers to the secrecy of data from those who are not authorized to view or access it. Data can be kept confidential using cryptographic techniques. Integrity is an assurance that the data can neither be altered or forged. The integrity of data can be protected through the use of digital signatures. These signatures provide means to authenticate the origin of the data as well as detect if the data has been altered. Availability is the property that adversaries cannot prevent or hinder access to data or services required to process/transmit/receive that data. This can be guarded using peer-to-peer technologies to provide redundancy, thereby increasing the availability of data.

Decentralized technologies such as blockchain present a unique advantage over the traditional client-server model. They offer a resistance to Denial of Services (DoS) and Distributed Denial of Service (DDoS) attacks [19], owing to the lack of a single point of failure [2] and distributed ledger containing the desired data. Cisco has projected 15.4 million DDoS attacks will occur in 2023, nearly double the 7.9 million which were expected in 2018 [12].

2.1 Blockchain

Dwork and Naor [15] first introduced the idea of Proof-of-Work, a way of providing means for making an assertion without the need of cryptographic trust, a precursor to Blockchain. Vishnumurthy *et al.* [29] made use of the concept of Proof-of-Work by creating a credit system to incentivize equal contribution of all nodes within peer-to-peer systems. This system provided a public ledger of transactions and involved the payment of "karma", a digital token for work performed by peers. Nakamoto [24] developed the idea of a decentralized, anonymous digital currency, now known as Bitcoin.

Blockchain is essentially a distributed database [21] that consists of chunks of data (blocks) that are linked together in a linear order. Each block contains the cryptographic hash of the block prior [24].

In the Proof-of-Work (PoW) consensus scheme, miners assemble a block with pending transactions. A miner assigns an arbitrary value to the nonce (number used once) field and calculates the hash of this proposed new block. The miners then check if the hash is less than the difficulty value [3]. When a miner succeeds, it broadcasts its newly mined block to connected peers, who then verify its validity. If valid, the network accepts the block, and work begins on the next block. Miners race with others to find values which satisfy these criteria. The difficulty is adjusted to maintain a predetermined duration of time between creation of new blocks, which is called the "block time".

A change in any block along the chain will result in one of these hashes not matching. For an attacker to successfully alter an existing portion of a blockchain, they must re-mine every block from the victim block on until the length of their altered blockchain exceeds the length of the currently accepted chain.

Finding a hash which meets the required difficulty parameter involves continual computation [3], and because mining is a race for the next block, it is only viable on hardware above a threshold of computational power. For this reason, Proof-of-Work is an impractical solution for securing a blockchain running on a network of low power IoT devices.

As an alternative to Proof-of-Work consensus, a voting-based system known as Proof-of-Authority (PoA) [13] may be used, in which blocks are approved (or rejected) by authorized accounts known as signers. The use of a PoA consensus algorithm creates a permissioned blockchain, whereas with PoW the blockchain would be permissionless. De Angelis *et al.* [13] analyzed permissioned blockchain consensus algorithms in terms of the CAP (Consistency/Availability/Partition tolerance) theorem [16] and performance. The implementations of PoA known as Aura and Clique were examined, as well as *Practical Byzantine Fault-Tolerant* (PBFT) schemes. While there were trade-offs in terms of the CAP theorem, Clique requires the least number of messages to achieve consensus, thereby making it advantageous for use on resource constrained systems.

On PoA, signers approve blocks by signing them with their cryptographic key and for a network to consider a block as valid, it must be signed by a majority of the authorized signers. Upon genesis of the blockchain, initial signers are defined. Accounts which maintain the transaction process of the blockchain accumulate positive reputation. Thus, signers can be voted in or out, based on their reputation within the blockchain network. This system eliminates the computationally demanding operations required by the Proof-of-Work scheme. Additionally, PoA allows for the block time to be explicitly set, thus allowing for some degree of control over the latency of contract functions which mutate the contract state and by extension, the latency in our proposed system.

2.2 Ethereum and Smart Contracts

Blockchain is best known for its use in implementing cryptocurrencies, but its applications are far more broad. Smart contracts are compiled code that is uploaded to the blockchain [30, 31]. These contracts contain functions that may be executed in a distributed manner as required. Contracts can contain persistent state information that is global to all devices on the blockchain. In order for the results of contract execution to be accepted by the network, there must be consensus on the postconditions of execution.

Smart contracts, as they are referred to in the context of Ethereum, contain functions which are divided into two groups: those that modify the contract state and those that do not. They have substantially different performance properties. Contract functions modifying the contract's state are called by sending a transaction [30]. This is done by broadcasting the transaction data to other devices

mining on the blockchain, for which the outcome state of the contract must be agreed upon by the miners. The mining nodes execute the function and must come to a consensus, introducing a latency which is primarily dependent upon the block time. Contract functions that do not modify the state variables of a contract need only be executed locally on the device. There is no need to come to a consensus on the result of this computation, as it does not modify public information in any way, so the latency is imperceptible.

The Ethereum Virtual Machine (EVM) is used to execute smart contract functions. EVM allows for looping, and thus introduced the possibility of poorly written or malicious code to invoke an infinite loop. As a remedy, the concept of "gas" is introduced. Each instruction depletes a finite quantity of gas allotted to a transaction. The sender of a transaction may choose the initial amount of gas available; the spent gas determines transaction fees charged to the account from which the transaction originated.

Ethereum accounts each possess their own pair of cryptographic keys that are used to sign transactions. These same keys are used to sign blocks when using Proof-of-Authority consensus.

2.3 Cryptography

Symmetric cryptography uses the same key to both encrypt and decrypt information for both parties, whereas asymmetric cryptography involves both actors having different cryptographic knowledge and abilities [17]. Asymmetric ciphers are more computationally expensive and a solely asymmetric approach to encryption is not feasible in many domains. Algorithms such as Diffie-Hellman allow for a shared key to be derived only from knowledge of one's own key-pairs and the partner's public key, preventing third parties from determining the shared key. Another unique advantage with asymmetric cryptography is the possibility for data to be digitally signed and verified [17], providing a very high degree of confidence that the data originated from the believed source (the actor which possesses the specific key-pair).

Both Rivest-Shamir-Adleman (RSA) and Elliptic Curve Cryptography (ECC) are asymmetric cryptographic systems. They both provide the same functionality, but differ in underlying mathematics, computation difficulty, and security [22]. As keys become larger, the security the cipher provides increases. When an ECC key becomes larger, RSA keys must grow at a disproportional rate to be able to match the level of security [20]. ECC can offer an equal level of security with a much shorter set of keys.

The Advanced Encryption Standard (AES) symmetric cipher has been heavily used since its acceptance by NIST in 2001 [25]. AES uses keys of either 128, 192, or 256 bits, with 10, 12, and 14 rounds respective of key length. There have been limitations and shortcomings identified with AES in the intervening years. A cache timing-based attack [6] on AES exposed the possibility of key recovery. This not only breaks the confidentiality of the current ciphertext, but all other messages that are encrypted with the same key.

The Salsa20 stream cipher [9] offers encryption that is consistently faster than AES. Salsa20 may be applied using a differing number of rounds, with Salsa20/20 (20 rounds) being the recommended standard. Cryptanalysis of Salsa20 has shown Salsa20/8 or fewer rounds to be vulnerable to attacks [5, 9].

The Salsa20 family of ciphers uses 3 operations: 32 bit addition, 32 bit XOR, and constant-distance 32-bit rotation. These instructions are all CPU friendly, and therefore faster across a wider number of platforms than other ciphers such as AES [9]. Due to the lack of S-box lookup tables, Salsa20 also avoids the cache timing attacks possible with AES.

XSalsa20 specifies a longer nonce than Salsa20 (128 bits vs. 64 bits) [8]. The nonce does not need to be secret; a third party obtaining the nonce does not compromise security of the cipher. The longer nonce makes it safe to use a randomly-generated nonce. XSalsa20 offers the exact same speed as Salsa20, with the minimal extra cost of generating the larger nonce.

A cipher with a higher degree of diffusion does a better job in hiding the relationship between plaintext and ciphertext. The family of ChaCha ciphers [7] is based on the Salsa cipher and provides improved diffusion. This modification does not increase the computational expense, nor does it reduce the potential for parallelism. In fact, ChaCha20 uses one fewer register than Salsa20, which on some platforms may yield minor performance gains. Aumasson *et al.* [5] performed a differential cryptanalysis of Salsa20 and ChaCha. They found that while they could break up to 8 rounds of Salsa20, they were only able to break up to 7 rounds of ChaCha (ChaCha7). For symmetric encryption, XChaCha20 was selected due to its improved strength against cryptanalysis over other variants in the Salsa20 family, its imperviousness to side channel attacks, and CPU friendly operations which allows for efficient operation on embedded systems.

3 RELATED WORK

Biswas and Muthukkumarasamy [10] conducted an analysis of smart cities and how blockchain technology could be used to provide a security framework to protect them. These researchers point out that IoT devices used in smart cities utilize various communication layer technologies such as Ethernet, Wifi, Bluetooth, 6LoWPAN, 3G, and 4G. They argue a security framework should support these technologies and allow for communication between differing communication systems. The recommendation for use of a permissioned blockchain was made over an permissionless blockchain, due to faster consensus and reduced potential for anonymous attacks.

Huh, Cho, and Kim [19] proposed an Ethereum-based system for managing RSA public keys as an IoT management system. Their proof of concept modelled electrical appliances and monitored power consumption. Smart contracts provided an interface to set a power usage limit when the devices would be automatically turned off.

Özyilmaz and Yurdakul [26] investigated an Ethereum-based IoT data collection system. Wireless nodes used LoRaWAN to communicate with a "smart proxy" that performed blockchain-related functions. This work focused on blockchain technology for decentralized storage and robust data availability, but did not employ cryptography to ensure data confidentiality. Data was stored using Ethereum's SWARM storage service, a peer-to-peer data storage system. Many of the design aspects of Özyilmaz's work will be used in the formulation of the system in this paper.

Minoli *et al.* [23] conducted an analysis of blockchain technology in the scope of providing security for IoT. Proposals were made

for the different roles a "Network Element" (NE) may serve in the greater scope of the network/blockchain. Some of these configurations defer protection of data integrity to other more powerful devices within a network to account for NEs which may be less capable of securing the integrity on their own. These devices include gateways, and concentration nodes (routers, switches, firewalls, etc.). Additional uses for blockchain systems for IoT are also suggested, including device configuration, data storage, micro-payments, automated payments between things to create a shared economy, Digital Rights Management (DRM), history of ownership throughout the supply chain, smart cities, device communication/synchronization, and software rollout.

Dorri *et al.* [14], examined the design of the blockchain itself in the context of smart home IoT. The authors highlighted barriers to using cryptocurrency-based blockchain systems with IoT systems. These issues include high consumption of system resources, latency, and scalability problems arising from the need for consensus among nodes. A layered design of the network is proposed, involving no need for use of Proof-of-Work. In one layer, a private blockchain is used to connect a group of devices within a home. One device in the home with plenty of computational resources is designated the Smart Home Manager (SHM), which acts as the miner. At the top layer, smart homes are connected to a public blockchain (independent from the private blockchain), for which the SHM relays transactions, and communicates with cloud-based services. This separation of blockchains greatly reduces the storage needs of the resource constrained IoT devices, as well as reduces the bandwidth and energy demands placed on them.

4 METHOD

Ethereum will be used as the underlying blockchain technology due to the Turing complete virtual machine it makes available for distributed computation. While other blockchain technologies such as Bitcoin also make scripting possible, these alternatives are not Turing complete [11, 19], as looping is not possible. This design choice greatly limits their practicality for use in our framework.

The resource constraints of the IoT devices restrict our design parameters. In order to encompass this range of devices into one system, a proxy is built into the design. This allows devices that are incapable of running an Ethereum client to participate in the network. The programming language used must allow for efficient use of hardware, and allow for multiple threads to make best use of resources. We chose C++ with a custom system to manage communication with our selected Ethereum client (Geth, see Section 4.2) and its JSON API through Unix domain sockets, as the commonly used Web3.js library is written for JavaScript.[2] This keeps resource consumption as low as possible.

A conceptual overview of the proposed system is found in Figure 1. Three different types of devices exist: devices running a Geth client, without LoRa (Section 4.5.1), devices running a Geth client and operating as a LoRa proxy (Section 4.5.2), and devices not running Geth with LoRa, requiring the services of a proxy (Section 4.5.3).

[2]https://web3js.readthedocs.io/en/v1.2.6/

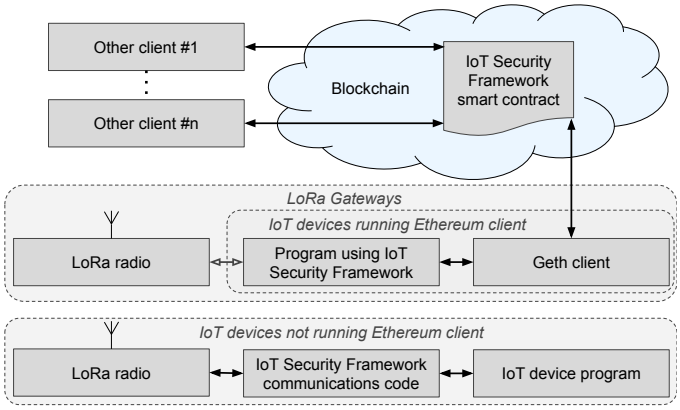


Figure 1: Architecture of the network

4.1 Design Considerations

4.1.1 Compute Power. Embedded systems generally possess low compute power. This amplifies the trade-offs when selecting cryptographic algorithms. The trade-offs between computational latency and security become far more pronounced than on devices with faster processors. Security will be prioritized when reasonable, while minimizing computational complexity.

4.1.2 Memory and Storage. On some devices, memory and storage become severely limited, in some cases as low as tens of kilobytes. Offloading much of the work to a proxy/gateway will minimize the memory footprint of the compiled binary for these platforms.

4.1.3 Network Bandwidth. Many IoT devices use wireless communications to perform their functions. One such common technology is LoRa [4]. LoRa allows for throughput ranging from 0.3 Kbps to 50 Kbps, depending on configuration and regional differences [21]. The proxy solution must operate over these low bandwidth connections, while maintaining a high degree of security.

4.1.4 Power Source. Very often, IoT devices have limited power supply such as batteries or solar power. Both the processor and wireless radios can be significant consumers of energy; minimizing power consumption is important for the feasibility of a solution.

4.1.5 Cryptographic Functions. The required cryptographic functions consist of public/private key generation, Diffie-Helman key exchange, a symmetric key cipher, and signature creation and verification. ECC was selected over RSA, due to both its smaller key size without compromised security and computational speed, which is well suited to embedded systems [22].

4.2 Ethereum

The blockchain security framework was tested on a private Ethereum network, using Proof-of-Authority as the method of consensus. Using PoA over PoW allows for more devices to participate in the voting process, as compared to the mining process in PoW. This makes the security of the blockchain dependent on the quantity of signers, rather than the mining compute power. In general, this lends itself well to a network of IoT devices, since such networks

often consist of hundreds or thousands of devices. Additionally, control over block time is an advantage since this will directly impact the latency of data sent by devices on the network.

A block time of 5 seconds was used, as it provides lower latency than the block time of 12 seconds used on Ethereum's public blockchain which uses PoW consensus. While 1 second would result in even lower latency, the rate at which storage costs grow must also be weighed. A test was carried out with an Ethereum blockchain, a 5 second block time, 1 signer, and no transactions being made. The size of chain data on the filesystem was recorded at 5 second intervals, which showed the chain grew at an average rate of 3465 bytes per block.

4.3 Go Ethereum Client

We selected the Go Ethereum client (Geth).[3] The implementation of Proof-of-Authority used in Geth is known as Clique. Geth provides many options and modes which allow for control over the extent that the blockchain is stored and verified locally. These settings allow for some adjustment over the use of system resources, such as processor, memory, storage, and bandwidth. It has three different modes of operation/communication: full sync, fast sync, and light sync.

In **full sync** mode, Geth stores the entire blockchain on the device and verifies every block created and transaction contained within the blocks. This is the most resource demanding mode of the three. As the blockchain adds blocks to the chain, the costs of storing the chain increases. **Fast sync** mode, like full sync mode, obtains all blocks since genesis and verifies all blocks, but does not verify transactions, until a set number of blocks behind the present head of the blockchain.[4] This mode trades some processing power for bandwidth. Once a fast sync client has obtained the entire chain, it functions the same as full sync mode. **Light sync** mode consumes the least amount of system resources, with the exception of bandwidth. In this mode, all block headers and data are downloaded, but transactions are not obtained. Geth only randomly validates blocks in light sync mode. The use of light mode requires full sync mode devices on the network to serve light clients which must be explicitly enabled on the full sync client.

4.4 Contract Design

The smart contracts will be used to store the following information for each device:

- Human friendly name,
- Numeric ID ("device ID"),
- Device creation timestamp,
- Public encryption key,
- Public signature key,
- Encrypted data & nonce,
- Timestamp of when data was last received,
- Numeric ID of the decrypting device ("data receiver"), and
- whether this device is managed by a gateway/proxy (T/F) ("gateway managed").

[3]https://geth.ethereum.org/

[4]Szilágyi, Péter. October, 2015. eth/63 fast synchronization algorithm #1889. https://github.com/ethereum/go-ethereum/pull/1889

The contract facilitates the allocation of new devices, removal of devices, changing of cryptographic keys, and storage and retrieval of encrypted data. Some of these are administrative functions that should only be callable by authorized Ethereum accounts. The contract allows for an arbitrary number of Ethereum accounts to be granted access to call such functions.

Each device is assigned a partner device which may decrypt the sender's data, termed a "data receiver". Allowing for a specific data receiver to access data from one or more devices permits data privacy even with multiple users within a blockchain security framework. This limits potential damage if a cryptographic key becomes compromised.

4.5 Devices

The hardware utilized in the test network consists of 3 types Raspberry Pi devices and AdaFruit Feather M0 devices[5] as described more fully in Table 1. In addition to the information in the table, the Raspberry Pi 2B+s are equiped with a Dragino LoRa (SX1276) & GPS HAT. One AdaFruit Feather M0 device uses a 1200mAh LiPo battery and the other devices are mains powered. All LoRa chips are of the RF9X family, operating in the 915MHz frequency range. All Raspberry Pis ran Raspbian on a headless installation. The devices were run in a network as illustrated in Figure 2.

4.5.1 Devices Running Geth Client, without LoRa. Devices which are connected to the Internet and have sufficient system resources will run the Go Ethereum client locally. The blockchain security framework will communicate with Geth through Unix domain sockets to request services of the smart contract. Only devices with more capable hardware will run Geth in full sync mode as a signer. Other devices will be tested in light sync mode.

4.5.2 LoRa Gateways/Proxies. Devices operating as a LoRa gateway constantly listen for transmissions from broadcasting LoRa devices and run a local instance of Go Ethereum. When the gateway receives an incoming message, it retrieves the public signature key that corresponds to the device ID in the LoRa packet from the smart contract. If the signature is verified as valid using the public key, the gateway can be confident the message originated from the claimed device. Since the gateway is already registered on the blockchain, the smart contract implicitly trusts the gateway and the gateway may forward the already encrypted message payload to the blockchain. In order for the smart contract to permit the gateway acting on the behalf of the device, the device must be registered on the blockchain as "gateway managed". Any gateway is capable of pushing the data of any registered LoRa device to the blockchain. This allows for geographic mobility of these devices.

4.5.3 LoRa Connected Devices. Systems using LoRa for connectivity do not possess the bandwidth necessary to run Go Ethereum locally. These devices may also lack other resources to run Go Ethereum. The Adafruit Feather M0 meets none of these requirements, as is true for a large portion of IoT devices; mechanisms for this category of device must be included.

Since these devices cannot run Go Ethereum, this must be done by proxy/gateway. A protocol was created to allow for a LoRa

[5]https://www.adafruit.com/product/3178

Table 1: Test Devices (all ARM CPUs)

Device	CPU	Cores	Frequency (MHz)	RAM	Network
Raspberry Pi 2B+	Cortex-A53	4	900	1 GB	WiFi and LoRa
Raspberry Pi 4B	Cortex-A72	4	1500	4 GB	WiFi
Raspberry Pi Zero W	117676JZF-S	1	1000	512 MB	WiFi
AdaFruit Feather M0	Cortex M0 ATSAMD21G18	1	1000	32 KB	LoRa module (SX127X)

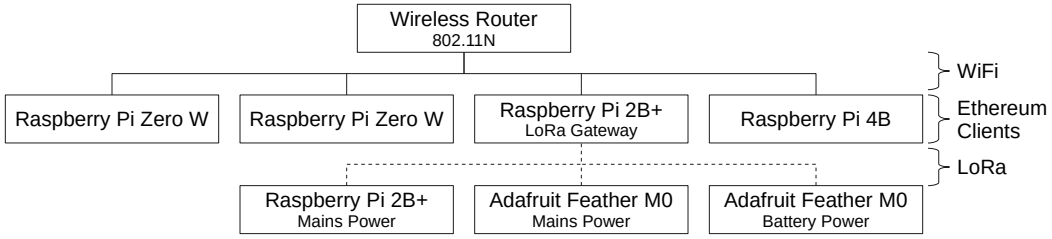


Figure 2: Configuration of test network

gateway to act upon a device's behalf, while maintaining data confidentiality and integrity. Devices communicating over LoRa must be authenticated and cannot be implicitly trusted. Our protocol detects forged/altered data and prevents eavesdropping as described in Section 4.6.

The custom LoRa protocol allows for a payload of up to 154 bytes which is three times larger than LoRaWAN and by extension, The Things Network. The packet structure is as follows:

- Source and destination device ID (4 bytes each),
- Message ID (1 byte),
- Packet fragment number (1 byte),
- Flags (1 byte),
- Reserved (1 byte),
- Data length (1 byte),
- Message signature (64 bytes),
- Cryptographic nonce (24 bytes), and
- Encrypted data (max 154 bytes).

The "message ID" and "packet fragment number" are presently unused, but are left in for future versions, to enable fragmented messages, similar to packet fragmentation in IPv4.

When a LoRa device boots, it pre-calculates the shared key used to encrypt data for its data receiver. This key is stored to avoid having to recompute it, wasting processor cycles and power. Once data is ready to be sent, it is encrypted (Section 4.6) and encapsulated in a packet that is then digitally signed and transmitted.

4.6 Cryptography

The libSodium[6] library provides the desired algorithms and supported all but one of the platforms being used for testing. Minor changes[7] were required to port the library to the ARM Cortex M0.

Since embedded LoRa devices do not run a local Go Ethereum client, an external cryptography library will be used for digital signatures in addition to key generation, key exchange, and symmetric encryption. Digital signatures will utilize the Edwards-Curve Digital Signature Algorithm (ECDSA) using *edwards25519* parameters. This creates a 512-bit signature that the LoRa gateway can verify to ensure the authenticity of the sender. Should the message have been altered or corrupted after it is signed, it is discarded. libSodium's *crypto_sign_init(), crypto_sign_update(), crypto_sign_final_create()* are used to create a signature and *crypto_sign_final_verify()* to verify a signature.[8]

Before data can be encrypted, the shared key must be computed between the device and its data receiver using Elliptic Curve Diffie-Hellman key exchange (ECDH). Once the shared key has been determined, it is used with the symmetric XChaCha20 stream cipher to encrypt the data being transmitted. A 192-bit, randomly generated nonce is used during encryption and must also be transmitted and stored on the blockchain. This nonce itself does not need to be kept secret, but is required for decryption.

Data is encrypted on an end-to-end basis. Before a LoRa device transmits data to a gateway, it is both encrypted and signed. Upon receipt, the gateway verifies the signature. The gateway does not decrypt the data, as it does possess the necessary key to do so, unless it is designated as the data receiver for the originating device. In the case of a device running its own instance of Geth, data is pushed to the blockchain in encrypted format and the identity of the sender is verified in the smart contract. Data remains in this encrypted format while on the blockchain. A data receiver may choose to subscribe to changes to new data on the blockchain for which they are capable of decrypting. These notifications are implemented through Ethereum's event logs and Go Ethereum's *eth_subscribe* JSON API calls.

The public cryptographic keys of all devices are stored on the blockchain and can be trusted as authentic. When a signed LoRa

[6]https://libsodium.org

[7]Limited to removal of function pointers which permitted different functions to be used. No alterations were made to functions which impact the integrity/security of the cipher.

[8]https://doc.libsodium.org/public-key_cryptography/public-key_signatures

message must be verified, the gateway retrieves the devices public signature from the blockchain to perform the verification.

When a data receiver wishes to read encrypted data on the blockchain, it obtains the public key of the device it is reading data from, as well as the ciphertext, and nonce. The data receiver then calculates the shared key using libSodium's key exchange function *crypto_kx_server_session_keys()* to perform ECDH key exchange. This shared key is then used to decrypt the data.

5 RESULTS

In our experiments, devices pushed data to the blockchain every 6 seconds. This interval was selected so latency tests would not be synchronized with the block time (5 seconds) and consequently skew measurements. Data generated by each type of device consists of the following:

- Adafruit Feather M0 (LoRa): power source voltage and uptime,
- RPi 2B+ (LoRa): `uptime`,
- RPi Zero W: `uname -a`, `nproc`, `uptime`, and `free -h`, and
- RPi 4B: `uname -a`, `nproc`, `uptime`, and `free -h`.

LoRa device transmissions have fewer bytes due to the limitations of packet payload size, and for the Feather M0's, the lack of a general-purpose operating system. Since the Raspberry Pi Zero Ws are the most resource constrained devices that run their own Go Ethereum client, data was collected with Geth running in full sync and also in light sync mode. Fast sync was not used as it only affects the speed of joining a blockchain and not normal operation.

While attempting to run Geth in light sync mode on the Raspberry Pi Zero Ws, issues with memory usage arose. An initial *--cache* value of 400 (MB) was used. This resulted in the system over-using the swap space. The cache value was decreased to 128, but did not alleviate the issue entirely. To further adjust for these memory demands, the memory reserved for the GPU was decreased to 8 MB, all non-essential system services were disabled, and the system "swappiness" value was set to 1. These changes resulted in stable operation on the Raspberry Pi Zero Ws.

The LoRa gateway/proxy (Raspberry Pi 2B+) did not experience any memory-related issues, as with the Raspberry Pi Zero Ws. With 1GB of memory, the gateway is the second most memory constrained device. This device was run with a *--cache* value of 512 (MB) and had a total of 969 MB of usable memory (the difference is reserved for the GPU).

Latency was measured on The Things Network over LoRaWAN using a Raspberry Pi 2B+ as a single channel gateway and the other Raspberry Pi 2B+ as a LoRaWAN end node. The end node also ran an MQTT client which subscribed to new data on this The Things Network application. The node logged the UNIX epoch time in milliseconds upon transmission and data notification. A summary of the measured latencies is found in Table 2. The network RTT from the LAN to The Things Network router was measured using the *tcptraceroute* utility, as the router does not reply to pings. An average network latency of 63.9 ms was measured to *us-west.thethings.network:1883*.

Figures 3 and 4 show latency under the experimental scenarios. These measurements were made by transmitting text data. With both The Things Network and the blockchain-based system, messages are subscribed to, and the time between transmission and notification are recorded. The measurements on The Things Network showed a mean latency of 353 milliseconds. Measurements on the Raspberry Pi 4B, and Raspberry Pi Zero Ws with 1 light serve node and 2 light serve nodes showed a mean latency of 3949 ms, 19488 ms, and 18934 ms respectively.

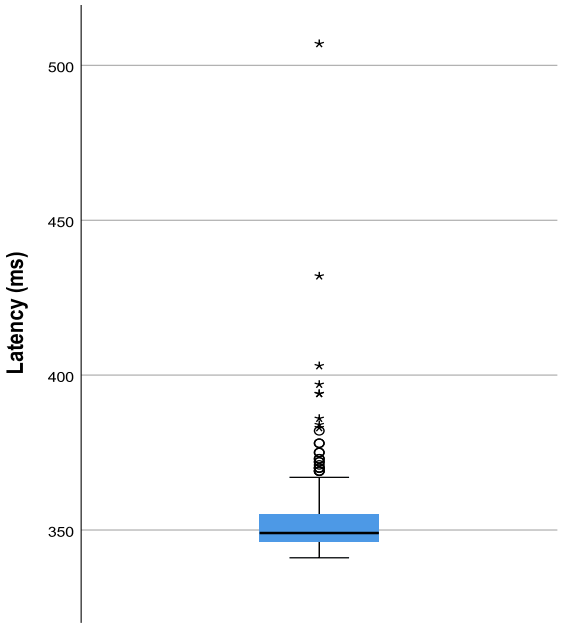


Figure 3: Measured Data Latency: The Things Network

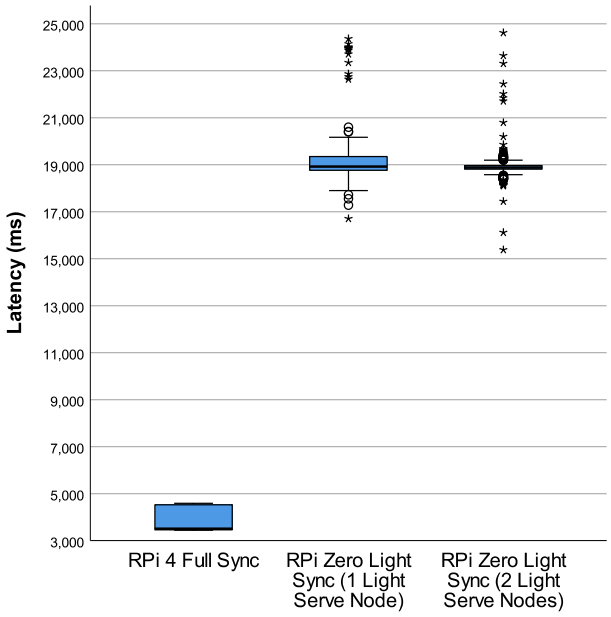


Figure 4: Measured Data Latency: RPi 4B Full Sync and RPi Zero W Light Sync

Latency measurements made on the Raspberry Pi 4B in full sync mode showed a larger standard deviation (529.2) than the measurements from The Things Network (13.8), however, it showed no outlier values. This is likely due to the lack of a dependence on an external system to process the request and transport over the public internet. The Raspberry Pi Zero Ws also resulted in many larger outliers in both tests. The large variance in latency observed on the Raspberry Pi Zero Ws may be an issue for some IoT applications. Cloud-based services such as The Things Network offer considerably lower latencies than the blockchain-based system in all configurations examined.

From the latency measurements, it is clear that a blockchain-based system introduces a considerable latency. This latency is exacerbated when using Go Ethereum's light sync mode to reduce processor and memory requirements. Since IoT tends to run on abundant, inexpensive hardware, it is clear that more IoT devices would be likely to run an Ethereum client in a light sync mode over full sync mode. This would restrict these devices to applications where latencies of approximately 19.5 seconds is permissible. This cannot rival cloud-based services, such as The Things Network, for fast delivery of data. Even on devices with sufficient resources to use full sync mode, the latencies will clearly exceed those of cloud-based services.

The second latency test conducted on the Raspberry Pi Zero Ws had an additional Raspberry Pi 4B on the test network (not shown on Figure 2). Both Raspberry Pi 4Bs were run in full sync mode and served the Raspberry Pi Zero Ws in light sync mode. The measurements of this experiment are shown in the right-most boxplot of Figure 4. The additional light serve node did not reduce the average latency, but did reduce the variance in latency. Both of these scenarios did, however, have a substantial number of outliers.

To assess the demand on the compute resources of the devices, the load averages were sampled over time. Data from the first 16 minutes of each experiment was discarded to eliminate any startup effects. The Raspberry Pi Zero W load averages were sampled at intervals reflecting the observed latency whilst the other devices were sampled every 6 seconds.

While running the Raspberry Pi 4B in full sync mode (Table 3) and serving light clients, the load averages were well below the systems total load capacity of 4.0. The demand on this client will increase as additional light clients would be added to the blockchain. It has the capacity to service more light sync clients, but how many cannot be concluded without larger scale evaluation.

The Raspberry Pi 2B+ operating as a LoRa gateway/proxy was run as a a full sync node with signing autority for the Proof-of-Authority consesus scheme. The load averages measured on this device during operation are found in Table 4. This device was configured to not serve light sync clients at any point. Despite the fact that this device has less compute power than the Raspberry Pi 4B, it experienced less load on its processor due to the lack of serving light clients.

In full sync mode, the Raspberry Pi Zero Ws (Table 5) were observed to have load averages well above their capacity of 1.0. On a single core device this indicates the system is overloaded. The Raspberry Pi Zero W is therefore not capable of running Go Ethereum as a full sync node. When tested as a light sync node (Table 6), on average it did not overload the processors, although the maximum load averages do indicate periods in which they were overloaded.

Go Ethereum exhibited periodic bursts of heavier processor utilization (Figure 5) on the Raspberry Pi 4B in full sync mode (Table 3). This behaviour was not present on the Raspberry Pi Zero W in full sync mode, which may be attributed to the system being overloaded.

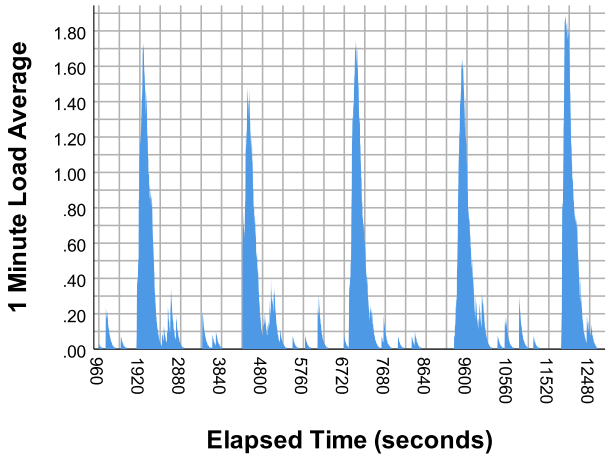


Figure 5: Load Average: RPi 4B Full Sync Node, Light Serve

6 DISCUSSION

For IoT devices which provide real-time data, such as a security camera, this system may not offer data storage, but other services can be rendered by the blockchain. These services include a cryptographic key management system, a registry of the devices IP address, and a remote device administration platform.

The use of a private network allowed for greater control of blockchain parameters. These included a degree of control over latency, avoiding need for transaction fees, and the use of Proof-of-Authority consensus over Proof-of-Work. This also allowed for the inclusion of some IoT devices in the security of the blockchain itself through the voting process used in Proof-of-Authority. This could be further leveraged in scaled up networks through the use of multiple segregated blockchains to limit blockchain growth rate, reduce the network throughput on each device, and increase security through isolation.

Although the Raspberry Pi Zero Ws did manage to run the blockchain IoT security framework as a light client, it used the vast majority of their resources and may border on being impractical. Due to the broad range of hardware used in IoT systems, an all-encompassing system will require more modes of operation to best tailor the system to the needs of each device. Future designs for a blockchain-based IoT security framework could account for such devices (IPv4/IPv6 connected, but limited compute or memory resources), by extending the concept of the proxy used for LoRa devices to devices over IP networks. This would not only serve to shift a considerable amount of the demands on the processor and memory to a more capable device, but also reduce the latency closer to those measured on the Raspberry Pi 4B.

Table 2: Measured Data Latencies (ms)

System	N	Min	Max	Mean	Std. Deviation
The Things Network	310	341	507	353	14
RPi 4B Full Sync	249	3,441	4,586	3,949	529
RPi Zero W Light Sync	104	16,706	24,364	19,488	1,689

Table 3: RPi 4B Load Avg as Full Sync, Light Serve

	1 minutes	5 minutes	15 minutes
Mean	0.22	0.21	0.17
Minimum	0.00	0.00	0.00
Maximum	1.89	0.91	0.45

N = 1950 (after discarding)

Table 4: RPi 2B+ Load Avg as Full Sync, LoRa Gateway

	1 minutes	5 minutes	15 minutes
Mean	0.10	0.10	0.13
Minimum	0.00	0.06	0.08
Maximum	0.34	0.18	0.18

N = 193 (after discarding)

Table 5: RPi Zero W x2 Load Avg as Full Sync

	1 minutes	5 minutes	15 minutes
Mean	1.44	1.50	1.48
Minimum	0.43	1.00	1.04
Maximum	3.16	2.22	1.87

N = 2881 (after discarding)

Table 6: RPi Zero W x2 Load Avg as Light Sync

	1 minutes	5 minutes	15 minutes
Mean	0.47	0.48	0.46
Minimum	0.00	0.21	0.21
Maximum	1.41	0.92	0.72

N = 1462 (after discarding)

Data in this system remains encrypted end-to-end. While the data on the blockchain itself must be considered publicly visible, data exists on the blockchain in encrypted form. The public keys of devices also exist on the blockchain, and it is possible to publicly determine the public key of the recipient device. Since the blockchain is also an immutable ledger, this history will persist on the blockchain. Although the cryptographic systems used are not known to be insecure, it should be noted that if any of these ciphers are broken, the history on the ledger will be exposed.

The integrity of data is maintained in two ways. Data that is already on the blockchain remains immutable by virtue of the design of the blockchain system itself. Secondly, data being sent to the blockchain is validated for integrity either by the LoRa gateway (which is trusted by the smart contract), or if the device is running its own Go Ethereum client, the blockchain network will validate the signature of the transaction sent to the contract. While the gateways are not insecure, gaining control of a single gateway would permit an attacker to exploit the smart contracts implicit trust of the gateway. This would allow the attacker to submit data to the blockchain on behalf of any LoRa device, but not devices which do not use this proxy system. This level of exposure is due to the design choice to allow any LoRa device to operate with any LoRa gateway on the system to allow for geographic mobility of devices. The alternative of this being each LoRa device may only communicate with a specific gateway rendering nodes less mobile.

The availability of data that already exists on the blockchain is extremely considerable, due to the distributed nature of blockchain technologies. This makes the existing data almost impervious to Denial of Service attacks. Individual nodes may be targeted and temporarily disabled, but the greater system itself would continue to function and previously received data from the victim device would continue to be available.

The proposed system is vulnerable to jamming attacks by virtue of the LoRa LPWAN technology itself. As with any wireless communications technology, a transmission can be disabled or interrupted by overwheming the channel(s) with noise. LoRa is particulairly succeptible to this due to its low transmission power and use of license-free frequencies. While nothing can be done to eliminate this vulnerability, monitoring of the most recent data timestamps could provide an indication of potential communications issues.

Since the only devices that presently do not run their own Ethereum client are devices communicating exclusively over LoRa, these devices are the only class of device which cannot be implicitly trusted. To address this, the use of digital signatures was used to authenticate the sender. This addresses the potential issue of data forgery, but the issue of replay attacks remains. While an attacker cannot view the message due to it being encrypted, nor can they alter the message due to the signatures, replaying the exact same message will appear to LoRa gateways to be authentic. This can be addressed by introducing a mechanism at the gateway which examines a message identifier which must be incremented by the sender.

While the system supports the changing of both encryption and signature keys, issues exist regarding the communication of new keys between LoRa-only devices and gateways. Since LoRa is an unreliable communication network, this creates potential inefficiencies/overhead when synchronizing keys between LoRa-only devices and gateways. Should either class of device change one of its keys, it would then need to inform the devices it communicates

with over LoRa of its new public key. Should this message not be properly received, this would lead to the public keys being out of sync and breaking communications between the pair of devices. A reliable protocol for the exchange of keys is therefore required for this system to be practical in real-world applications, as changing of cryptographic keys is paramount to the ongoing confidentiality and integrity of data.

7 CONCLUSIONS AND FUTURE WORK

The Internet of Things is a rapidly growing industry that can solve many novel problems and improve the efficiency of others, but it also exposes much risk if it is not properly secured. The dangers of vulnerable IoT devices is not merely hypothetical; security flaws have already been found which endangered lives [28]. Blockchain technology can provide the backbone required to create a strong, unified security framework for a network of heterogeneous IoT systems. Utilizing a blockchain-based solution introduced longer latencies, but did successfully consolidate a broad range of hardware into one security framework. Through additional modes of operation, such as a proxy over IP, the maximum latencies of the system could be reduced. In addition, our system delivers a superior resistance to the growing threat of Denial of Service attacks, by virtue of the distributed nature of blockchain systems. The system presented caters well to wireless sensor networks and other delay tolerant applications, and with further development can be significantly improved.

Our system as described and implemented is useful for a subset of IoT applications and could offer other functionality in a Denial of Service resistant manner. As part of future work, the system will be extended to more classes of devices to provide a comprehensive framework. Ways of improving the usage of system resources will be further explored and compared to lower the threshold of capabilities that are required of devices in order to participate in the Proof-of-Authority voting process. The use of distributed storage systems, such as SWARM and IPFS will be explored for the use of data storage, opposed to the smart contract state. Metrics will be gathered with these technologies and compared to determine the most feasible solution for resource constrained systems.

Many additional mechanisms could be added to further harden the security of this system and expand its utility. Security can be improved by addressing the issues of LoRa replay attacks and changing cryptographic keys described in Section 6. Additional features could include a registry of IP addresses, and a secure remote device administration platform. It will also be necessary to evaluate these systems at scale and examine the tails of response time distributions in more detail, because of the need for IoT security to be deployed in real-time.

REFERENCES

[1] D. R. Aleko and S. Djahel. 2019. An IoT Enabled Traffic Light Controllers Synchronization Method for Road Traffic Congestion Mitigation. In *2019 International Smart Cities Conference (ISC2)*. IEEE, Casablanca, Morocco, 709–715.

[2] Pelin Angin, Melih Burak Mert, Okan Mete, Azer Ramazanli, Kaan Sarica, and Bora Gungoren. 2018. A blockchain-based decentralized security architecture for IoT. In *International Conference on Internet of Things*. Springer, Seattle, WA, 3–18.

[3] A.M. Antonopoulos. 2014. *Mastering Bitcoin: Unlocking Digital Cryptocurrencies*. O'Reilly Media, Sebastapol, CA.

[4] A. Augustin, J. Yi, T. Clausen, and W.Wm. Townsley. 2016. A Study of LoRa: Long Range & Low Power Networks for the Internet of Things. *Sensors* 16, 9 (2016), 1–18. Article 1466.

[5] J. P. Aumasson, S. Fischer, S. Khazaei, W. Meier, and C. Rechberger. 2008. New features of Latin dances: Analysis of Salsa, ChaCha, and Rumba. In *Fast Software Encryption*, Vol. 5086 LNCS. Springer, Lausanne, Switzerland, 470–488.

[6] Daniel J. Bernstein. 2004. Cache-timing attacks on AES.

[7] Daniel J. Bernstein. 2008. ChaCha, a variant of Salsa20. https://cr.yp.to/chacha/chacha-20080128.pdf

[8] Daniel J. Bernstein. 2008. Extending the Salsa20 nonce. https://cr.yp.to/snuffle/xsalsa-20110204.pdf

[9] Daniel J. Bernstein. 2008. *The Salsa20 Family of Stream Ciphers*. Springer Berlin Heidelberg, Berlin, Heidelberg, 84–97.

[10] K. Biswas and V. Muthukkumarasamy. 2016. Securing Smart Cities Using Blockchain Technology. In *2016 18th International Conference on High Performance Computing and Communications; 14th International Conference on Smart City; 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, Sydney, Australia, 1392–1393.

[11] Vitalik Buterin. 2014. A next-generation smart contract and decentralized application platform. (2014), 36 pages. White Paper.

[12] Cisco Systems Inc. 2020. Cisco Annual Internet Report (2018-2023) White Paper. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[13] Stefano De Angelis, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri, and Vladimiro Sassone. 2018. PBFT vs Proof-of-Authority: Applying the CAP Theorem to Permissioned Blockchain. In *Italian Conference on Cyber Security*. CINI, Milan, Italy, 1–11.

[14] A. Dorri, S. S. Kanhere, and R. Jurdak. 2017. Towards an Optimized BlockChain for IoT. In *2nd International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE/ACM, Pittsburgh, PA, 173–178.

[15] Cynthia Dwork and Moni Naor. 1993. Pricing via Processing or Combatting Junk Mail. In *Advances in Cryptology — CRYPTO' 92*. Springer Berlin Heidelberg, Berlin, Heidelberg, 139–147.

[16] Seth Gilbert and Nancy Lynch. 2002. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services. *SIGACT News* 33, 2 (June 2002), 51–59.

[17] J. Hoffstein, J. Pipher, and J.H. Silverman. 2014. *An Introduction to Mathematical Cryptography*. Springer New York, New York, NY.

[18] Sunghyuck Hong. 2017. Secure and light IoT protocol (SLIP) for anti-hacking. *Journal of Computer Virology and Hacking Techniques* 13, 4 (01 Nov. 2017), 241–247.

[19] Seyoung Huh, Sangrae Cho, and Soohyung Kim. 2017. Managing IoT devices using blockchain platform. In *2017 19th International Conference on Advanced Communication Technology (ICACT)*. IEEE, Phoenix Park, PyeongChang, South Korea, 464–467.

[20] K. Lauter. 2004. The advantages of elliptic curve cryptography for wireless security. *IEEE Wireless Communications* 11, 1 (2004), 62–67.

[21] Jun Lin, Zhiqi Shen, and Chunyan Miao. 2017. Using Blockchain Technology to Build Trust in Sharing LoRaWAN IoT. In *Proceedings of the 2nd International Conference on Crowd Science and Engineering*. Association for Computing Machinery, Beijing, China, 38–43.

[22] Kerry Maletsky. 2015. RSA vs ECC comparison for embedded systems. (2015), 4 pages.

[23] Daniel Minoli and Benedict Occhiogrosso. 2018. Blockchain mechanisms for IoT security. *Internet of Things* 1-2 (2018), 1–13.

[24] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System.

[25] National Institute of Standards and Technology. 2001. *FIPS PUB 197: Announcing the ADVANCED ENCRYPTION STANDARD (AES)*. National Institute of Standards and Technology, Gaithersburg, MD.

[26] K. R. Özyilmaz and A. Yurdakul. 2019. Designing a Blockchain-Based IoT With Ethereum, Swarm, and LoRa: The Software Solution to Create High Availability With Minimal Security Risks. *IEEE Consumer Electronics Magazine* 8, 2 (March 2019), 28–34.

[27] Phillip Sparks. 2017. White Paper: The route to a trillion devices. https://community.arm.com/iot/b/internet-of-things/posts/white-paper-the-route-to-a-trillion-devices

[28] US Food and Drug Administration. 2017. Cybersecurity vulnerabilities identified in St. Jude Medical's implantable cardiac devices and Merlin@ home transmitter: FDA safety communication. https://www.fda.gov/MedicalDevices/Safety/AlertsandNotices/ucm535843.htm

[29] Vivek Vishnumurthy, Sangeeth Chandrakumar, and Emin Gun Sirer. 2003. Karma: A secure economic framework for peer-to-peer resource sharing. In *Workshop on Economics of Peer-to-peer Systems*. Berkeley School of Information, Berkeley, CA, 1–6.

[30] S. Wang, L. Ouyang, Y. Yuan, X. Ni, X. Han, and F. Wang. 2019. Blockchain-Enabled Smart Contracts: Architecture, Applications, and Future Trends. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 49, 11 (2019), 2266–2277.

[31] Gavin Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. (Oct. 2014), 32 pages. https://ethereum.github.io/yellowpaper/paper.pdf.

A Survey of Security Vulnerabilities in Ethereum Smart Contracts

Noama Fatima Samreen, Manar H. Alalfi
noama.samreen,manar.alalfi@ryerson.ca
Department of Computer Science, Ryerson University
Toronto, ON, Canada

ABSTRACT

Ethereum Smart Contracts based on Blockchain Technology (BT) enables monetary transactions among peers on a blockchain network independent of a central authorizing agency. Ethereum Smart Contracts are programs that are deployed as decentralized applications, having the building blocks of the blockchain consensus protocol. This enables consumers to make agreements in a transparent and conflict-free environment. However, there exists some security vulnerabilities within these smart contracts that are a potential threat to the applications and their consumers and have shown in the past to cause huge financial losses. In this study, we review the existing literature and broadly classify the BT applications. As Ethereum smart contracts find their application mostly in e-commerce applications, we believe these are more commonly vulnerable to attacks. In these smart contracts, we mainly focus on identifying vulnerabilities that programmers and users of smart contracts must avoid. This paper aims at explaining eight vulnerabilities that are specific to the application level of BT by analyzing the past exploitation case scenarios of these security vulnerabilities. We also review some of the available tools and applications that detect these vulnerabilities in terms of their approach and effectiveness. We also investigated the availability of detection tools for identifying these security vulnerabilities and lack thereof to identify some of them.

CCS CONCEPTS

• **Security and privacy → Software and application security**; *Cryptography*; **Vulnerability management**.

KEYWORDS

blockchain, ethereum, smart contracts

1 INTRODUCTION

Attributing to the wide range applicability of Blockchain Techhnology(BT), it has been finding popularity in many domains. Bitcoin was the first version of cryptocurrency applied using BT [6] and has since been used in many other applications such as e-commerce, trade and commerce, production and manufacturing, banking, and gaming. BT uses a peer-to-peer (peers are known as miners in BT) framework which is a more decentralized approach to storing transactions and data registers. As there is no single point of failure or a third-party centralized control of transactions, BT has been standing out from cryptocurrency-based other technologies. It uses a chain of blocks in which each block is locked cryptographically using the hash of the previous block it is linked to, which creates an immutable database of all transactions stored as a digital ledger, and it cannot be changed without affecting all the blocks linked together in the chain [14]. However, recent research to identify the existence of security vulnerabilities in Ethereum Smart Contracts have shown that many applications have been exposed to attacks because of vulnerabilities found in application level of Ethereum Smart Contracts [18].

Ethereum Smart Contracts are typically written in Solidity Language and this paper presents a classification for vulnerabilities in these Solidity-based Ethereum Smart Contracts. Our methodology can be characterized as targeting security from three perspectives: vulnerabilities, exploitation case studies, and preventive techniques. For each vulnerability, we discuss, among other things, its research statistics (i.e., detection tools available to identify the vulnerability, analysis method most preferred by researchers to identify the vulnerability). For each exploitation case, we discuss, among other things, the vulnerability exploited, tactic, and financial losses incurred in terms of ether. For each preventive technique, we discuss its mechanism and the vulnerability it aims to protect from exploitation. We aim to provide future research directions by providing statistics of research done on each vulnerability to address the severity of a vulnerability and the requirement of further work on open problems.

We identify eight application level security vulnerabilities and classify them according to the NIST's Bugs Framework [11]. To do so, we collected information from the NIST's website to match the descriptions of existing bug classes with the Ethereum Smart Contracts security vulnerabilities discussed. As highlighted in Table[1], most of these vulnerabilities are classified as *Not Available (NA)* concerning NIST-BF classification. This is because most of these vulnerabilities are specific to developmental methodology of a Solidity based Ethereum Smart Contract and BT and do not match an exact *Bugs Framework (BF) category* outlined by NIST. This classification concerning NIST categorization of software bugs will aim at a better understanding of the nature of each of the Ethereum Smart Contract's application level security vulnerabilities.

2 BACKGROUND

2.1 Blockchain Technology (BT)

Based on the industry they target, BT applications can be broadly classified into three categories: Public Blockchain 1.0, Public Blockchain 2.0 and Private Blockchain 3.0.

CASCON'20, November 10 - 13 2020, Toronto, Canada

- Public Blockchain 1.0 is used predominantly in the finance industry for digital payments and currency transfers.
- Public Blockchain 2.0 is used in the e-commerce industry and it includes Ethereum Smart Contracts, such applications process financial contracts intelligently and provide a foundation for digital asset ownership. As Ethereum smart contracts find their application mostly in e-commerce applications, we believe these are more commonly vulnerable to attacks caused by simple coding errors.
  In this paper, we mainly focus on identifying vulnerabilities in these smart contracts that programmers and users of smart contracts must avoid. Case studies outline that Ethereum smart contracts are vulnerable to simple coding errors such as re-entrance (recursive calling of functions: A calling B while B is calling A), wrong constructor name, typecasts, unintended function exposure, stack overflow, etc. as this generation of BT is public and distributed. These coding errors can aid an attacker in manipulating transactions to successfully launch an intrusion attack by techniques such as publishing malicious contract on the BT network to receive more transactions than it sends out thereby collecting Ether (digital asset bearer or a token by which applications are processed on an Ethereum network) many times over within a single transaction, exploiting the visibility modifiers to misuse function delegation etc.
- Private Blockchain 3.0 is used in the government, health, science industries and are hence considered private.

2.2 NIST Bugs Framework

In this paper, we leverage the National Institute of Standards and Technologies Bugs (NIST'S) Framework to provide another classification of these vulnerabilities to provide a basis of comparison with other common software bugs reported by NIST[11]. This categorization by NIST is an effort to accurately describe a commonly occurring software bug and it incorporates definitions and attributes of each software bug class along with their causes and consequences. This classification provides researchers and developers to match vulnerabilities in new technologies to the previously researched bugs and adopt appropriate preventive techniques.

2.3 Analysis Methodologies

Literature review of the selected Smart Contracts vulnerabilities detection tools/frameworks for this paper shows that these tools/frameworks adopted one of the following methods of smart contract analysis,

2.3.1 *Static Analysis.*

(1) Symbolic Execution - The execution of code using symbols rather than real values for the variables. This analysis results in algebraic terms of operating these symbols and the conditional statements in the program result in propositional formulas that direct the flow of execution. The feasibility of a path of flow is determining if the conjunction of formulas on the path is satisfied.
(2) CFG (Control Flow Graph) Construction - The representation of a program in a directed graph. An edge of this graph represents the flow of execution with conditions mentioned on the edge as a label.
(3) Pattern Recognition - The classification of a program's basics units or data depending on the prior knowledge or statistical information gained from patterns.
(4) Rule-based Analysis - The checking/analyzing of code against a rule-based specification of its behaviour. The rule-based specification describes scenarios during execution and enforces constraints on the sequence of operations and data inputs.
(5) De-compilation Analysis - The representation of Ethereum Virtual Machine (EVM) bytecode with a higher abstraction level to improve the parsing of the code and data flow analysis.

2.3.2 *Dynamic Analysis.*

(1) Execution Trace at Run time - Tracing the sequence of instructions that are executed during a particular run of the code
(2) Fuzzing Input Generation - Fuzzing is an automated analysis method that tests program execution by providing structured data as inputs to a computer program. The program under analysis is then monitored for unexpected behaviour such as unusual code path, or crashes.

3 RELATED WORK

We conducted a literature review and we inferred that there is a lack of a comprehensive study of these security vulnerabilities in Solidity based Ethereum Smart Contracts. A survey of vulnerabilities by L. Luu et al. [26] focuses only on the security vulnerabilities that exist in the Ethereum Smart Contracts without providing a detailed study of their exploitation cases or preventive techniques. Another study by N. Atzei et al. [15] discusses security vulnerabilities and some of their real world attack scenarios in general without providing a mapping between the vulnerabilities and the attacks. Then there is a survey of the current research available in the field of ethereum smart contract by Alharby and Moorsel. [13] that characterizes the surveys available depending on their nature of survey as the surveys identifying codifying issues, security issues, privacy issues, and performance issues. One of the most recent surveys in this area is by Praitheeshan et al. [27] which lists 13 security vulnerabilities (involving application level, BT level and Ethereum Virtual Machine-EVM level security vulnerabilities) in Ethereum Smart Contracts but maps them to only four major attacks scenarios. Therefore, it fails to give an example pattern or a real world exploitation case scenario for each of the security vulnerability discussed.

On the other hand, there has been extensive development of automated tools in the industry to detect these vulnerabilities. These developers utilize the research available in this area to produce highly efficient state-of-the-art tools to detect these vulnerabilities. Some of the research work in developing automated tools for detecting these vulnerabilities focuses on detecting only a specific type of vulnerability without analyzing the vulnerability in detail with its exploitation cases and preventive techniques. [25],[4], [23] One such recent tool is ETHPLOIT by Q. Zhang et al. [33] to automatically detect vulnerabilities that have been exploited in Ethereum smart contracts. This tool adopts light-weight techniques to answer

the problems of previously developed tools. These problems consisted of unsolvable constraints and Blockchain effects. It is claimed by Q. Zhang et al. [33] that this tool achieves precise and efficient smart contract analysis and successfully detects more exploits than previous exploit generation tools.

However, a new research paper SMARTSHIELD by Y. Zhang et al. [34] utilizes EVM bytecode analysis and provides an automatic correction mechanism to avoid vulnerable patterns in Ethereum smart contracts. It does this rectification by extracting EVM bytecode level semantic data to transform the vulnerable smart contracts into secure ones. And then, there are also some surveys like the one by Angelo and Salzer [17] that compare only the detection tools available in the market without actually discussing the characteristics of the vulnerabilities these tools excel or fail at detecting.

One of the most recent surveys published in Feb. 2020 by Durieux et al. covers three research questions regarding these automated tools available in the market. This research work by Durieux et al. questions the effectiveness of these tools in terms of precision in detecting these vulnerabilities in Ethereum Smart Contracts. Durieux et al. next articulates about the quantitative analysis of the vulnerabilities present in the Ethereum blockchain main network.

Different from the existing surveys discussed above, our paper aims to particularly analyse the vulnerability pattern, their real world exploitation cases, their preventive techniques and the detection methods adopted by currently available tools for analyzing ethereum smart contracts. We highlight the need for a comprehensive study on the security analysis methods of vulnerable smart contracts on the Ethereum platform. Furthermore, this paper is different from the existing ethereum smart contracts surveys because we investigate each security vulnerability in detail along with the available detection tools to analyse the security vulnerability and the methodology adopted by these detection tools to identify this vulnerability in ethereum smart contracts.

4 VULNERABILITIES

This paper combines the identification and analysis of eight of the application level security vulnerabilities along with their real-world exploitation cases to better capture the vulnerabilities scenarios in Solidity-based Ethereum Smart Contracts. Table[1] shows the vulnerable contracts used in exploitation cases, their preventive techniques respectively along with the matching NIST bug class for each vulnerability.

4.1 Reentrancy

A reentrancy attack can drain a smart contract of its ether, can aid an intrusion into the contract code. When an external call function to another untrustworthy contract is made and an attacker gains control of this untrustworthy contract, they can make a recursive call back to the original function, unexpectedly repeating transactions that would have otherwise not run, and eventually consume all the gas.

Exploitation Case of Reentrancy Vulnerability - The DAO Attack. The Decentralized Autonomous Organization (known as the DAO) was initiated in May 2016 as a venture capital fund for the crypto and decentralized space[14]. During the creation period of the DAO, anyone could send Ether to a unique wallet address in exchange for DAO tokens. Anyone with DAO tokens could vote on the pitch and receive rewards in return if the projects turned a profit. However, on June 17, 2016, a hacker was able to attack this Smart contract by exploiting a vulnerability in the code that allowed him to transfer funds from the DAO. As reported by M. Saad et al.[29] approximately, 3.6 million Ether was stolen, the equivalent of USD 70M at the time. The reentrancy vulnerability exploitation in the DAO attack(as shown in Listing 1) was accomplished in four steps,

- The Attacker initiates a transaction by calling withdraw function of Victim;
- The Victim transfers the money and calls the fallback function of the Attacker;
- The fallback function recursively calls the withdraw function again, i.e., *Reentrancy*;
- Within an iteration bound, extra ether will be transferred multiple times to the Attacker.

```
1  contract Victim {
2      bool etherTransferred = false;
3      //Attacker calls the withdraw() function to initiate
           the attack
4      function withdraw(){
5      //Victim transfers ether which invokes the fallback
           function of the attacker
6      if(etherTransferred ||
7      !msg.sender.call.value(1) ()) throw;
8      etherTransferred = true;
9      }}
10 contract Attacker{
11     uint count = 0;
12     function () payable{
13     if(++count < 10) Victim (msg.sender).withdraw();
14     }}
```

Listing 1: Simplified DAO Attack - Reentrancy Vulnerability

Preventive Techniques. Reentrancy vulnerability can be prevented by ensuring that state changing logic is committed before ether is sent out of the contract through an external call. It is also a good coding practice to put any logic that performs external calls to unknown addresses at the last operation in a program's execution. This is known as the checks-effects-interactions pattern. Another technique is to use a mutex by adding a state variable which locks the contract during code execution, thus preventing re-entrant function calls.

4.2 Out-of-Gas exception

The primitive function *send* may cause an unexpected out-of-gas exception when transferring ether among contracts. There is a prefixed units of gas available to allow execution of a limited set of bytecode instructions and the call function will end up in an out-of-gas exception if not enough gas units are available.

Exploitation Case of Out-of-Gas Exception Vulnerability - King Of Ether Throne Attack. The King of the Ether Throne contract ("KotET contract") is a game, where players compete to become the king by paying some ether as the claim price to the current king plus some fees to the contract owner [14]. After the contract declares a new King of the Ether Throne, the new claim price for the throne goes up by 50%. When the KotET contract sent ether to the new King aspirant, it inadvertently included 2300 gas with the payment.

Table 1: Smart contract vulnerabilities, their preventive techniques and their NIST bug classification;NIST-BF - National Institute of Standards and Technology- Bugs Framework; NA - Not Available, UCE - Unchecked Error, ARC - Arithmetic or Conversion Fault

S.no.	Contract Name	Vulnerability	Vulnerability Level	NIST-BF Class	Preventive Technique
1.	The DAO	Re-entrancy(recursive-calling vulnerability: A calling B calling A)	Security	NA	Placing external call logic as the last piece of code in a program
2.	King of the ether	Out-of-Gas Exception Handling	Functional	NA	use *transfer()* instead of *send()*
3.	Governmental (Ponzi Scheme)	Unpredictable state due to mishandled exceptions	Security/Functional	NA	Updating Solidity Language to handle exceptions in a uniform manner is required
4.	Second Parity MultiSig Wallet	Call-to-Unknown vulnerability	Security	UCE	Making stateless libraries of vulnerable contracts to avoid external state changing of the contract
5.	Reentrancy Honey Pot	Typecast vulnerability	Developmental	NA	Using *new* to create an instance of referenced contract
6.	Odd and Even Game	Weak Field Modifiers vulnerability	Developmental	NA	Using *internal* to protect information leakage
7.	Proof of Weak Hands Coin (PoWHC)	Integer Underflow/Overflow vulnerability	Arithmetic or Conversion Fault	ARC	Using mathematical libraries instead of the standard math operations (addition, subtraction and multiplication)
8.	HYIP	DoS by external call vulnerability	Unchecked Error Class	UCE	Asking recipient to *pull* funds out rather than sender using *push* to send out the funds. Removing dependence of conditional statements or iterational statements on an external call.

As this was not enough gas to successfully process the payment and declare a new king, the wallet contract failed. This failure resulted in the ether being returned to the KotET contract. The KotET continued processing, thereby making the caller King despite the compensation payment not having been sent to the previous king (see listing 2).

```
contract KoEth {
    address public king;
    uint public claimPrize= 100;
    address owner;
    function KoEth () {
        owner = msg.sender;  king = msg.sender;}
    function () payable {
        if(msg.value < claimPrize) throw;
        uint compensation = calculateCompensation();
        //"send" fails if the fallback function of
            reciever is expensive
        king.send(compensation);
        king = msg.sender;
        claimPrize = calculateNewPrice(); }
        }
```

Listing 2: Simplified Vulnerable King of Ether Attack - Out-Of-Gas Exception Vulnerability

Preventive Technique. This vulnerability can be prevented by using *transfer()* function instead of *send()* as the former will revert the local transactions if the external transaction reverts. However, if *send()* is required then the return value of this function needs to be monitored. Another technique is to adopt a withdrawal pattern, wherein, each user is required to call an isolated function that manages ether transactions and does not affect the rest of the contract execution. Therefore, making the transaction management independent of the consequences of failed *send()* transactions.

4.3 Call to the unknown

When a function invocation or an ether transfer unexpectedly invokes the fallback function of the callee/recipient. Some of the primitives of Solidity language that causes this are:

- *call* used to invoke a function or transfer ether
- *send*, used to transfer ether from the running contract to some other contract
- *delegatecall*, used to invoke a function or transfer ether in the caller environment
- direct call (see listing 3)

```
contract Alice{ function ping(uint) { returns (uint); }}
contract Bob{ function pong (Alice c) { c.ping (42); }}
```

Listing 3: Call to the unknown - Direct Call

If an invoked function's signature does not match with any existing function, then the call results in a call to recipient's fallback function.

Exploitation Case of Call-to-Unknown Vulnerability - Second Parity MultiSig Wallet Attack. On July 19, 2017, a major attack, in terms of

Ether stolen, on the Ethereum network took place. The attacker's account had drained 153,037 ETH from three high-profile multi-signature contracts used to store funds from past token sales [5]. The vulnerable MultiSig wallet was divided into two contracts(as shown in Listing 4) to reduce the size of each wallet and save gas:

- A library contract called "WalletLibrary",
- An actual "Wallet" contract

To begin with, the vulnerable contract had a simple constructor that delegates the initialization of the contract's state to *WalletLibrary*, followed by a *withdraw()* function which also delegates its execution to *WalletLibrary*. Using these two functions, the attacker initiated two transactions to each of the vulnerable contracts: the first to obtain exclusive ownership of the MultiSig, and the second to transfer all of its funds to itself. To obtain the ownership of the contract, the attacker needs to execute *Wallet.initWallet(attacker)*. This triggers the *Wallet*'s fallback function. In the *Wallet*'s fallback function then initiates the *delegatecall* in the *WalletLibrary*. When *WalletLibrary* receives the call, it finds that it's *initWallet* function matches the function selector and runs *initWallet(attacker)*. Thereby making the attacker the owner of the wallet and allowing him to be able to withdraw funds.

```
contract WalletLibrary {
    address owner;
    function initWallet(address _owner) {
        owner = _owner;
    function changeOwner(address _new_owner) external {
        if (msg.sender == owner) {
            owner = _new_owner;}}
    function () payable { // receive money}
    function withdraw(uint amount) external returns (
        bool success) {
        if (msg.sender == owner) {
            return owner.send(amount);}
        else {
            return false;}}}
```

Listing 4: Simplified Vulnerable MultiSig Wallet [5]

Preventive Technique. Solidity has provision for implementing library contracts by using the keyword library (see [9]). These library contracts are stateless and non-self-destructive. Forcing libraries to be stateless mitigates attacks whereby attackers modify the state of the library directly to affect the contracts that depend on the library's code. Therefore, when using *call*, *DelegateCall*, the call-to-the-unknown attack that may change the state of the victim contract can be prevented by building stateless libraries.

4.4 Typecasts

The fact that the Solidity compiler can detect some type errors may cause the programmers to believe that it also checks for the address of the contract being called and the interface declared by the caller function matches callee's actual interface. The execution of a contract in the presence of such type mismatch errors will not throw exceptions at run-time and the caller is unaware of the error resulting in three different cases at run-time:

- Incorrect contract address of callee function, the call returns without executing any code,
- Contact address of callee function matches with any other function's signature, then that function is executed
- Contact address of callee function does not match with any function's signature, then its fallback is executed.

Exploitation Case of Typecasts Vulnerability - Reentrancy honey-Pot Attack. Honey pot contracts are deployed on the Ethereum main network to capture Ethereum hackers who try to exploit the contracts. A small scale attack using the typecasts vulnerability was successfully launched on a honey pot developed to capture hackers trying to exploit the reentrancy vulnerabilities in smart contracts (see listing 5) In listing 5, this vulnerability can be exploited by replacing an expected contract address with a malicious address in the constructor.

```
contract Bank_Contract{
    mapping (address => uint) public balances;
    uint public MinDeposit = 1;
     function Bank_Contract(address _sender){
     //update malicious address here
     }
    function sendDeposit() public payable{
        if(msg.value >= MinDeposit){
            balances[msg.sender] += msg.value;
            }}
    function withdraw(uint _am){
        if(_am <= balances[msg.sender]){
            if(msg.sender.call.value(_am)()){
                balances[msg.sender] -= _am;}}}
    function() public payable{}}
```

Listing 5: Reentrancy Honey Pot Contract - Typecasts vulnerability

Preventive Technique. To prevent typecasting to malicious contract, the *new* keyword can be used. This way an instance of the referenced contract cannot be changed without modifying the contract as this is created at deployment time. In listing 6, the constructor could be written like:

```
constructor() {referenceContract = new reference(); }
```

Listing 6: Using new to create an instance of a contract

Another technique is to hard code any external contract addresses in the contract to avoid malicious contracts getting referenced.

4.5 Mishandled Exceptions

There are many situations when an exception can be raised in Solidity but the way these exceptions are handled is not always the same. The exception handling is based on the interaction between contracts. This makes the contracts vulnerable to attacks because programmers will be unaware of any ether that is lost if these exceptions are not handled properly and the transactions are reverted.

```
contract Alice {
function ping(uint) {
// this function throws an exception
returns (uint);}}
contract Bob {
    uint x=0;
    function pong(Alice c){ x=1; c.ping(42); x=2;} }
```

Listing 7: function *ping* of contract *Alice* throws an exception

In Listing 7, the value of variable x after the execution of contract *Bob* varies depending on the method of the function call. If the *ping* function of contract Alice is called using a direct call, then the value of x will be 0. Whereas, if the same function is called using the in-built function *call* of Solidity, then the value of x will be 2. Moreover, in case of exceptions, if no bound is specified then all the available gas is lost.

Exploitation Case of Mishandled Exceptions Vulnerability - Governmental scheme Attack. (as shown in Listing 8) In this attack, a contract that implements a flawed Ponzi scheme is targeted [14]. This attack is executed by exploiting the mishandled exceptions vulnerability in smart contracts. This scheme requires a participant to send a certain amount of ether to the scheme contract. If no one joins the scheme for 12 hours, the owner of the contract keeps his fee and transfers the remaining ether to the last participant. To join the scheme, a player must invest at least half of the claim prize. This claim prize increases upon each new investment. Anyone can invoke *resetInvestment*, which transfers the claim prize (half of the invested total) to the last participant and sends the remaining ether to the contract owner. There is a key assumption in this contract that players are either users or contracts with empty fallback, and so will not cause an out-of-gas exception during send (as shown in Listing[8]

```
contract Governmental {
    address public owner;
    address public lastInvestor;
    uint public claimPrize= 1;
    function Governmental () {
        owner = msg.sender;
       if(msg.value < 1) throw; }
    function invest (){
        if (msg.value < claimPrize/2) throw;
       lastInvestor = msg.sender;
       claimPrize += msg.value/2; }
    function () resetInvestment {
        lastInvestor.send(claimPrize);
        //contract sends the prize money to the winner
        owner.send(this.balance - 1);
        // and sends the remaining ether to the owner
        lastInvestor = 0;
        claimPrize = 1;}
        }
```

Listing 8: Simplified Governmental Attack - Mishandled Exceptions Vulnerability

Preventive Technique. One technique to avoid this vulnerability would be to use one method of external call throughout. However, this is not an ideal preventive technique as different variations of an external call can be a necessity. Therefore, this vulnerability requires an update on the Solidity Language to make the consequences of a thrown exception uniform.

4.6 Weak Field Modifiers

Fields in smart contracts can be labelled as *Public* or *Private*. However, these attributes are not enough to protect a field's value. This is because the default access modifier of afield in Solidity is *public*. Whenever a field's value is changed, this change is published on the BT chain and there is a chance that an attacker would infer the changed value through previous hashes and new transaction hash.

```
contract OddsAndEvens {
    struct Player {address addr; uint number;}
    Player[2] private players;
    uint tot = 0;
    address public owner;
    function OddsAndEvens () {
         owner = msg.sender; }
    function play (uint number){
        if (msg.value != 1 ) throw;
        player[tot] = Player (msg.sender, number);
        tot++;
        if(tot == 2) winner(); }
    function winner() private{
        uint n = players[0].number + players[1].number;
        //contract sends 1.8 ether to the winner
        players[n%2].addr.send(1.8);
        delete players;
        tot = 0;}
    function getProfit(){
        //and sends remaining ether to the contract owner
         owner.send(this. balance);}
  }
```

Listing 9: Simplified Multiplayer Games Attack - Weak Field Modifiers Vulnerability

Exploitation Case of Weak Field Modifier Vulnerability - Odd and Even Game Attack. In this attack, a contract that implements a simple "odds and evens" game between two players is exploited [14]. An attacker impersonates the second player and when the first player makes his bet, the attacker infers this by BT network transactions. After inferring the first player's bet value, the attacker adjusts his bet accordingly that would guarantee his win. (as shown in Listing [9])

Preventive Technique. To avoid this smart contract's vulnerability, use the *internal* modifier for functions instead of public.

4.7 Integer Underflow/Overflow Vulnerability

An Integer overflow/underflow occurs when an arithmetic operation is performed that requires a fixed size variable to store data that falls outside the range of the variable's data type. The EVM [7] specifies data types with fixed-size for integers. Therefore, an integer variable can be represented by only a certain range of numbers. This vulnerability may be exploited by attackers by misusing the smart contract code and create unexpected logic flows.

Exploitation Case of Integer Underflow/Overflow Vulnerability - BEC-Token Attack. On 22nd April 2018, there was an unusual token transfer in an *ERC20 Smart contract* that prompted the contract owners to analyse the related smart contract code. The analysis resulted that the transfer was initiated as an "in-the-wild" attack that exploited the arithmetic overflow vulnerability in the contract.

Preventive Technique. This vulnerability can be avoided by Using mathematical libraries instead of the standard math operations (addition, subtraction and multiplication).

4.8 DoS By An External Call Vulnerability

When the flow of control is transferred to an external contract, the execution of the caller contract can fail accidentally or deliberately,

which can cause a DoS state in the caller contract. The caller contract can be in a DoS state when a transaction is reverted due to a failure in an external call, or the callee contract deliberately causes the transaction to be reverted to disrupt the execution of the caller contract.

Exploitation case of DoS By An External Call - HYIP (High Yield Investment Program). The contract *HYIP* is yet another Ponzi scheme. This contract sends payments to lenders from funds collected via new lenders each day. The function *sendPayment()* in Listing[10] contains the DoS by an external call vulnerability. The attack proceeds as follows:

(1) The *AttackerContract* lends funds to the *HYIP* contract and throws an exception in its fallback function.
(2) When function *sendPayment()* is called to pay the lenders, the fallback function of all the lenders is invoked and and the fall back function of this *AttackerContract* throws an exception, causing a deliberate revert of the transaction and subsequently, a DoS to contract HYIP.

```
1  contract HYIP {
2     Lenders[] private lender;
3     function sendPayment() {
4          for(uint i = lender.length;    i      > 0; ) {
5              uint payment=(lenders[i].amount*/1000;
6              if(!lenders[i].addr.send(payment)) throw;
7              } }}
8  contract AttackerContract {
9     bool private attack = true;
10    function() payable {
11         if (attack) throw;
12         // callee fails the caller execution deliberately
                }
13         }}
```

Listing 10: Contract HYIP - Exploited for DoS by an External Call Vulnerability

Preventive Technique. This vulnerability exists because of inadequate exception handling around conditional and iteration statements. Placing any external calls initiated by a callee contract into a separate transaction can help reduce the damage caused by this vulnerability. Isolating statements with the following pattern can help avoid this vulnerability: • an if-statement with an external function call in the condition and a throw or a revert in the body; • a for- or an if-statement with an external function call in the condition. Also, by asking the recipient to *pull* funds out rather than sender using *push* to send out funds.

5 RESEARCH ANALYSIS AND INSIGHTS

There has been extensive research going on to identify, characterize and prevent vulnerabilities in Ethereum Smart Contracts. For this paper, we considered the following research studies,

- Luu et al.[4] developed the Oyente analyzer that performs symbolic execution on contract functions and identifies vulnerabilities based on simple patterns. According to this framework, the vulnerabilities are classified into the following groups: transaction-ordering dependent, timestamp dependence, re-entrance handling, and mishandled exceptions.
- SmartCheck [30] is a pattern-based analysis tool that uses XPath to detect if any vulnerabilities pattern exists in a Smart Contract. To do so, it transforms the Smart Contract into XML representation.
- ReGuard [25] is a combined static and dynamic analysis tool to detect reentrancy vulnerabilities in Smart-Contracts developed by Liu et al.[25]. This tool tests the Smart-Contracts by initially transforming the Smart-Contract code into C++ and then generating fuzzing inputs to recreate Blockchain transactions as possible attacks. Then, ReGuard performs vulnerability detection through dynamic analysis.
- Contract Fuzzer [23] is a tool developed by Jiang et al. that tests the Smart-Contracts for identifying vulnerabilities in them by using the fuzzing technique. To detect the vulnerabilities this tool starts with an initial analysis of the interfaces that the Smart-Contract exposes, it then randomly develops fuzzing inputs for these interfaces and observes the execution logs of the application.
- Mythril [3] is a command-line tool in Python developed by ConsenSys for analyzing smart contracts interactively. It executes EVM bytecode symbolically and represents it in the form of a CFG, with the nodes containing disassembled code and the edges being labelled by path formulas.
- MAIAN [2] is a python based tool that uses Oyente [4] for the detection of vulnerabilities that require multiple transactions. It executes EVM bytecode symbolically and checks for execution traces. To discard false positives, the contracts are dynamically analyzed by deploying them on a private blockchain and attacking them with the computed transactions.
- Securify [32] uses EVM bytecode and security properties of a smart contract as inputs. A Security property consists of compliance and violation patterns. This tool uses the decompilation analysis method and represents the code as Data Log facts. This framework infers that if a pattern is detected, then the code possesses the corresponding security vulnerability.
- Vandal [16] is a command-line tool written in Python which disassembles and decompiles EVM bytecode into an intermediate representation and constructs a CFG.
- Zeus [24] is a tool developed by IBM Research India. Similar to Securify [32], this tool takes Solidity code and policies as input. These policies are checks that specify if the code meets a safety property expressed in the policy. Zeus converts Solidity code into LLVM bitcode, which is then instrumented with assertions corresponding to the policy.
- EthIR [12] is written in Python and analyzes only particular versions of the Solidity compiler, and Go-Ethereum. This framework transforms bytecode into an intermediate representation compatible with a static analyzer built by the same developers as of EthIR. This framework extends Oyente [4]. The CFG is represented as guarded rules and this rule-based representation is then supplied as an input to the general purpose static analyzer.

Figure[2] shows the statistics surrounding the research of individual vulnerabilities. It is evident from Figure[2] that the reentrancy vulnerability has been talked about the most and there has

Table 2: Classification of Framework/Detection Tools Available for Vulnerabilities; ✗: None Available

(a) Classification of Framework/Detection Tools Available for Vulnerabilities

S.no.	Vulnerability	Static Analysis					Dynamic Analysis		Rectification provided at Level		
		Symbolic. Execution	CFG Construction	Pattern Recognition	Rule-Based Analysis	Decompilation Analysis	Execution Trace at Runtime	Fuzzing Transactions	Solidity Level	Bytecode Level	Blockchain Level
1.	Re-entrancy	Oyente [4], Mythril [3], Securify [32], SmartCheck [30]	Oyente [4], ContractFuzzer [23], SmartCheck [30], SmartShield [34]	Reguard [25], Securify [32], SmartCheck [30]	Mythril [3],EthIR [12]	Vandal [16]	Reguard [25], SmartShield [34]	ContractFuzzer [23], Ethploit [33]	SmartCheck [30]	SmartShield [34]	✗
2.	Out-of-Gas - Failed Send Exception	Mythril[3]	✗	✗	Mythril[3]	Vandal[16]	✗	✗	✗	✗	✗
3.	Unpredictable state due to mishandled exceptions	SmartCheck [30], Securify [32]	ContractFuzzer [23], SmartCheck [30]	Securify [32], SmartCheck [30]	EthIR[12]	✗	✗	ContractFuzzer [23]	✗	✗	✗
4.	Call-to-Unknown	Mythril[3], SmartCheck [30], Securify [32]	SmartCheck [30], SmartShield [34]	Securify [32] , SmartCheck [30]	Mythril [3]	✗	MAIAN [2], SmartShield [34]	ContractFuzzer [23], Ethploit [33]	SmartCheck [30]	SmartShield [34]	✗
5.	Typecast	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
6.	Weak Field Modifiers	SmartCheck [30]	SmartCheck [30]	SmartCheck [30]	✗	✗	✗	Ethploit [33]	SmartCheck [30]	✗	✗
7.	Integer Underflow/OverFlow	Zeus [24], Mythril [3]	SmartShield [34]	✗	Mythril [3]	✗	SmartShield [34]	✗	SmartCheck [30]	SmartShield [34]	✗
8.	DoS by an External Call	SmartCheck [30]	SmartCheck [30], SmartShield [34]	SmartCheck [30]	✗	✗	SmartShield [34]	Ethploit [33]	SmartCheck [30]	SmartShield [34]	✗

(b) Usage of Ethereum Smart Contracts Format for Analysis by Frameworks/Detection Tools

S.no.	Framework/Detection Tool	Abstract Syntax Tree	Solidity Code	Intermediate Representation of Solidity code	Bytecode
1.	Mythril	✗	✗	✗	✓
2.	Zeus	✓	✗	✗	✗
3.	Oyente	✓	✗	✗	✓
4.	ContractFuzzer	✓	✗	✗	✓
5.	EthIR	✓	✗	✗	✓
6.	MAIAN	✓	✗	✗	✗
7.	SmartCheck	✓	✓	✗	✓
8.	Reguard	✓	✗	✓	✗
9.	Securify	✗	✓	✓	✓
10.	Vandal	✓	✗	✗	✓
11.	SmartShield	✗	✗	✓	✓
12.	Ethploit	✗	✓	✗	✗

been almost no research related to typecasts vulnerability. However, Figure[2] suggests that the most amount of Ether was lost due to Parity Multisig Wallet Attack which was an exploitation of this vulnerability that cost $150 Million worth of Ether, followed by the DAO attack which was an exploitation of reentrancy vulnerability that cost $70 Million worth of Ether. Literature review of the selected detection tools/frameworks for this paper shows that these tools/frameworks adopted one of the analysis methodologies mentioned in the background section of this paper to analyse Ethereum Smart Contracts Figure [1] illustrates the adopted analysis methodology by various research frameworks and detection tools for each vulnerability respectively.

The reentrancy vulnerability's statistics illustrated in Figure[1] show that most of the frameworks and detection tools Oyente [4], Mythril [3], SmartCheck [30], Vandal [16], Securify [32], EthIR [12] surveyed in this paper, adopted the static analysis method to detect this vulnerability in smart contracts. Static analysis methods can detect the existence of the pattern defined for this vulnerability, however, defining the pattern of this vulnerability is also a challenge. The confirmation of the existence of this vulnerability can be more accurately outlined by a successful reentrancy generating transaction from an external contract to the contract under test. Only two of the analyzed research works, ContractFuzzer [23], Reguard [25], utilized the combined static and dynamic analysis method which is believed to be a better analysis methodology for this vulnerability.

The out-of-gas due to failed send vulnerability's statistics depicted in Figure[1] shows that only static analysis methods were adopted by detection tools/frameworks to detect this vulnerability [16], [3].

Figure[1] shows that the vulnerability caused by mishandled exceptions where the state of a smart contract becomes unpredictable was found to be identified mostly by using static analysis SmartCheck [30], EthIR [12], Securify [32]. However, ContractFuzzer [23] also successfully detects this vulnerability using the fuzzing technique to generate multiple transaction scenarios. The Call-to-Unknown vulnerability was found to be detected by using combined static and dynamic analysis approach by two of the detection tools surveyed [23], [2]. (See Figure[1]) The weak field modifiers vulnerability was addressed by only one vulnerability detection tool [30] (See Figure[1]. The vulnerability caused due to unchecked math or more specifically Integer underflow/overflow ignorance was detected by two of the detection tools [24], [3]. Whereas, none of the detection tools had an analysis method to detect the vulnerability caused due to typecasting in Solidity.

6 CONCLUSION

This paper presents an analysis of the security vulnerabilities of Ethereum smart contracts, real-world exploitation cases of these vulnerabilities and their preventive techniques. Our paper targets eight security vulnerabilities in Blockchain 2.0 applications, specifically in Ethereum Smart Contracts. The vulnerabilities discussed are at the level of the application layer. The preventive techniques thus require alterations at the programming level. The research analysis and insights provided in this paper aim at directing the future study in this field towards the development of more robust vulnerabilities detection tools. Our analysis is based on

- the growing academic literature on the topic,
- the discussion forums and Internet blogs of smart contracts programmers.

ACKNOWLEDGMENTS

This work is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

[1] Accessed on 01-09-2019. Etherscan Home Page. https://etherscan.io/
[2] Accessed on 01-09-2019. MAIAN. https://github.com/MAIAN-tool/MAIAN
[3] Accessed on 01-09-2019. Mythril. https://github.com/ConsenSys/
[4] Accessed on 01-09-2019. Oyente. https://github.com/ethereum/oyente
[5] Accessed on 01-09-2019. Parity Multi-Sig Wallet Attack. https://blog.openzeppelin.com/on-the-parity-wallet-multisig-hack-405a8c12e8f7/
[6] Accessed on 01-10-2019. Bitcoin Home Page. https://bitcoin.org/
[7] Accessed on 01-10-2019. Ethereum Home Page. https://www.ethereum.org/
[8] Accessed on 01-10-2019. Geth Home Page. https://geth.ethereum.org/downloads/
[9] Accessed on 01-10-2019. Solidity Home Page. https://solidity.readthedocs.io/en/v0.5.1/
[10] Accessed on 01-10-2019. TXL Home Page. http://txl.ca/
[11] Accessed on 05-09-2019. NIST Bug Framework. https://www.nist.gov/publications/bugs-framework-bf-structured-approach-express-bugs
[12] Elvira Albert, Pablo Gordillo, Benjamin Livshits, Albert Rubio, and Ilya Sergey. 2018. EthIR: A Framework for High-Level Analysis of Ethereum Bytecode: 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings.
[13] M. Alharby, A. Aldweesh, and A. v. Moorsel. 2018. Blockchain-based Smart Contracts: A Systematic Mapping Study of Academic Research (2018). In *2018 International Conference on Cloud Computing, Big Data and Blockchain (ICCBB)*.
[14] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. 2017. A Survey of Attacks on Ethereum Smart Contracts SoK. Springer-Verlag New York, Inc., New York, NY, USA.
[15] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. 2017. A Survey of Attacks on Ethereum Smart Contracts SoK. In *Proceedings of the 6th International Conference on Principles of Security and Trust - Volume 10204*. Springer-Verlag, Berlin, Heidelberg.
[16] Lexi Brent, Anton Jurisevic, Michael Kong, Eric Liu, François Gauthier, Vincent Gramoli, Ralph Holz, and Bernhard Scholz. 2018. Vandal: A Scalable Security Analysis Framework for Smart Contracts. *CoRR* abs/1809.03981 (2018).
[17] M. di Angelo and G. Salzer. 2019. A Survey of Tools for Analyzing Ethereum Smart Contracts.
[18] A. Dika and M. Nowostawski. 2018. Security Vulnerabilities in Ethereum Smart Contracts.
[19] Wesley Dingman, Aviel Cohen, Nick Ferrara, Adam Lynch, Patrick Jasinski, Paul Black, and Lin Deng. [n.d.]. Defects and Vulnerabilities in Smart Contracts, a Classification using the NIST Bugs Framework. *International Journal of Networked and Distributed Computing*.
[20] Thomas Durieux, João Filipe Ferreira, Rui Abreu, and Pedro Rodrigues Souza Cruz. 2019. Empirical Review of Automated Analysis Tools on 47, 587 Ethereum Smart Contracts. *ArXiv* (2019).
[21] J. Feist, G. Grieco, and A. Groce. 2019. Slither: A Static Analysis Framework for Smart Contracts. In *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*.
[22] Huru Hasanova, Ui-jun Baek, Mu-gon Shin, Kyunghee Cho, and Myung-Sup Kim. 2019. A survey on blockchain cybersecurity vulnerabilities and possible countermeasures. *International Journal of Network Management* 29 (01 2019).
[23] Bo Jiang, Ye Liu, and W. K. Chan. 2018. ContractFuzzer: Fuzzing Smart Contracts for Vulnerability Detection *(ASE 2018)*. ACM, New York, NY, USA.
[24] Sukrit Kalra, Seep Goel, Mohan Dhawan, and Subodh Sharma. 2018. ZEUS: Analyzing Safety of Smart Contracts.
[25] Chao Liu, Han Liu, Zhao Cao, Zhong Chen, Bangdao Chen, and Bill Roscoe. 2018. ReGuard: finding reentrancy bugs in smart contracts. ACM.
[26] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. Making Smart Contracts Smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA.
[27] Purathani Praitheeshan, Lei Pan, Jiangshan Yu, Joseph Liu, and Robin Doss. 2019. Security Analysis Methods on Ethereum Smart Contract Vulnerabilities: A Survey.

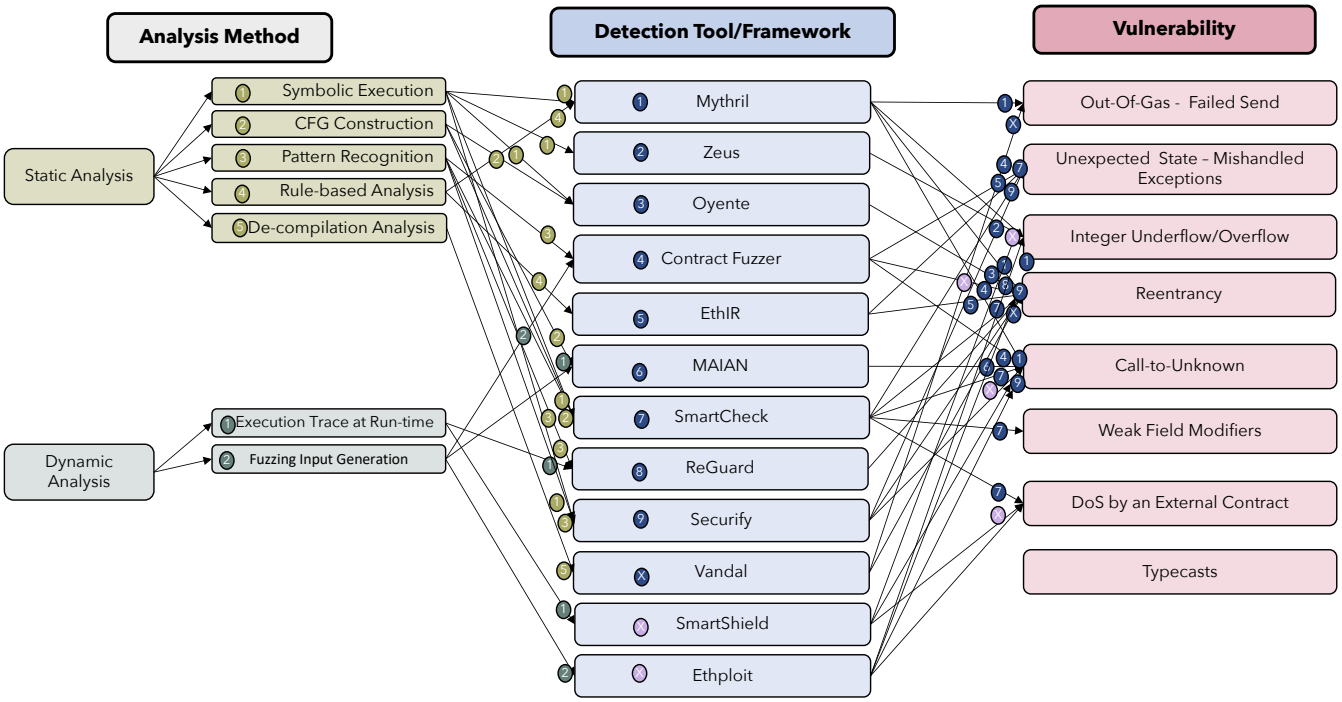
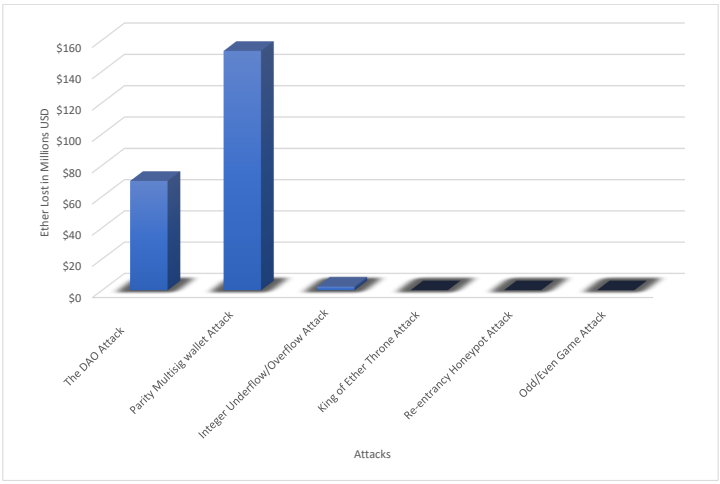
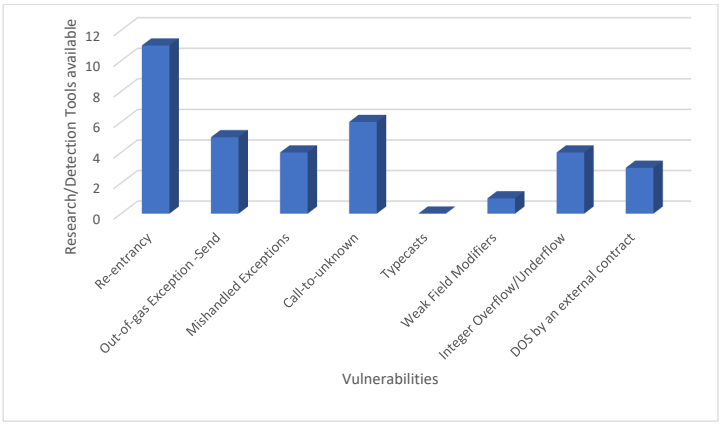


Figure 1: Relationship between Framework/Detection Tools Available and Vulnerabilities



(a) Most Severe Attack in terms of Ether lost



(b) Most Researched Vulnerability

Figure 2: Research Statistics of vulnerabilities

[28] Purathani Praitheeshan, Lei Pan, Jiangshan Yu, Joseph Liu, and Robin Doss. 2019. Security Analysis Methods on Ethereum Smart Contract Vulnerabilities: A Survey.

[29] Muhammad Saad, Jeffrey Spaulding, Laurent Njilla, Charles A. Kamhoua, Sachin Shetty, DaeHun Nyang, and Aziz Mohaisen. 2019. Exploring the Attack Surface of Blockchain: A Systematic Overview. *CoRR* (2019).

[30] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov. 2018. SmartCheck: Static Analysis of Ethereum Smart Contracts.

[31] Christof Ferreira Torres, Julian Schütte, and Radu State. 2018. Osiris: Hunting for Integer Bugs in Ethereum Smart Contracts. In *Proceedings of the 34th Annual Computer Security Applications Conference* (San Juan, PR, USA) *(ACSAC '18)*. Association for Computing Machinery, New York, NY, USA.

[32] Petar Tsankov, Andrei Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Bünzli, and Martin Vechev. 2018. Securify: Practical Security Analysis of Smart Contracts.

[33] Q. Zhang, Y. Wang, J. Li, and S. Ma. 2020. EthPloit: From Fuzzing to Efficient Exploit Generation against Smart Contracts. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*.

[34] Yuyao Zhang, Siqi Ma, Juanru Li, Kailai Li, Surya Nepal, and Dawu Gu. 2020. SMARTSHIELD: Automatic Smart Contract Protection Made Easy.

Towards Topology Aware Pre-Emptive Job Scheduling with Deep Reinforcement Learning

Bon Ryu
York University
Toronto, ON
bonryu@eecs.yorku.ca

Aijun An
York University
Toronto, ON
aan@eecs.yorku.ca

Zana Rashidi
York University
Toronto, ON
zrashidi@eecs.yorku.ca

Junfeng Liu
IBM Canada
Markham, ON
jfliu@ca.ibm.com

Yonggang Hu
IBM Canada
Markham, ON
yhu@ca.ibm.com

ABSTRACT

We present a topology aware Deep Reinforcement Learning (DRL) scheduler that simultaneously chooses jobs to run and elastically allocates resources to them for Distributed Deep Learning data parallel jobs in a multi-GPU, multi-machine cluster. This work addresses multiple limitations in the state-of-the-art methods: 1) Not sufficiently accounting for the bandwidth sharing between multiple jobs running simultaneously in a cluster, 2) Using overly simply heuristics to solve the resource allocation problem, 3) Pretending that job speed is not affected by the topology of allocated resources in simulation environments. This DRL method calculates unique job speeds by taking advantage of a graph representation of the cluster topology. This enables modeling realistic sharing of inter and intra machine bandwidths such as QPI speed, CPU-GPU speed, GPU-GPU speed, Infiniband card to Top-of-Rack Switch, etc. Our neural network model is trained using the REINFORCE algorithm which is a policy gradient method. The model outputs a multiple softmax designed to represent an assignment table that specifies the resource allocation of GPU's to Jobs. Using this design we can dynamically choose/change which GPUs to assign to which jobs at discrete time steps. Our simulation experiments show that our method can outperform baseline schedulers that use heuristics for job picking and resource allocation.

CCS CONCEPTS

• **Computing methodologies** → **Planning and scheduling**; **Reinforcement learning**; Parallel computing methodologies.

KEYWORDS

GPU job scheduling, reinforcement learning, neural networks

ACM Reference Format:
Bon Ryu, Aijun An, Zana Rashidi, Junfeng Liu, and Yonggang Hu. 2020. Towards Topology Aware Pre-Emptive Job Scheduling with Deep Reinforcement Learning. In *Proceedings of the 30th Annual International Conference on Computer Science and Software Engineering (CASCON '20)*. Toronto, ON, IBM Corp., USA, 10 pages.

CASCON '20, Nov 10–13, 2020, Toronto, Canada

1 INTRODUCTION

Job scheduling is an important part of running a computing cluster, which could be a massively parallel supercomputing centre, a cloud data centre, or an on-site group of servers at a university research lab, or even possibly a small cluster of edge devices. Jobs must be scheduled both in terms of when they should run, as well as what resources to assign them. Colloquially, job scheduling refers to both tasks. In this paper, scheduling a job in time will also be referred to as job picking, and resource to job assignment will be referred to as resource assignment or resource allocation. In this paper, we focus on the problem of both job picking and assigning resources to data parallel Deep Neural Network (DNN) jobs that are meant to run on GPU-capable cloud-based data centres. In short, we present a Deep Reinforcement Learning (DRL) method that leverages the modeling of intra and inter machine network topology and decides which jobs to run by elastically assigning GPU resources to them.

Most schedulers use heuristics (such as the shortest job first) for GPU resource scheduling. Heuristic-based methods can produce good solutions in some situations, but they often lead to a solution far from an optimal one in many other situations. Recently, deep reinforcement learning has been used for GPU resource scheduling [8], which uses a DNN as the policy function for reinforcement learning to learn a scheduler by interacting with the environment. However, these DRL-based methods fail to consider the topology of the resources within the cluster. Further, these schedulers usually solve the job picking problem and do not deal with resource allocation which is crucial in order to design an effective scheduler. Preemption is also another issue not dealt with in schedulers currently in use. We discuss some of these in detail below.

Currently, there exists no "intra" and "inter" machine topology aware DRL-based scheduler that can schedule and assign resources to multiple jobs on a multiple machine, multiple GPU cluster. This includes most of the intra and inter machine bandwiths such as QPI speed, CPU-GPU speed, GPU-GPU speed, Infiniband card to Top-of-Rack Switch speeds, etc. Both the topology of resources allocated to a job and the resulting bandwidth sharing between multiple jobs affect the unique speeds of jobs. In other words, job speeds and resource assignments (aka allocations) have a non-linear relationship. Schedulers that fail to sufficiently taking into account

topology in their decision making, will fail to make optimal job picking and resource to job assignment decisions, resulting in sub-optimal scheduling performance. This study addresses this issue through detailed modelling of intra and inter network topology, bandwidth sharing, and unique job speeds. Unlike heuristic methods, DRL schedulers can in theory avoid modeling the topology by training their policy function (modeled by a neural network function approximator) directly on bare metal of one's cluster, but one loses the ability to use simulations to initialize the training of the neural network (NN) function approximator. In fact, the model parallelization work by [9] used DRL to train a NN scheduler on bare metal on a single machine for single jobs at a time, but would need time consuming re-training on bare metal if the topology or cluster machines changes.

Another issue not thoroughly explored in literature are preemption strategies for DDL. Preemption means that resource allocation of a running job can be changed after the job has started but not yet completed. For example, in the no-preemption case, the resources (i.e. GPUs) allocated to a job does not change, and a job cannot be paused/resumed once started. In what we call *partial preemption*, the number of GPUs assigned to a job can be changed during runtime but the job is not allowed to be totally paused. In what we call *full preemption*, jobs can also be paused/resumed but the number of GPUs assigned upon resuming can be different than before the job was paused. Full preemption allows the most elasticity of job scheduling, and better/fuller use of resources, potentially leading to a shorter makespan for a set/sequence of jobs.

In this study, we address both intra and inter network topology considerations, job scheduling, and resource assignment/allocation. We make some initial investigations into using a DNN as a full preemption scheduler that is trained with RL, and compare this to no-preemption heuristic baselines. To hasten the development and investigation of DRL to solve DNN job scheduling/resource allocation problem, this work remains in a simulation environment. Still, our initial findings compel us to believe full preemptive schedulers could can make better/fuller use of resources than schedulers that can only perform no preemption at all.

Our contributions are summarized as follows. We propose a reinforcement learning based GPU resource scheduler, called **RL-TAPS** (Reinforcement Learning based Topology-Aware Preemptive Scheduler), that considers the topology of the underlying infrastructure. **RL-TAPS** solves both the job selection and resource allocation problems at the same time. Furthermore, our method allows for preemption. Our use case in this paper is data parallel distributing deep learning jobs across multiple GPUs in multiple machines, although **RL-TAPS** can be applied to other job types with different resources. The results from both streaming and non-streaming scenarios show the superiority of our method compared to three baseline methods.

The paper is organized as follows: In section 2 we discuss related work and section 3 gives a brief background in reinforcement learning and the policy gradient algorithm. In section 4 we describe our proposed method and we report the evaluation results in section 5. Finally we share related insights and discuss future work.

2 RELATED WORK

2.1 Mathematical Programming vs Heuristics

It is quite useful to note for context that many combinatorial optimization problems have traditionally been solved by formulating a mathematical program and using algorithmic methods such as simplex, or branch and bound, to solve them. If mathematical programming approaches are too slow, then a custom heuristic would be designed. The general problem of which fraction of which resource to assign to which job can be formulated as a Mixed Integer Non-Linear Programming (MINLP) problem. By constraining decision variables to be discrete (i.e we assign whole GPUs to jobs), we can formulate an Integer Non-Linear Programming (INLP) problem. We can further relax the problem to be an Integer Linear Programming (ILP) problem by making some big assumptions such as pretending to know the job completion times ahead of time or setting the number of GPUs assigned per job to be equal and fair.

Two notable recent works in literature for scheduling and assigning resources for data parallel, parameter server (PS) based, Distributed Deep Learning (DDL) jobs on clusters, are Tiresias [7] and Optimus [11]. With respect to the resource assignment problem (aka job/device placement), the authors of Tiresias first tried to formulate an ILP program that minimized network bandwidth usage. Even despite assuming equal resource assignment to linearize the problem, their ILP solution was too slow. The authors of Optimus merely described their resource assignment problem as an INLP problem that minimized Job Completion Times. They state that the problem is NP-hard, and decided instead to use a heuristic for assigning resources to jobs. Both Tiresius and Optmius ultimately used heuristics for both time based job scheduling and the resource assignment. For added context, even prior to learning of these works, we attempted to solve the general MINLP problem with mathematical programming, but failed to find a formulation that produced fast nor close to optimal results.

Popular cloud-based cluster computing schedulers such as Yarn and Slurm use very simple heuristics for time based job scheduling, such as DRF [5], First in First Out (FIFO), Shortest Job First (SJF), etc., or some combination of these. They also use very simple heuristics for resource allocation. These heuristics do not properly consider network topology nor do they adjust resource assignment to existing jobs to better utilize resources. We believe that heuristic methods for job scheduling and resource allocation, will be ultimately inferior to the potential benefits that DRL can provide, namely due to the ability of DRL to solve highly non-linear and complex problems.

2.2 DRL-based schedulers

There have been a resurgence in the study of using Neural Networks to solve optimization problems with the creation of pointer networks by [14]. Vinyals et al. [14] used existing algorithmic mathematical programming approaches to supervise the training of their pointer networks. More recently though, Bello et al. [1] built upon the work of [14] by using Deep Reinforcement Learning (DRL) to train pointer networks. Deep Reinforcement Learning (DRL) can train a Deep Neural Network (DNN) function approximator to solve complex non-linear decision problems without explicitly labeled data, by instead using scalar reward signals to guide the iterative

training of NNs. The first known cases of using DRL for job-shop scheduling problems, however, were by Zhang and Dietterich in 1995 and 1996 [16–18].

Some recent works have begun to use DRL for GPU resource management [4, 8, 9]. Mao et al. [8] used a DRL policy gradient method to choose jobs in a queue in a simple simulation environment with no network topology considerations. Their input layer for their simple Multi Linear Percepteron (MLP) policy network was a rectangular grid that represented the resources as contiguous columns, with each row representing a discrete time snapshot. Thanks to the simplicity of their design and availability of their code on github, this work has become very influential to subsequent investigations into using DRL for resource management for not just the field of cluster scheduling but also networking. [4] was inspired by [8] for example to pipe the input into a convolutional neural network (CNN) to choose a job, and then subsequently use a small NN to choose one of two heuristic placement algorithms. Both of these works ([4, 8]) are applicable only to data parallel jobs, as is our current work, which is also based on the work of Mao et al. [8].

Device placement/allocation for model parallelism of DL jobs is a much more complicated problem, which has been investigated on a single job and single machine basis by [9, 10]. Those authors brilliantly used sequence to sequence pointer networks to take a computation graph as input, and output a sequence of devices to assign to the nodes of the computation graph. Perhaps in the future, a DRL based method of automatically performing model and data parallelism simultaneously will be developed. For the time being, however, our current work attempts to address some of the many unsolved issues in DRL for data parallel DDL.

3 BACKGROUND

We give a short background on reinforcement learning and the policy gradient algorithm that will be used in the proposed method.

3.1 Reinforcement Learning

In reinforcement learning, an agent interacts with an environment and learns to take actions such that it maximizes some performance measure. The environment is represented via a state space and the agent sees a state s_t at each time step t. It then takes an action a_t based on the current state and is rewarded r_t by the environment while transitioning to the next state s_{t+1}. An episode is defined as a sequence of triples of state, action, reward, $((s_t, a_t, r_t))_{t=1..T_{max}}$, where T_{max} is the maximum length of the episode, and is finite.

Reinforcement learning is different from other forms of learning in that the agent has no explicit labelled data about the environment and other variables (i.e. states, reward, action). The agent collects data via interacting with the environment and learns optimal action sequences through experience.

3.2 Policy Gradients

In the theoretical development of the policy gradient algorithm for the episodic case, one first starts with the objective,

$$J(\theta) \doteq v_{\pi_\theta}(s_0), \tag{1}$$

where s_0 is the start state of an episode, θ are the parameters of a function π_θ, the policy determined by the parameters. A policy function π is a mapping from states S to actions A and can be defined as a probability distribution over actions a given a state s. Since the number of state and actions pairs can be very prohibitively large, function approximators such as neural networks are used to represent a policy. Thus a policy defined through a neural network with parameters θ is written as π_θ. v_{π_θ} is the performance measure called "true value function" and is the only function that can inform us of what actions to take (in transitioning between states) to get the highest possible performance. Theoretically, it is difficult if not impossible to determine v_π. In practice, it is common to approximate the objective as the expected return, $\mathbb{E}_{\pi_\theta}[G_t]$, where $G_t = [\sum_t^{T_{max}} \gamma^t r_t]$ [13, 15]. The discount factor γ is set between 0 and 1 and is used to put more weight on immediate rewards and discount later rewards. For the episodic case it is commonly set to 1 for simplicity. The REINFORCE algorithm [15] then uses the following update to the parameters:

$$\theta \leftarrow \theta + \alpha \mathbb{E}_{\pi_\theta} \left[G_t \nabla_\theta \log \pi_\theta(s_t, a_t) \right] , \tag{2}$$

where we are taking the expectation over many trials and time steps, sampled using π_θ. While we use the original formulation of G_t, it can be tailored to the problem being solved.

4 PROPOSED METHOD

In this section, we describe the problem we are solving, our proposed approach to solving the problem, and the simulation environment we train and test our proposed model.

4.1 Problem statement

Given a cluster of computers, each with one or more GPUs, and a set of deep learning jobs including the ones that are currently running on the cluster and the ones that were submitted and are waiting for GPU allocation, the goal is to determine:

(1) which jobs should be chosen to run.
(2) which GPUs should be assigned to jobs chosen to run.

so that the average job slowdown among all the jobs is minimized. A job's slowdown is defined as the difference between its finish and enter time, divided by its estimated time to run alone.

Note that we do not separate job selection and job allocation in this problem definition, and consider them simultaneously so that jobs are selected only if they can lead to an allocation that achieve a better outcome. Also, we allow preempting the already running jobs and elastically changing their resource allocations to improve overall performance.

4.2 Topology Awareness

Our scheduler is topology-aware in two ways: 1) intra and inter machine bandwidths in the cluster's topology are modelled in detail as part of the environment, and affect the observed reward signal; 2) the current GPU to job assignments (partially representing the cluster topology) are presented as input to an NN policy function. An NN input design that incorporates topology bandwidth information is left for future work. Even without bandwidth information directly incorporated into the input, the NN can in theory find GPU to job assignment combinations that make better use of the cluster topology and thus increase the reward signal.

4.3 State representation

The state S describes the status of the cluster in terms of GPU usage and the jobs in the job queue. Assume that the cluster has a set of X machines, $\{n_i | i = 1..X\}$, with a total number of M GPUs, $\mathbf{G} = \{g_i | i = 1 \ldots M\}$. Also assume that the job queue can take up to N jobs and each element of the queue is called a job slot. Each job slot can hold one job, and the slots are sequentially filled up as jobs arrive. We represent the set of N job slots as $\mathbf{J} = \{j_i | i = 1 \ldots N\}$.

Similar to [8], we use a set of matrices to represent the state, shown in the top part of Figure 1. Firstly, X two-dimensional matrices are dedicated for the **cluster representation**, one matrix per machine. Next, N two-dimensional matrices represent the N job slots in the job queue, one matrix per job slot. Finally, there is a matrix to represent a backlog.

The state representation in Figure 1 shows of a small cluster of $X = 4$ machines (labeled $n1$ to $n4$) and a job queue with $N = 3$ job slots (labelled $j1$ to $j3$). The first two job slots are occupied by a running job. The job in the third slot is waiting. In Figure 1, the number of columns of the matrix for machine n_i is the number of GPUs it has (e.g., n_1 has 4 GPUs). The number of columns of the matrix for a job slot j_i is the maximum number GPUs a job can request (e.g., 6 GPUs in the figure). In general, the rows of the matrices represent the time dimension, and the columns represent the resource dimension. Each row of the matrices is a representation of the entire environment along the columns (cluster usage, job resource request, etc.) during a specific discrete span of time. The multiple rows then allows for the presence of sequential information to be represented in the state.

The highlighted cells in the matrices representing machines indicate that the corresponding GPUs are occupied by the current running jobs. The highlighted columns in a matrix representing a job slot in the queue indicate the number of GPUs requested by the job, $gpusreq(j)$. The time span of each row of the image is calibrated to have a constant value in seconds (see Appendix A). Thus if we know the amount of computation a job has left, and its estimated speed, we can determine the number of times steps required for this job. In a job slot, the number of highlighted rows of a job is the number of time steps it is expected to run for if the job is 1) running alone, 2) is allocated exactly the number of GPUs it requests, and 3) there is zero communication time between GPUs. In the cluster representation, the number of highlighted rows of a job is the number of time steps left for it to complete, given its current unique topology-dependent speed, and the calibrated time span of the row. Job modelling is further explained in section 4.6.2.

When a job arrives at the cluster, but there are no empty job slots, the job gets appended to the small backlog matrix (illustrated at the right most end of the top part of Figure 1), for which a single job is represented by a single unit.

When a job is assigned GPUs by **RL-TAPS**, we paint the corresponding GPU columns in the cluster representation with the same color. In our full preemptive method, **RL-TAPS**, jobs that arrive to the job slots remain there as long as they are not finished. They can be not-started, paused, or currently running. Only when the jobs are finished are they removed. This way, currently running, paused, and not-started jobs in the queue are all candidates for being chosen by **RL-TAPS**. In contrast, in the RL simulation for no-preemption schedulers, jobs that are chosen (and thus assigned a static number of GPUs), are removed from their jobslot, so they only occupy the cluster representation and are no longer candidates for being chosen.

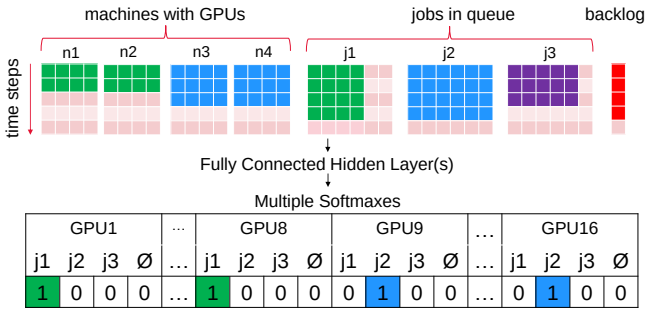


Figure 1: The architecture of the NN function approximator. Above there are three jobs in the queue. The first two are running, and the third is waiting. This assignment is not optimal but possible, depending on the level of training.

To simplify the "painting" of the cluster representation, we do not stack jobs vertically in the cluster representation. In other words, each column of the cluster representation is assigned to at most one job. This state representation will be used as the input to the neural network policy function.

4.4 Actions and Rewards

4.4.1 Actions, A. The NN takes multiple simultaneous actions using a multiple softmax approach. If there are M GPUs in the cluster, there are M output softmaxes, and all softmaxes share the previous hidden layer. Each softmax, as shown in Figure 1, consists of $N + 1$ units, where N is the number of job slots. The corresponding GPU-to-job assignment table is shown on the right hand side of Figure 2. In this study, each GPU can be assigned to at most one job slot, or not assigned to any job slot, i.e., the null job slot $\varnothing$. The decision of which job to assign a single GPU to, is done using a single softmax. With multiple GPUs in a cluster, we have multiple softmaxes, one per GPU. Note that if the job slot corresponding to a currently-running job is not assigned to any GPUs, it is paused.

In this manner, we can solve the problem of choosing a job and assigning resources to it, simultaneously. Furthermore, this approach is naturally pre-emptive. A job that has not finished, remains in its slot, and can be paused or resumed simply by not assigning or assigning GPUs to it. If a job is not assigned any GPUs, then it is not chosen. Thus, job picking and resource assignment are performed at the same time.

4.4.2 Topology-Aware Reward, R. In RL, a scalar signal for reward is needed to guide the training of the policy function (a neural network in our case). It can also be helpful to subtract penalties from the reward to discourage bad decisions. We describe exactly how we formulate rewards and penalties. The main measure of reward we use is computational throughput in units of FLOPS (Floating Point Operations Per Second), which we interchangeably refer to as speed. In general, we calculate the aggregate sum of speeds of all jobs.

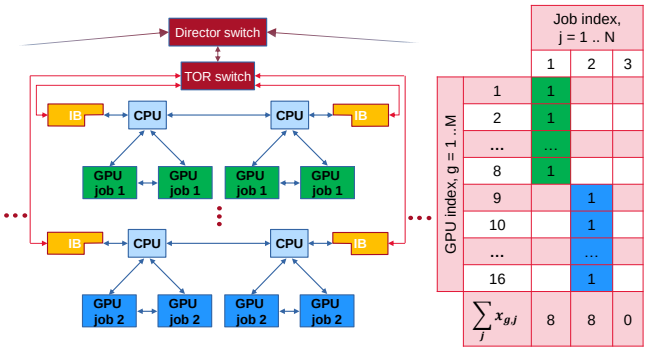


Figure 2: (Left) A rack is shown with its Top of Rack Switch (TRS), and two of its machines. The vertical ellipsis mean that there exists additional machines. Horizontal ellipsis on either side mean there may exist additional racks. The dark red arrows pointing to the DS indicate connections from two additional TRS's on either side. (Right) Example assignment of GPUs 1 to 8 for job 1 and GPUs 9 to 16 for job 2.

Rather than considering the instantaneous speed of a job, we prefer for simplicity to think about the average speed that a job experiences while interacting with other running jobs. Since time steps are not instantaneous but rather span two time points, it is suitable to consider the average speed of a job. The job's average speed at a discrete time step contributes to the total reward for the time step. This contribution is expressed as,

$$R_{t(j)} = v_{t(j)} = \frac{d_{m(j)}|\mathrm{G}_{(j)}|}{tt_{m(j)} + rt_{(j)}}, \tag{3}$$

where $d_{m(j)}$ is the computational "distance" of job j per minibatch. I.e. $d_{m(j)} = d_{ex(j)} m_{(j)}$, where $d_{ex(j)}$ is the computational "distance" of job j on a single example, and $m_{(j)}$ is the minibatch size. The computational distance is defined as the number of FLOPs (Floating Point Operations)[1]. $\mathrm{G}_{(j)}$ is the subset of GPUs assigned to job j. $|\mathrm{G}_{(j)}|$ is thus the number of GPUs assigned to job j. In the denominator, $tt_{m(j)}$ is the training time of a minibatch for job j. $rt_{(j)}$ is the time it takes for a reduction operation at the end of a minibatch such as gradient averaging across the GPUs of a job. Since reduction time is independent of the minibatch size, but is still characteristic of a job running in job slot j, $rt_{(j)}$ lacks a m subscript.

For each time step t of an episode i, we calculate a reward value for the time step as a sum of the throughputs (equation 3) across currently running jobs, in addition to any penalties associated with that time step.

$$R_t^i = \sum_j R_{t(j)} + Penalties \tag{4}$$

The superscript episode index, i, is omitted on the RHS of equations 3 and 4 for simplicity.

There were two main penalties designed to help aid the neural network to train.

[1]We use the term distance as proxy for computation (FLOPs not FLOPS) to intuitively use kinematic equations such as $\Delta d = vt$

(1) The cost of a job sitting idle in a jobslot or backlog, whose magnitude is calculated as

$$C_{t(j)} = \frac{d_{m(j)} gpusreq(j)}{tt_{m(j)} + srrt_{(j)}}, \tag{5}$$

where $srrt_{(j)}$ is called the single rack reduction time and is an estimate of the job's reduction time if the job running alone on a single rack using $gpusreq(j)$ GPUs.

(2) The cost of fewer than requested GPUs being assigned to the job in a jobslot, whose magnitude is calculated as

$$C_{t(j)} = \frac{d_{m(j)} [gpusreq(j) - |\mathrm{G}_{(j)}|]}{tt_{m(j)} + rt_{(j)}}, \tag{6}$$

The first penalty applies to both **RL-TAPS** and no-preemption methods. Without this penalty, **RL-TAPS** would fail to learn to decrease slowdown by running multiple jobs simultaneously. This is because running one job at a time with high resource usage could also increase throughput but comes at the expense of longer idle time for waiting jobs. The second penalty only applies to **RL-TAPS** and it is necessary for training the NN to learn to assign GPUs close to the number that is requested. Assigning more GPUs to a job than requested can be beneficial and is counted in equation 3.

4.5 Policy Function Design and Training

We use a neural network to represent the policy function.

4.5.1 Neural Network Structure. The overall NN architecture is shown in Figure 1. Its input, described in section 4.3, represents the state of the environment. The input is connected to the output using a single fully connected hidden layer. The output layer of the NN consists of multiple softmaxes (one for each GPU), all sharing the previous hidden layer. Each softmax represents a probability distribution over job slots given a GPU. The bottom part of Figure 1 illustrates the output layer of the NN. Each softmax corresponds to a decision to assign a GPU to a job slot, with the highest probability element of the softmax corresponding to the most favoured job slot. In addition, a GPU may not be assigned to any job slot. Thus, a null job slot $\varnothing$ is used to represent such a situation.

At a given time step t, let A_g be the event that a GPU g is assigned to one of $N+1$ job slots, which corresponds to the action with the highest probability outputted by GPU g's softmax, $p(A_g|s)$. We use A_t to represent the event that events A_g for all GPUs occur simultaneously, that is, $A_t = \bigcap_{g=1}^{M} A_g$. Assuming independence among A_g's, we can use the product rule of probability to express a single policy function as:

$$\pi_\theta(A_t, S_t) = p(A_t|S_t) = \prod_{g=1}^{M} p(A_g|S_t) \tag{7}$$

where S_t is the given input at t to the NN. Note the capitalized states, actions, and rewards (S_t, A_t, R_t) signify that they are sampled using the policy. Such a single function is needed in the policy gradient training algorithm described below.

4.5.2 Training Algorithm. As described in the background section, the REINFORCE algorithm is a result of direct policy differentiation where the goal is to maximize the expected return. However, the policy gradients suffers from high variance. This is in part alleviated

by using the "REINFORCE with baseline" algorithm, which calls for subtracting an appropriate baseline, b_t from the return:

$$\theta \leftarrow \theta + \alpha \mathbb{E}_{\pi_\theta} \left[(G_t - b_t) \nabla_\theta \log \pi_\theta(s_t, a_t) \right] \tag{8}$$

where the choice of b_t should leave the expectation of the gradient on the RHS unchanged. The expected reward G_t and baseline b_t are computed over a set of episodes generated based on a set of training job sequences. In our training algorithm, b_t is the average of the return over multiple episodic simulations,

$$b_t = \frac{1}{E} \sum_{i=1}^{E} G_t^i, \tag{9}$$

where $i = 1..E$ is an index for episode. The expectation on the RHS of equation (8) is taken simply by averaging across all job sequences and episodes involved in calculating the gradient prior to a parameter update.

In order to generate multiple episodes per job sequence during training, a softmax for a GPU is treated as a probability distribution, from which a sample is taken to determine which job to assign the GPU to. During inference, a GPU is assigned to the job slot with highest probability outputted by the corresponding softmax.

At each time step t of an episodes, the NN would be required to make a decision of which GPUs to assign to which job slot, with the option of GPUs not be assigned to any slot (i.e. the null job slot). Then the actions are implemented by the scheduler, followed by a calculation of the reward for the current step t. Then finally the time step t is advanced by 1, and a new job can arrive for this new time step.

We trained the policy gradient method with two kinds of training loops. First we trained using a method akin to the usual mini-batch gradient based training method in a static dataset scenario. This training loop is shown in Algorithm 1. Second, we trained the NN in a streaming data scenario.

In both scenarios, multiple sequence of jobs, representing job arrivals, are made before running finite length episodic simulations. Each job sequence represents an arrival of one job per time step t. Each epoch involves multiple job sequences.

A single job sequence is used to generate multiple episodes. At each step of an episode i, as usual, we calculate a cumulative discounted reward, the return G_t^i. An aggregate baseline for each time step, b_t, is calculated by averaging v_t^i across all episodes. A gradient, $\Delta\theta$, is accumulated across multiple job sequences, episodes, and time steps,

There are three main differences between [8] and our study with respect to the training algorithm. Firstly, in contrast, we test the performance of our model on separate test scenarios. Secondly, [8] employed epoch based learning, in which NN updates occur once per epoch. In our work, we update the NN multiple times per epoch. Thirdly, we additionally test the performance of our method in the face of changing simulation data.

The training loop for minibatch style training is shown in algorithm 1. Training and test job sequences are pre-made and remain static, and thus we refer to this as the **Non-Streaming training** method.

Algorithm 1 was modified in two small ways to simulate **Streaming Data training**. Firstly, before shuffling the job sequences, one would simply re-initialize a new set of J training job sequences every few epochs. Secondly, a new testing job sequence was generated for every epoch. This approach was used to investigate the potential for the NN to train with new incoming job sequences.

Algorithm 1: Minibatch Training with Static Job Sequences

```
Initialize network parameters θ
Initialize J training job sequences
Initialize batchsize B
for each epoch:
    Shuffle training job sequences
    for job sequence l = 1..J:
        for episode i = 1..E:
            Generate episode sequence ((S_t^i, A_t^i, R_t^i))_{t=1..T_i}
            for t = 1..T_max:
                G_t^i ← Σ_{k=t}^{T_max} γ^{k−t} R_k^i
        for t = 1..T_max:
            b_t ← (1/E) Σ_{i=1}^{E} G_t^i
            for episode i = 1..E:
                Δθ ← Δθ + (G_t^i − b_t) ∇_θ log π_θ^i(A_t, S_t^i)
        if l mod B is 0:
            θ ← θ + α (1/BE) Δθ
```

4.6 Simulation Environment

4.6.1 Modeling Cluster Topology. The RL scheduler is trained on a simulated environment. A neural network designed in Theano is used as a function approximator. All code is written in Python. The graph representation of the cluster was modelled using the NetworkX python package.

The main benefit of building a simulation environment is it enables quick testing of different ideas, without having to wait for real DL jobs to run on a cluster.

Intra and inter machine network topology is simulated by modeling the cluster as a graph. We assume that each machine in the cluster is an IBM Power8 Minsky box (model S822LC [2]), and that the cluster can consist up to 4 racks. The main idea was to model elements such as CPUs, GPUs, network cards and switches as nodes in the cluster and the communication links between them as edges. The full list of nodes modelled is shown in Table 1. Each machine consists of 4 GPUs, 2 CPUs, and 2 Infiniband cards. Two GPUs are connected to each CPU. This type of configuration was described in [3], which benchmarked the use of IBM's PowerAI library for performing DDL on a 256 GPU cluster of IBM Power8 Minsky boxes.

The following edge weights between the nodes were modelled: default bandwidth, run-time bandwidth, number of jobs per edge. The node and edge types modelled are described in table 1, and a visual representation of the nodes and edges are shown in a graphical representation of a portion of the cluster on the left hand side of Figure 2.

The main idea behind topology modeling is to count the number of jobs using each edge in the graph, that is, the number of jobs per edge. To do so, we need to keep track of the links used by GPU to GPU pair paths. With a large number of resources, it is infeasible

Table 1: Nodes and Edges modelled

Nodes	CPU, GPU, Infiniband card (IBC), Top-of-Rack Switch (TRS), Director Switch (DS) [1]
Edges (Default BW in GB/s)[2]	CPU-CPU (38.5), CPU-GPU(40), CPU-IBC(1000), IBC-TOR(12.5), TRS-DS(6.25)
Edges Weights [3]	Default BW, Run-Time BW,[4] Number of Jobs per Edge

[1] Only one Director Switch was simulated.
[2] BW = Bandwidth
[3] Each edge has three weights
[4] Run-Time BW = Default BW / Number of Jobs per Edge

to list all possible combinations of GPU-GPU assignments ahead of time. Thus we pre-compute the shortest paths between each unique GPU-GPU pair. During run time, for each job, we build up a set of edges, which is the set summation of all edges in the paths of all unique GPU-GPU pairs being used by a job. Obviously, if a job was only assigned a single GPU, that job would not contribute to the job count of any edge. The "Number of Jobs per Edge" edge weight value is used in determining the effective speed of simulated jobs. The speed of jobs become important in the reward formulation which is described in section 4.4.2.

4.6.2 Job Modelling. As already mentioned, our job picking and resource allocation method is meant for DNN type jobs. Here we discuss the attributes of the simulated DNN jobs that our scheduler tries to schedule as they arrive in our simulation, not the attributes of the NN function approximate used in our DRL method. Symbols representing modelled job attributes are suffixed with "(j)".

Let us assume for simplicity that the time it takes for a minibatch of data to feed into a GPU to be negligible compared to the time it takes for a job to complete one minibatch of computation. Furthermore, consider a job that runs on a single GPU, which has a Floating Points Operations Per Second (FLOPS) rating of v_{P100}. $P100$ is an NVIDIA GPU model. Let us think about the complexity of a DNN model, i.e. the Floating Point Operations (FLOPs not FLOPS) of a single forward pass on a single data example, as a distance per example, $d_{ex(j)}$.

The speed of a single GPU job, $v_{t(j)}$, at a given time step t during the simulation, can be expressed as follows and is roughly equivalent to speed of the GPU.

$$v_{t(j)} \approx \frac{d_{m(j)}}{tt_{m(j)}} \approx 0.9 v_{P100}, \tag{10}$$

where the 0.9 on right hand side is to simulate the fact that the actual FLOPS observed is less than the advertised FLOPS.

To use realistic values of attributes of DNN jobs, a table of 35 Convolutional Neural Network (CNN) architectures and their properties such as model complexity ($d_{ex(j)}$), gradient size (same as the memory footprint of all model parameters), etc. were conveniently obtained from [12]. During job initialization, it is randomly assigned a CNN architecture's model complexity, and gradient size.

Also during job initialization, the training time of a job's minibatch, $tt_{m(j)}$, is obtained from equation 10, where v_{P100} is a constant, $d_{m(j)} = d_{ex(j)} m_{(j)}$ is known because $d_{ex(j)}$ is simply the complexity a of model as reported by [12]. $m_{(j)}$ is a job's assigned mini-batch size randomly chosen as one of $\{32, 64, 128, 256, 512\}$. Finally, borrowing from [8], we sample from uniform distributions (see section 5.1) to assign a job a length $len(j)$ and number of resources (only GPUs in our case) requested by the job, $gpusreq(j)$.

During the simulation, a job finishes once it has undergone a total amount of computation, which we refer to as "total computational distance":

$$d_{tot(j)} = len(j) \times gpusreq(j) \times d_{cell}, \tag{11}$$

where $len(j)$ and $gpusreq(j)$ multiply to give the number of colored cells of a job slot (see figure 1), and d_{cell} is a calibrated computational distance associated with a single cell, in units of FLOPs. The details of this calibration is included in the Appendix A. The purpose of the calibration process is two fold. One, it ensures that the input image can represent the entirety of the computational distance of the job, whether that be in the queue, or in the cluster portion of the input. Secondly, the calibration associates a time value in seconds, t_{step}, with a row of the input image. Every time the simulation advances one step, the computational distance in FLOPs each job has "travelled" can be computed uniquely as $v_{t(j)} \times t_{step}$.

4.6.3 Reduction Time Measurement and Formulation. The reduction time of a job, $rt_{(j)}$ is modelled using a combination of measured data and a heuristic. In our modeling, it is expressed as follows:

$$rt_{(j)} = srrt_{(j)}(1 + (r-1)/5)s_{(j)} \tag{12}$$

The right hand side of equation 12 consists of three terms. The first, $srrt_{(j)}$, is the single rack ring reduction time, derived from ring reduction measurements on a single rack for up to 16 GPUs. The second factor in parentheses is a simple heuristic to model the increase in reduction time due to using increasing number of racks, r. The third, $s_{(j)}$ is a scale factor that accounts for bandwidth sharing. These three terms are explained in more detail below.

Single Rack Ring Reduction Time, $srrt_{(j)}$: Ring reduction measurements were taken for up to 4 machines on a single rack, for a total of 16 GPU's. It was measured for 100 MB and 500 MB gradient sizes, and the measurements are shown in Figure 3. The measurements were fitted with square root functions. In order to estimate the reduction time for different gradient sizes and GPU counts, we interpolate linearly between the two curves by drawing a vertical line at the desired GPU count. At 0 MB, we set the reduction time to be 0, and for all gradient sizes greater than 100 MB, we simply use the equation of the line between the 100 MB and 500 MB curves. We use the two fitted curves in Figure 3 to account for different number of GPUs allocated to a job, and interpolated between them as explained above, to account for different gradient sizes.

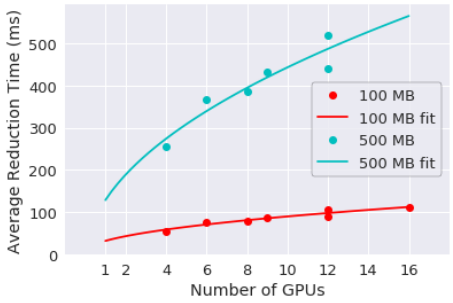


Figure 3: Single Rack Reduction Time measurements

Extrapolating Reduction time for Multiple Racks: Since at the time of measurement and data collection, reduction time measurements between racks were not available, we model between rack reduction times with the second term on the right hand side of 12. For example, if we require the reduction time of using 8 GPUs on 2 racks instead of 1 rack, we multiply the right hand side by $(1 + (2 - 1)/5) = 1.2$. Thus, for 2, 3, and 4 racks, this factor would be 1.2, 1.4 and 1.6 respectively.

Scale factor, $s_{(j)}$: Finally, the scale factor, $s_{(j)}$ is expressed as

$$s_{(j)} = \frac{limbw_{single(j)}}{limbw_{multi(j)}}, \tag{13}$$

where the numerator, $limbw_{single(j)}$, represents the limiting bandwidth of a job j if that job was running alone in the cluster. The denominator, $limbw_{multi(j)}$, represents the limiting bandwidth of a job j while there are multiple jobs running in the cluster. Given that a job j is assigned a set of GPUs, let $paths_{(j)}$ be the collection of shortest paths between unique GPU-GPU pairs among the assigned GPUs. Further, let $\mathbf{E}_{(j)}$ be the set of all undirected edges in $paths_{(j)}$. $limbw_{single(j)}$ and $limbw_{multi(j)}$ are defined as,

$$limbw_{single(j)} = \min_{e \in \mathbf{E}_{(j)}} \text{Default BW}(e) \tag{14}$$

$$limbw_{multi(j)} = \min_{e \in \mathbf{E}_{(j)}} \text{Run-time BW}(e). \tag{15}$$

To help remember the meaning of $limbw_{single(j)}$ and $limbw_{multi(j)}$, one can call them "single job limiting bandwidth" and "multiple job limiting bandwidth", respectively.

The main reason for using the scale factor is that it is impossible to pre-measure reduction times for the huge number of combinations of GPU to job assignments. The scale factor helps account for the difference between a job's reduction time due to sharing of bandwidth with other jobs during the RL simulation, versus the reduction time the job would have if it was running alone. This is needed since the reduction time measurements were carried out for one reduction process at a time on a cluster of four Minksy Boxes.

5 EVALUATION

We describe specifics of the cluster topology, dimensions of the NN, and the performance measures used for evaluation. Also, we briefly describe the non-topology aware baseline methods of comparison. Finally we present the results of our method.

5.1 Experimental Setup

We tested our method on a simulated cluster with a total of 8 machines on 4 racks, with 2 machines on each rack (which can be written down as [2,2,2,2]). Each machine has 4 GPUs, thus there is a total of 4 GPUs x 8 machines = 32 GPUs. Thus $M = 32$. Normally, one would keep as many machines on the same cluster as possible. However, as a proof of concept, we wished to investigate the effects of topology on the performance of our method compared to the non-topology aware baselines, while allowing a NN to train within a reasonable amount of time.

The number of job slots we used in the input image is $N = 10$, and the backlog could hold 60 jobs. The maximum $gpusreq(j)$ was limited to 12 per job, to limit the number of columns needed to represent a jobslot in the input image.

The number of hidden units used for the hidden layer was set to 1.5 times the total number of output units, rounded up. The total number of output units is $32 \times 11 = 352$. Updates to the NN parameters were done with an Adam optimizer and an initial learning rate of 0.001.

During each episode, one job per time-step would arrive for the first 400 consecutive time-steps. The maximum episode length, T_{max} was set to 1000. The number of episodes, E, per job sequence was set to 20. The number of job sequences, J, was different for the minibatch and streaming training experiments.

The length of the job, $len(j)$, and its resource request, $gpusreq(j)$, were sampled from uniform distributions. Roughly half the jobs had $len(j)$ between 1 and 3, and the other half between 6 and 10. Similarly, roughly half of the jobs had $gpusreq(j)$ between 1 and 7, and the other half between 8 and 12.

5.1.1 Non-Streaming vs Streaming Training. For non-streaming data training, a static set 200 training and 200 testing job sequences were created before the start of an epoch.

For streaming data training, a fresh set of 60 training job sequences were used every 10 epochs. At the end of every epoch, a fresh set of 60 testing job sequences were used for testing.

For both training scenarios, the training and testing performances were plotted for 50 epochs, every epoch.

5.2 Performance Measures

5.2.1 Mean Reward. To display the reward across all E episodes and J job sequences of a single epoch, the first return, G_1^i of each episode i is collected for every job sequence. Notice that the usual formula for the first return, G_1^i, is already an aggregate (a discounted sum) of all rewards of a single episode. In this paper, we refer to **Mean Reward** as the mean of all G_i^i across all job sequences and episodes of an epoch:

$$Mean\ Reward = \frac{1}{J}\frac{1}{E}\sum_j \sum_i G_1^i. \tag{16}$$

where job sequence index j is omitted in the return for simplicity.

5.2.2 Slowdown. In scheduling studies, it is also instructive to plot the slowdown. Slowdown for a single job is defined as,

$$Slowdown(j) = \frac{finishtime(j) - arrivaltime(j)}{len(j)}, \tag{17}$$

where $finishtime(j)$ is the time step at which a job finished. If by time step T_{max} there exists unfinished jobs, then those jobs are assigned T_{max} as the finish time. $arrivaltime(j)$ is the time-step at which a job arrives.

To associate an aggregate slowdown for an entire epoch, we calculate the **Mean Slowdown** by simply taking the average job slowdowns across all jobs that arrived during an epoch. This mean is taken across all job sequences and all episodes of an epoch.

5.3 Evaluation baselines

To compare with baseline job picking and resource allocation methods, we used the following static schedulers from [8]: Random, Shortest-Job-First (SJF), and Tetris. These schedulers only performed job picking, by selecting one job at a time among the jobs in the slots. The Random and SJF job pickers are self-explanatory. Tetris is based on [6]. Its implementation in our environment works by picking the job whose resource request $gpusreq(j)$, when multiplied by the number of available GPUs, leads to the largest value. For resource allocation for these schedulers, the behaviour of heuristic resource allocator from [8] was preserved. This baseline resource allocator simply numbered the resources of a certain type consecutively, then picked the first x resources to assign, where x is the number resources of a certain type requested by a job. The allocator would search for free resources in the cluster representation of the input image from left to right starting at the top of the image. For a given job, its resource allocations across the GPUs were required to begin at the same time step. Although this baseline resource allocator is not topology aware, we made sure to use the capabilities of our simulator to calculate the unique speed of jobs scheduled by these schedulers.

The Total Rewards and Average Slowdowns were also computed for these baseline schedulers. The extra penalties explained in section (4.4.2) were of course not used because they are not applicable to static resource allocators.

5.4 Performance Results

The results of measuring Mean Reward and Mean Slowdown for the non-streaming data and streaming data training experiments are shown in figure 4 and 5, respectively. The figures show that mean reward and slowdown performance of **RL-TAPS** is clearly better than the baseline schedulers. The baseline schedulers performed poorer because they do not learn to, nor even heuristically, take into account topology. In order for an NN to take into account topology, there must be some signal to guide it to understand that certain values of units in the input correspond to desirable choices made by the output. In the case of our prototype, this signal only came in the form of a topology-sensitive reward signal and current GPU to job assignments. To our surprise, the NN was able to learn even though the bandwidth usages of the topology were not explicit features in the input.

The non-streaming training method showed slightly better performance than the streaming training method with respect to the mean reward, but the performance with respect to mean slowdown were similar. This is good news as the end goal is to first train **RL-TAPS** in a simulation environment, but deploy and continue to train in a real cluster. The big benefit of detailed topology modelling

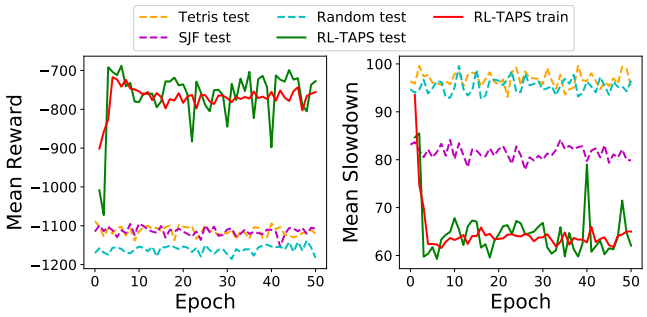


Figure 4: Mean Reward (top) and Slowdown (bottom) for the non-streaming data training scenario (see 5.1.1).

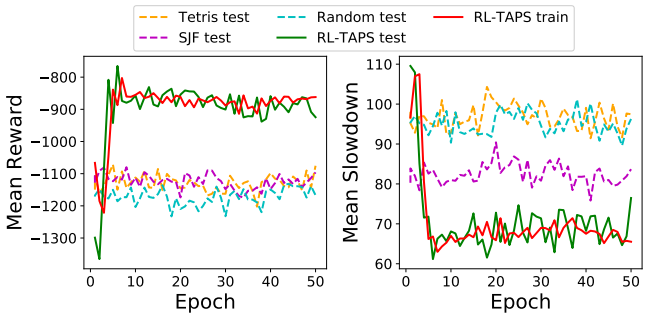


Figure 5: Total Reward (left) and Slowdown (right) for the streaming data training described (see5.1.1).

is that training can start in simulation and thus the scheduler could potentially be useful when it is deployed, without having to wait months to be trained.

6 DISCUSSION

Training NNs with the Policy Gradient method is often fraught with difficulties such as unstable policy parameters. Interesting and also concerning is that most of the learning in our experiment happens very early on and quickly plateaus. Further investigation is needed with regard to whether learning is stopping prematurely, or actually progressing well very quickly. With respect to the environment modelling, one limitation is that we have not yet modelled the cost of pausing and restarting DDL type jobs. Given the strong performance of **RL-TAPS** thus far, however, we are hopeful that **RL-TAPS** will still be capable of outperforming the no-preemption methods. Making this improvement may require some change to the input or the RL simulation. Another limitation currently is that the NN input design makes simulating large cluster sizes prohibitive. The jobslot representations are quite wasteful as many units of the input may end up with zeros. A fully connected hidden layer to a large input layer is not scalable. The large output search space also contributes to the high parameter count of our NN (roughly 1 million). Nonetheless, once trained, our NN completes a single inference step in less than 0.1 seconds on CPU. If incorporated into a production scheduler, **RL-TAPS** will not require many resources.

7 CONCLUSION AND FUTURE WORK

We proposed a deep reinforcement learning based scheduler that simultaneously selects jobs and assigns resources to them. Job selection and resource allocation are non-linearly related and this scheduler addresses the challenge associated with attempting to simultaneously solve these two dependent combinatorial optimization problems. Our scheduler **RL-TAPS** considers the topology of the environment and allows for full preemption. We evaluated the performance of **RL-TAPS** in different scenarios and compared it against various baselines demonstrating the efficiency and effectiveness of our method.

Going forward, we wish to address the scalability issue of the NN. In the short term, we will try reformulating the input into something that is more compact, with room to incorporate topology bandwidth information. There is also the need to solve additional assignment problems such as which reduction algorithm to assign to a job for both intra-machine and inter-machine communication. For example, Nvidia's NCCL library as well as IBM's DDL library consists of various different gradient reduction (i.e. averaging) algorithms for both within and between machine communication. The question of how to solve multiple simultaneous assignment problems without exploding the search space must be investigated.

In addition to the full preemption scheduling of **RL-TAPS**, we wish to explore other preemption methods mentioned in section 1. Designing a full-preemption NN was less complicated then designing an NN that can handle partial-preemption decisions due to the requirement of enforcing constraints in the latter. Currently, to our knowledge, there exists no NN architecture method that would allow one to enforce inequality constraints. The combinatorial optimization techniques with NNs to date have all avoided tackling such problems. We approached this issue in our work through penalties, but **RL-TAPS** fails to enforce them strictly. Likely this problem may require novel NN ideas to solve, and would be a very interesting endeavour.

REFERENCES

[1] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. 2017. Neural Combinatorial Optimization with Reinforcement Learning. *ArXiv* (Jan. 2017). http://arxiv.org/abs/1611.09940 arXiv: 1611.09940.

[2] Alexandre Caldeira, M. Kahle, Gerard Saverimuthu, and K. C. Vearner. 2015. IBM power systems S822LC technical overview and introduction. *IBM Red Paper* (2015).

[3] Minsik Cho, Ulrich Finkler, Sameer Kumar, David Kung, Vaibhav Saxena, and Dheeraj Sreedhar. 2017. PowerAI DDL. (Aug. 2017). https://arxiv.org/abs/1708.02188

[4] Giacomo Domeniconi, Eun Kyung Lee, and Alessandro Morari. 2019. CuSH: Cognitive ScHeduler for Heterogeneous High Performance Computing System. In *DRL4KDD 19: Workshop on Deep Reinforcement Learning for Knowledge Discover.* 7.

[5] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. 2011. Dominant resource fairness: fair allocation of multiple resource types. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation (NSDI'11).* USENIX Association, Boston, MA, 323–336.

[6] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. 2014. Multi-resource packing for cluster schedulers. In *Proceedings of the 2014 ACM conference on SIGCOMM - SIGCOMM '14.* ACM Press, Chicago, Illinois, USA, 455–466. https://doi.org/10.1145/2619239.2626334

[7] Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. 2019. Tiresias: A {GPU} Cluster Manager for Distributed Deep Learning. 485–500. https://www.usenix.org/conference/nsdi19/presentation/gu

[8] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. 2016. Resource Management with Deep Reinforcement Learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets '16).* ACM, New York, NY, USA, 50–56. https://doi.org/10.1145/3005745.3005750

[9] Azalia Mirhoseini, Anna Goldie, Hieu Pham, Benoit Steiner, Quoc V. Le, and Jeff Dean. 2018. A Hierarchical Model for Device Placement. (Feb. 2018). https://openreview.net/forum?id=Hkc-TeZ0W

[10] Azalia Mirhoseini, Hieu Pham, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. 2017. Device Placement Optimization with Reinforcement Learning. *arXiv:1706.04972 [cs]* (June 2017). http://arxiv.org/abs/1706.04972 arXiv: 1706.04972.

[11] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. 2018. Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters. In *Proceedings of the Thirteenth EuroSys Conference (EuroSys '18).* ACM, New York, NY, USA, 3:1–3:14. https://doi.org/10.1145/3190508.3190517 event-place: Porto, Portugal.

[12] Samuel. 2020. albanie/convnet-burden. https://github.com/albanie/convnet-burden original-date: 2017-08-04T10:11:16Z.

[13] Richard S. Sutton. 2018. *Reinforcement learning: an introduction* (second edition. ed.). The MIT Press, Cambridge, Massachusetts.

[14] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2017. Pointer Networks. *arXiv:1506.03134 [cs, stat]* (Jan. 2017). http://arxiv.org/abs/1506.03134 arXiv: 1506.03134.

[15] Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach Learn* 8, 3 (May 1992), 229–256. https://doi.org/10.1007/BF00992696

[16] Wei Zhang. 1996. Reinforcement learning for job-shop scheduling. (1996).

[17] Wei Zhang and Thomas G. Dietterich. 1995. A reinforcement learning approach to job-shop scheduling. In *IJCAI*, Vol. 95. Citeseer, 1114–1120.

[18] Wei Zhang and Thomas G. Dietterich. 1996. High-performance job-shop scheduling with a time-delay TD (λ) network. In *Advances in neural information processing systems.* 1024–1030.

A INPUT CALIBRATION

To integrate the simulation with the reduction time measurements, we associate a time span in seconds, $t_{rows(c)}$, with each row of the input image, and a computational distance for per cell, d_{cell}. The table below shows how $t_{rows(c)}$ is calibrated as the time in seconds it would take a vgg-vd-19 model to complete 1000 minibatch iterations using a single GPU, over the horizon of the input image.

Table 2: Formulas to derive $t_{row(c)}$ and d_{cell}

subscript c subscript f vgg-vd-19	calibration variable means final or total calibration DNN model
$d_{ex(c)}$	20×10^9 FLOPs, complexity of model
$m_{(c)}$	256, minibatch size of model
$d_{m(c)} = d_{ex(c)} m_{(c)}$	computational distance per minibatch
$it_{f(c)}$	1000, total number of job iterations
$g_{(c)}$	1, number of columns to represent the job
$d_{f(c)} = it_{f(c)} d_{m(c)} g_{(c)}$	total computational distance of the job
$v_{(c)} = v_{P100}$	GPU speed
$t_{f(c)} = \frac{d_{f(c)}}{v_{(c)}}$	total job run time in seconds
$rows_{(c)}$	10, number of rows (horizon)
$n_{(c)} = rows_{(c)} g_{(c)}$	number of highlighted cells for job in jobslot
$d_{cell} = \frac{d_{f(c)}}{n_{(c)}}$	calibrated computational distance per cell
$t_{row(c)} = \frac{t_{f(c)}}{rows_{(c)}}$	calibration time of each row in seconds

Pred-Cache: A Predictive Caching Method in Database Systems

Omar El Zarif, Safwat Hassan
(oelzarif,shassan)@cs.queensu.ca
Queen's University
School Of Computing
Kingston, Ontario, Canada

Ying Zou
ying.zou@queensu.ca
Queen's University
Department of Electrical and Computer Engineering
Kingston, Ontario, Canada

Calisto Zuzarte, Vincent Corvinelli, Mohammed Alhamid
(calisto,vcorvine)@ca.ibm.com
mohammed.alhamid@ibm.com
IBM Canada Ltd

ABSTRACT

The performance of large-scale systems (LSS) depends heavily on the time consumed in retrieving users' data from the databases. The database management system (DBMS) is essential to handle the storage and retrieval of users' data. Recent studies show that performance degradation in retrieving users' data can cause a severe revenue loss. Hence, improving the performance of the DBMS is essential for maintaining and enhancing user experience.

Query caching is a technique employed by the DBMS that presents immense improvements to the overall performance of the system. Prior work improves query caching techniques by maximizing the reuse of the cached queries (e.g., deciding on the beneficial queries to cache and deciding on the cache eviction and replacement policies). However, the existing work is tailored to specific server query languages and lacks in the adaptation to the different changing factors in the system, such as the occurrences of queries, the time of their occurrence, and query coupling.

In this work, we propose a predictive database caching framework, which can be deployed as a middleware layer independently from the database system. Our framework uses deep learning models to predict expensiveness (in terms of execution time) and the occurrences of queries to guide the caching process. We evaluate our framework using the TPC Benchmark DS (TPC-DS) database where we generate a 50GB database with 100,000 queries. Remarkably, our framework improves the cache hit ratio by 6% to 29% over the existing query caching mechanisms in the different benchmark scenarios that simulate the different types of query histories.

KEYWORDS

database systems, deep learning, neural networks, query caching

1 INTRODUCTION

The composition of modern software systems, particularly large-scale software systems, relies heavily on the interaction of the software with the database system to process the imposed huge amounts of data. *Query caching* is considered an essential technique to improve the performance of the software systems by improving the execution times in the database [8, 16, 17, 55].

CASCON'20, November 10 - 13 2020, Toronto, Canada

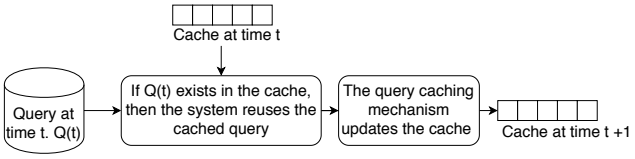


Figure 1: An overview of the query caching mechanism.

Query caching increases the performance of a system as it eliminates the re-execution of queries. Figure 1 shows an overview of the query caching mechanism. As shown in Figure 1, the new coming query at time t Q(t) is matched against the cache to eliminate the re-execution of queries in the system [38, 59]. Then, the query caching mechanisms decide whether to update the content of the cached queries if Q(t) is not already cached. The decision to cache a query should aim to maximize the reuse of the query, and thereby increase the system performance while adhering to the memory constraints of the system [36, 44]. Query caching is an extensively studied topic in database systems [4, 20, 38, 40, 42, 43, 59]. It has been approached by different aspects, such as identifying beneficial queries to cache in a system and deciding the optimum query eviction or replacement policies [4, 5, 40, 42].

Prior work proposes **reactive caching** mechanisms that maintain the cache content by controlling the cache eviction mechanisms, such as Least Recently Used (LRU) [36], Least Frequently Used (LFU) [44], and Least Recently Frequently Used (LRFU) [37] mechanisms. The content cached by applying reactive mechanisms is driven by the current access patterns in the system (e.g., caching the most frequently used queries). The reactive caching mechanisms are easy to implement. They proved their efficiency in the cases of repetitive data in the system. However, under constrained cache sizes these strategies lead to thrashing in the system [14]. The mechanisms are passive and slow as they start by caching a huge amount of insignificant data until popular data patterns start to emerge [10, 12, 61].

In contrast, **proactive caching** mechanisms are introduced to solve the slow responsiveness of the reactive caching mechanisms [40, 43, 62]. Proactive caching tends to evaluate the cost of queries (in terms of execution time) before deciding on the caching process. For example, prior work relies on the execution time estimation using query execution plans (QEPs) to cache the queries with a high-execution time [26, 43]. However, query execution plans are system-oriented and require manual work and tuning of the database system for better cost estimation [13, 31, 52].

To introduce a cache between the software's query request and its execution in the database system, we present a framework for

proactive query caching in database systems as follows. First, the framework **predicts the upcoming queries** using a recurrent neural network (RNN) [47]. Second, the framework **predicts the cost** (as the execution time and memory requirements) of the upcoming queries using a feed-forward neural network (FFNN) [65]. Finally, our framework caches the upcoming queries with long-execution time and low-memory requirements predictions (i.e., **prefetch and cache the cost-efficient queries**).

We assess the impact of using our framework on the performance of the system by calculating the cache hit ratio [71]. In particular, we compare the performance of our framework with the traditional reactive caching mechanisms (i.e., LRU, LFU, and LRFU mechanisms). We benchmark our work following various generated scenarios of query executions (e.g., running queries in sequential order) in the system. Each scenario was produced from the TPC-DS workload benchmark [49, 56] where we generated 100,000 queries on a 50GB database to simulate our study. In particular, we analyze the following research questions (RQs) to evaluate our framework:

RQ1: How accurate are the cost estimation and the prefetching functions?

The feed-forward neural network exhibits high accuracy in the prediction of memory and runtime with AUC above 0.9. In addition, the recurrent neural network exhibits high accuracy in predicting the next queries with a perplexity score of 25.

RQ2: What mechanisms exceed in terms of the cache hit ratio in the different benchmark scenarios?

The prefetching of the predicted queries (i.e., using RNN) and the prefetching of the cost-efficient queries (i.e., using both RNN and FFNN) outperform the traditional reactive caching mechanisms in all of our benchmark scenarios.

RQ3: What is the percentage of improvement of our framework over the traditional mechanisms?

Using our proactive caching framework, the percentage of improvement in the cache hit ratio ranges from 6% to 29%, on average, over the traditional reactive caching mechanisms (i.e., LRU, LFU, and LRFU).

Our main contributions can be described as follows:

(1) We present a proactive caching framework that combines the query cost estimation and the prefetching of future cached queries in the system.
(2) Our framework can be deployed as an independent in-memory middleware layer between any software and database as the queries are abstracted. Hence, it does not require any configuration or modification in the database system.
(3) The benchmark results exhibit improvements of our proactive caching mechanisms over the work of the traditional reactive caching mechanisms.

Paper Organization: The rest of the paper is organized as follows, we provide a background on FFNN and RNN in Section 2. We describe the data collection process in Section 3. We describe our framework in Section 4. We showcase the results in Section 5. We discuss the threats to validity in Section 6. We present the related work in Section 7. Finally, we conclude the paper in Section 8.

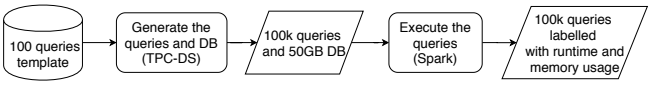


Figure 2: The data labeling process.

2 FFNN AND RNN BACKGROUND

We use the RNN and the FFNN to predict the upcoming queries and prefetch the cost-efficient queries.

The FFNN is the basis of supervised deep learning problems. FFNN is a form of the basic neural networks where the information flow from one layer to the next layer in a unidirectional manner, unlike recurrent neural networks where the information flow bidirectionally from next and previous layers [65]. The FFNN works by estimating a function f^*. The problem is defined as a classifier $y = f^*(x)$ that maps an input x to a class y. The FFNN defines a function $y = f(x, \theta)$, where it learns the parameter θ to approximate the value of $f^*(x)$. The input x defines the first layer, the function f represents the intermediate layers, and the output y defines the last layer. The information flows from the first to the last layer, where the parameter θ is evaluated at the output layer, and recalculated for the next $f^*(x)$ estimation [23].

The RNN is a descendant of the FFNN. The work of the RNN suits the problem of classifying a sequence of inputs to a sequence of outputs. The RNN maps a sequence of inputs $x...x_n$ to outputs $y...y_n$. Where the function $y = f^*(x)$ maps each x to y over the time-steps of $t_1...t_n$. To approximate the function f^*, the RNN defines the functions $y_t = f(x, \theta_t + \theta_{t-1})$, where the parameter θ is reevaluated at each time step to approximate the value of $f^*(x)$ [23, 47].

3 DATA COLLECTION

Our study is based on the TPC-DS database [49]. The database represents a data warehouse that revolves around online analytical processing tasks. TPC-DS database emulates a decision-support system of a retail product supplier with 100 defined queries that represent reporting jobs, which covers all the database tables. The database schema constitutes of 24 tables with an average of 18 columns per table, and 108 foreign keys, which signifies the complexity of the system.

The TPC-DS database employs benchmarking capabilities where it allows scaling and augmenting the database size and queries [56, 57]. Neural networks require data samples in the order of thousands to be trained [6, 67]. Hence, we augmented the number of original queries in the TPC-DS database to attain a sufficient number for the training and validation of our neural networks.

The augmentation of queries uses each of the 100 most used queries as *templates*. We augment to 1,000 queries from each template query by replacing the conditional operations that follow the WHERE clause in the query with random values. In the end, we generate a 50 GB SQL database, with 100 thousand queries augmented from the 100 most used queries in TPC-DS.

We showcase the labeling process in Figure 2. The queries need to be labeled by their execution time and memory consumption to train our FFNN model, as the FFNN serves as a cost estimator. The FFNN model predicts the runtime and memory consumption of a new incoming query to guide the caching decision. We labeled

Table 1: The data set description.

Data Set	# of queries	# of unique queries	Description
Full data	100,000	29,000	The whole number of generated queries.
Training data	70,000	23,000	Data used to train and validate the FFNN.
Benchmarking data	30,000	6,000	Data used to train the RNN and benchmark our approach.

the queries by loading the TPC-DS dataset to Spark [74]. Then, we execute the 100 thousand queries using Spark framework. Spark is an open-source parallel computing framework that allows importing large-scale databases and the parallelization of the execution of multiple queries in the system. This process allows an accurate memory and runtime recording in terms of milliseconds and bytes.

The initial data collection process allows us to establish a benchmark of 100 thousand queries that serve as the basis of our approach for training, testing and benchmarking our two neural networks in different scenarios. As shown in Table 1, the 100 thousand queries were cut into 70% to train and validate the FFNN and 30% to train the RNN. The same 30% of the queries were also used to benchmark our approach.

4 OVERVIEW OF OUR FRAMEWORK

Figure 3 shows an overview of our approach. For each incoming query in the system, first, our framework *generates the query embeddings* that converts the input query text to an embedding vector. Second, our framework *predicts the next upcoming five queries* using the RNN. We chose the number five since the RNN is able to correctly predict the next five queries with high accuracy of 95%, on average, in our hyperparameter tuning experiments. The accuracy in predicting more than five upcoming queries drops by 15% when that number is incremented by five progressively (i.e., next 10, next 15, etc.). We discuss that process further in Section 6. Third, the FFNN *predicts the cost* (i.e., the runtime and memory consumption) of the upcoming queries. Finally, the framework *prefetches the cost-efficient queries* among the upcoming ones. Our framework evicts queries based on any reactive caching algorithm (e.g., LRU, LFU, and LRFU) when the cache size would not fit the upcoming queries. In the next sections, we describe the steps for generating query embeddings, the architecture of the used neural networks (i.e., RNN and FFNN), and the benchmark process.

4.1 Generate Word Embeddings for Queries

The FFNN processes the *text embeddings* of queries as an input. The FFNN is trained to estimate the memory consumption and runtime of the queries. Transforming the query text to query embeddings inherently guards the meaning of the operations of the query which makes the cost estimation achievable.

To extract embeddings for each query, we employ Word2Vec [21] to transform each word in the query text to a 64 bits array. The Word2Vec algorithm works by reducing each word in the input (i.e., corpus) into a unique vector. The vectors are positioned in a hyperdimensional space where words that share common contexts are positioned close to each other [60]. To represent the overall embedding of a query, the vectors for each word in a query text are summed and normalized using L2 normalization [70, 72]. To validate the correctness of the final query embeddings, we use the cosine similarity metric [53]. The cosine similarity measures the angle between two vectors projected in a hyperdimensional plane where a value of 1 signifies a total overlay of the two vectors meaning that the vectors are identical [69].

To verify the contextual similarity of the generated embeddings that are derived from the same template, we measure the cosine similarity among the generated 1,000 queries of each template of the 100 TPC-DS query templates. The mean cosine similarity varies from 0.85 to 0.92 in the 100 query templates, which proves the correctness of our embedding extraction.

The RNN extracts the patterns from a sequence of executed queries. Hence, RNN needs to represent every query as a unique value in the input sequence of queries [7]. To represent the queries as unique values, we hashed the whole query text into 256 bits hash using SHA256 [58]. The algorithm secures collision resistance giving a unique hash for each text [33], thus securing unique words for each unique query.

4.2 Architecture of Neural Networks

The Feed-Forward Neural Network. The FFNN [65] is used to estimate the cost of executing a query. It consists of two hidden layers with rectified linear units (ReLU) [48]. The input layer of the neural network is of size 64 bits and accepts the aforementioned extracted embeddings of the queries. The neural network has two outputs layers: the first layer is used for predicting the memory consumption of the query, and the second layer is used to predict the execution time of the query. The two outputs employ a Softmax activation function. Softmax serves a multi-class probability prediction at the last layer of the neural network by normalizing the outputs by the sum of their exponents to represent them as a probability distribution [22].

To adhere to the usage of Softmax as output layers, we quantized the runtime and memory consumption. Quantization serves in increasing the accuracy of predictions in neural networks by predicting the quantiles (i.e., the classes) instead of predicting a distribution of real numbers [18]. We quantize the runtime and memory distributions to three classes (e.g., low, medium, and high). The first class represents the quantile from 0% to 33% that is the low-level memory consumption or runtime, the second class describes the 34% to 66% quantile that is the medium-level memory consumption or runtime, and the last class from 67% to 100% quantile represents the high-level runtime or memory consumption. This process avoids predicting the exact runtimes or memory consumption to a classification task that predicts ranges of runtime or memory consumption described as low, medium, and high.

The FFNN is trained on 70 thousand of the 100 thousand generated queries (i.e., training and validation dataset) as shown in Table 1. We guarantee the uniqueness in the training dataset to eliminate overfitting in the neural network that may occur if the same data appears in the training and the validation process. Hence, we train and validate the FFNN on the 23 thousand unique queries

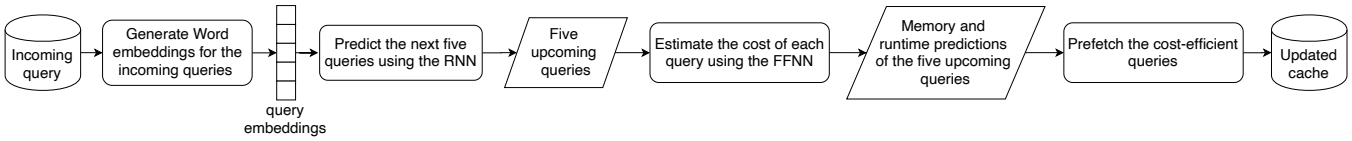


Figure 3: An overview of our query caching framework.

of the training dataset after filtering the 70 thousand queries from duplicates.

This data (i.e., the 23,000 queries) is split into 80% training and 20% validation split using 10 folds cross-validation. Each fold is split into the same percentage (80%-20%) as we trained and validated the FFNN.

The Recurrent Neural Network. The neural network consists of two long short term memory (LSTM) layers [19] with 10 units per layer. The LSTM layer serves the objective of predicting future queries at each time step. The time steps are the index of the query in the sentence of queries. The LSTM is widely used to process a sequence of data as it solves the vanishing gradient problem when the sequence of data lingers in its length [64].

As shown in Table 1, we train the RNN on the 30 thousand queries that form the benchmark dataset. The vocabulary of the dataset consists of 6 thousand unique queries after filtering the 30 thousand queries from duplicates. Hence, the output layer of the RNN consists of a Softmax layer with 6 thousand classes that represents the vocabulary of our corpus (i.e., the number of unique queries in the benchmark dataset). The Softmax layer serves as a probability prediction of the most suitable word (i.e., query) to occur at each time step. We take the highest five probabilities to predict the upcoming five queries from the previous queries.

4.3 Generating The Different Datasets

The datasets for training the RNN is formed by hashing the query texts and generating the order of queries under three different scenarios. Each scenario represents a different plausible real-life occurrence of queries. The RNNs identify the occurrences of queries to form patterns that lead to the prediction of upcoming queries. The occurrences are represented by the repetition of the same hash.

Sequence Dataset. The sequence dataset represents the queries that occur sequentially in real-life scenarios. For example, some queries might be part of a task where they are always executed consecutively. This could be part of a reporting job where different queries on different tables are executed in sequence to extract the data. To generate the sequence dataset, we choose a random query from each of the 100 templates in our benchmark dataset sequentially until we reach all the queries in the benchmark dataset.

Batch Dataset. The batch dataset represents a scenario where the queries might co-occur as a batch of jobs. Similarly to the sequential scenario, the batch represents jobs where a group of queries is repeated in a sequence. The generation of the batch dataset is similar to the sequence dataset where the only difference is instead of choosing one query of each template to run sequentially, we chose n numbers of queries of each template from the benchmark dataset randomly to run sequentially.

Random Dataset. The random dataset is formed by shuffling the whole benchmark dataset. We chose random scenario as a stress test for our framework. The sequential and batch histories will guarantee patterns that will enhance the work of the prefetching mechanism (i.e., the RNN). In contrast, we test our prefetching mechanism when the scenario is formed by a random occurrence of queries. An effective prefetching mechanism should not suffer from a major degradation in performance in this scenario.

4.4 The Benchmark Process

The benchmark process relies on three different workloads that are generated similarly to the datasets for training the RNN. To simulate the work of the RNN and the FFNN in practice, we generate from the benchmark dataset another sequential, batch, and random datasets. The benchmark workloads were also generated from the same data that was used to train the RNN to guarantee that the same vocabulary (i.e., corpus for the RNN) is consistent. But the generation guarantees a different order of occurrences of queries. Hence, the training and the benchmark data for the RNN are different.

We evaluate the results on each generated benchmark dataset on its own comparing the cache hit ratio of the proactive caching mechanisms (e.g., the RNN, and the combination of the RNN with the FFNN) and the reactive caching mechanisms (e.g., LRU, LFU, and LRFU). The execution time in the system should be reduced with the presence of the same incoming query in the cache, while the cache hit ratio increases when the same incoming query is present in the cache. Thus, the cost estimation and the prefetching mechanisms are compared against the traditional mechanisms LRU, LFU, and LRFU. The cache hit ratio is recorded gradually while we process the queries from each generated benchmark dataset into the system and either execute them or reuse the queries from the cache if they are present as shown in Figure 4.

4.5 Caching Decision for the Different Mechanisms

As shown in Figure 4, we compared each of our mechanisms (i.e., the RNN plus the combination of the RNN and the FFNN) against the traditional reactive caching mechanisms (i.e., LRU, LFU, and LRFU) to validate the performance of each our mechanisms.

The **RNN** predicts for each incoming query the next five incoming queries and caches them. The prefetching mechanism relies on the patterns of occurrences of queries learned from the training process.

We combined the recurrent **neural network and the FFNN (RNN-FFNN)** to prefetch and cache expensive queries only. After predicting the next five upcoming queries, each query would fall under the prediction of the FFNN to estimate the runtime and memory

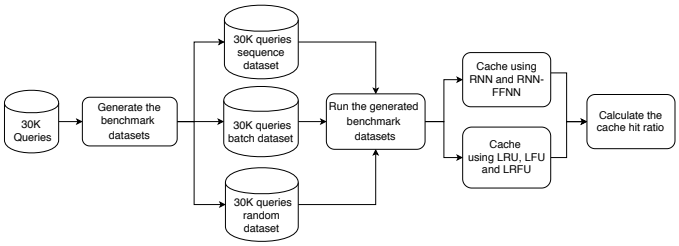


Figure 4: The benchmark process.

consumption. The mechanism prefetches and caches the queries with medium or high runtime and low memory consumption when the cache is 70% full. The condition is relaxed when the cache does not reach that threshold to cache any incoming query. We have tested this mechanism under different thresholds and found that 70% is the best suitable threshold under different cache sizes. The low, medium and high predictions are the results of the quantization of the real values as described in the data processing to three classes or quantiles.

The two caching mechanisms that serve the prefetching (RNN), and the cost estimation and prefetching combined (RNN-FFNN) would guide the decision for caching the queries, but their work is incomplete with the absence of caching eviction mechanisms. Hence, we implemented the work of LRU, LFU, and LRFU to be employed as cache eviction policies, either on their own or with the combination of our proactive caching mechanisms (RNN, and RNN-FFNN).

We benchmark our results and compare the usage of each traditional mechanism with its modified version respectively (e.g., comparing LRU with the RNN that uses LRU as an eviction mechanism).

5 RESULTS

In this section, we present the motivation, the approach, and the results of the studied research questions.

5.1 RQ1: How accurate are the cost estimation and the prefetching functions?

Motivation: Before employing our two neural networks (FFNN and RNN), we need to evaluate the accuracy of our work in cost estimation and the prediction of the next queries. Hence, we can ensure the correctness of the predictions of our two neural networks to employ them in prefetching the cost-efficient queries.

Approach: To evaluate the FFNN we rely on the area under the curve (AUC) metric [24, 25]. Our model outputs probability predictions for runtime and memory consumption. AUC tests the fit of our model in probability predictions with different thresholds. A model with an AUC value of 0.5 signifies a random prediction model. A model with an AUC value of 1 signifies a model with perfect true positives and true negatives predictions, while a model with an AUC value of 0 signifies a model with perfect inverse probability predictions [51].

We train our model on 10 folds cross-validation, thus the AUC is calculated as the average AUC of the 10 folds on the validation sets [45]. The 10 fold cross-validation reshuffles the data and retrains the network from scratch on each fold. Hence, the average AUC demonstrates the effectiveness of our model in different data distributions.

To ensure that the model is not showing bias in the result we enforced weight class distribution in the training sets. The weights added to the cross-entropy loss function (i.e., the neural network loss function) ensure that the loss function is weighted to overcome the biases in predicting one certain class since it is more dominant [35, 73]. The average class weight across the 10 sets for the runtime is 1.03 for class 0, 1.01 for class 1, and 0.95 of class 2. (The classes 0 to 2 represent the low, medium, and high classes respectively). The average class weight for the memory is 0.80 for class 0, 1.92 for class 1, and 0.81 for class 2.

The RNN is evaluated based on the perplexity score of the predictions. Given a sequence of input queries, the RNN predicts the next query to occur. The RNN model uses a sequence loss function (e.g., cross-entropy function) that predicts the most suitable query from the vocabulary of queries at each time step. The model can then be evaluated as a sequence to one prediction error by using the perplexity metric. The perplexity measures the fit of the query distributions on unseen data. The measured value represents the inverse of the prediction probability of what is the next query to occur. The perplexity is an unbounded function that can span to infinity [30].

Results: We recorded the AUC for the runtime classification and the AUC for the memory consumption classification. The runtime AUC is 0.94 and the memory consumption AUC is 0.92. The AUC function is resilient for varying class distributions, so the result does not show biases [29]. The AUC was recorded as the average of the 10 folds cross-validation on the validation sets. The training and testing of the FFNN are produced on the training dataset, as described in Section 3.

The RNN achieves an average perplexity score of 25 on the three benchmark datasets. The lowest perplexity measure is 1, it means that the fit is perfect, while a good language model should have a perplexity lower than 200 [9].

Summary of RQ1

The FFNN exhibits high accuracy in the prediction of memory and runtime with AUC above 0.9. In addition, the RNN exhibits high accuracy in predicting the next queries with a perplexity score of 25.

5.2 RQ2: What mechanisms exceed in terms of the cache hit ratio in the different benchmark scenarios?

Motivation: In the previous RQ, we observe that our approach can accurately predict the upcoming queries. In this RQ, we aim to understand whether our proactive caching mechanism outperforms the existing reactive caching mechanisms (i.e., LRU, LFU, and LRFU). Understanding the optimum caching mechanism can help system owners better select suitable caching mechanisms to improve the performance of the software systems.

Table 2: The cache hit ratio of the sequence scenario with different caching mechanisms and varying cache sizes.

Mechanisms	Small cache	Medium cache	Large cache	Normal cache
LRU	0.12	0.41	0.62	0.37
LFU	0.17	0.48	0.66	0.43
LRFU	0.12	0.41	0.62	0.38
RNN + LRU	**0.34**	**0.54**	0.68	**0.51**
RNN + LFU	0.29	**0.54**	0.68	0.50
RNN + LRFU	**0.34**	**0.54**	0.68	**0.51**
RNN-FFNN + LRU	0.32	**0.54**	**0.69**	**0.51**
RNN-FFNN + LFU	0.29	**0.54**	**0.69**	**0.51**
RNN-FFNN + LRFU	0.32	0.53	0.68	**0.51**

Table 3: The cache hit ratio of the batch scenario with different caching mechanisms and varying cache sizes.

Mechanisms	Small cache	Medium cache	Large cache	Normal cache
LRU	0.14	0.41	0.62	0.39
LFU	0.23	0.49	0.66	0.46
LRFU	0.14	0.41	0.63	0.39
RNN + LRU	0.27	0.49	0.69	0.48
RNN + LFU	**0.37**	**0.57**	0.69	**0.54**
RNN + LRFU	0.27	0.50	0.69	0.48
RNN-FFNN + LRU	0.17	0.39	0.67	0.41
RNN-FFNN + LFU	0.28	0.51	**0.70**	0.52
RNN-FFNN + LRFU	0.18	0.40	0.68	0.41

Approach: The performance of query caching mechanisms can be impacted by the cache size. Hence, we evaluate the performance of the studied caching mechanisms using different cache sizes (i.e., small, medium, large, and the normal cache size). We design the different cache sizes as follows. First, we set up the *cache limit*. The cache limit represents a relative portion of the workload data that can be stored in the memory. Caching the full data of all users in a large-scale system can be very expensive and practically infeasible [54]. Hence, we choose 30% (i.e., 2,000 queries) of the number of unique queries in our benchmark dataset as the cache limit.

Then, given a certain cache limit, we define the used cache sizes as follows. The *small cache size* is calculated by multiplying the cache limit (i.e., 2,000 queries) with the minimum memory size of a query. The *medium cache size* is calculated by multiplying the cache limit with the median memory size of a query. The *large cache size* is calculated by multiplying the cache limit with the maximum memory size of a query. Finally, the *normal cache size* is calculated by multiplying the cache limit with the average memory size of a query. For every cache size, we recorded the cache hit ratio using three different benchmark datasets (i.e., sequence, batch, and random).

We benchmark two proactive mechanisms (1) fetching the upcoming queries (i.e., using RNN) and (2) fetching the cost-efficient queries (i.e., using RNN and FFNN named as *RNN-FFNN*). In addition, we benchmark three reactive approaches LRU, LFU, and LRFU. As described in Section 4, proactive caching can use different eviction mechanisms. Hence, we benchmark every proactive caching mechanism (i.e., RNN and RNN-FFNN) with the three reactive approaches LRU, LFU, and LRFU. We represent the RNN that uses the LFU cache eviction mechanism as *"RNN + LFU"*, the RNN that uses LRU as *"RNN + LRU"*, and the RNN that uses LRFU as *"RNN + LRFU"*. Similarly, we represent the RNN-FFNN that uses the LRU cache eviction mechanism as *"RNN-FFNN + LRU"*, the RNN-FFNN that uses LFU as *"RNN-FFNN + LFU"*, and the RNN-FFNN that uses LRFU as *"RNN-FFNN + LRFU"*.

Results: Benchmarking the Sequence Scenario. As shown in Table 2, the RNN-FFNN and the RNN exhibit the best results with the normal cache size. The cache hit ratio ranges from 0.50 to 0.51 when using RNN or RNN-FFNN combined with any reactive caching mechanism including LRU, LFU, and LRFU.

As shown in Table 2, in the small cache sizes, the RNN + LRU and the RNN + LRFU perform the best. In the medium and large cache sizes, adding the notion of cost estimation ameliorates the performance of prefetching. RNN-FFNN on top of the two reactive caching mechanisms LRU and LFU exhibits the best cache hit ratios of 0.54 and 0.69 for medium and large cache sizes respectively.

In summary, the RNN and the RNN-FFNN perform the best. In the small cache sizes, the RNN performs better than the combination of RNN and FFNN. The RNN-FFNN performs the best on medium and large cache sizes.

Prefetching all the co-occurrent queries would save time more than prefetching the expensive ones only. This is due to some queries that might occur frequently but are not expensive. In large cache sizes, prefetching more co-occurrent queries would cause thrashing, thus leading to caching unnecessary queries. The cost estimation function (i.e., using FFNN) would tune the work of the prefetching mechanism as it combines the expensive and the co-occurrent queries.

Summary of benchmarking the sequence scenario

The prefetching function solely (i.e., RNN) or combined with the cost estimation function (i.e., RNN-FFNN) exhibits the best results in terms of the cache hit ratio in the sequence scenario.

Results: Benchmarking the Batch Scenario. The cache hit ratio results summarized in Table 3 indicates that the prefetching mechanism using the RNN + LFU performs the best with a 0.54 cache hit ratio on normal cache size. The RNN-FFNN + LFU comes in second place with a cache hit ratio of 0.52 on normal cache size. The cache hit ratio for the RNN + LFU outperforms all the other mechanisms on all the cache sizes except on the large cache size, where RNN-FFNN + LFU performs slightly better.

In summary, for the batch scenario, the RNN + LFU performs the best. In smaller and medium cache sizes, the RNN solely performs better than the combination of RNN and FFNN. This is due to the similar reasons in the sequence scenario that with bigger cache capacities caching more content can cause thrashing. Hence, the usage of FFNN to estimate the expensiveness and cache the expensive queries would tune the performance of the system on larger cache sizes.

Table 4: The cache hit ratio of the random scenario with different caching mechanisms and varying cache sizes.

Mechanisms	Small cache	Medium cache	Large cache	Normal cache
LRU	0.13	0.41	0.62	0.38
LFU	0.18	0.46	0.66	0.43
LRFU	0.13	0.41	0.63	0.38
RNN + LRU	0.15	0.46	0.66	0.41
RNN + LFU	**0.19**	**0.51**	0.67	**0.45**
RNN + LRFU	0.15	0.46	0.66	0.42
RNN-FFNN + LRU	0.14	0.45	0.66	0.41
RNN-FFNN + LFU	**0.19**	**0.51**	**0.68**	**0.45**
RNN-FFNN + LRFU	0.18	0.45	0.66	0.41

Summary of benchmarking the batch scenario

The prefetching function using RNN solely exhibits the best results in terms of the cache hit ratio. The combination of the prefetching and the cost estimation functions (i.e., using the RNN-FFNN) comes in second place in the batch scenario.

Results: Benchmarking the Random Scenario. Following the cache hit ratio results in Table 4, the combination of the RNN-FFNN over LFU reaches 0.45 cache hit ratio, as well as the RNN solely.

As listed in Table 4, in small cache sizes, the RNN and the combination of the RNN-FFNN over LFU perform the best with a cache hit ratio of 0.19. For the medium cache sizes, the RNN and the RNN-FFNN over LFU show the best performance with a cache hit ratio of 0.51. Finally, for the large cache sizes, the RNN-FFNN over LFU performs the best with a cache hit ratio of 0.68.

On average, the two mechanisms (RNN or RNN-FFN) over LFU perform the best. In a random scenario, the prefetching mechanism suffers from degradation compared with the other scenarios. This is due to the absence of recognizable patterns for the recurrent network to learn. However, adding the notion of prefetching still improves the performance over the reactive caching mechanisms. In addition, adding the notion of cost estimation helps to improve the prefetching in larger cache sizes.

Summary of benchmarking the random scenario

The prefetching function (RNN) or the combination of the prefetching and the cost estimation functions (i.e., using the RNN-FFNN) on top of LFU outperforms the reactive caching mechanisms (e.g., LRU, LFU, and LRFU) in the random scenario.

5.3 RQ3: What is the percentage of improvement of our framework over the traditional mechanisms?

Motivation: In this RQ, we study the percentage of improvement of our framework over the reactive caching mechanisms in the three benchmark scenarios. This can help demonstrate the added benefits in using our proactive mechanism on top of the existing reactive caching mechanisms.

Table 5: The percentage of improvement in cache hit ratio of our framework over the traditional mechanisms.

Baseline mechanism	% of improvement		
	Sequence	Batch	Random
LRU	35.4%	5.4%	8.7%
LFU	17.3%	7.3%	8.3%
LRFU	34.7%	6.1%	5.6%
Average	29.1%	6.3%	7.5%

Approach: The percentage of improvement is calculated as the percentage of the increase in the cache hit ratio of one mechanism over another. Table 5 shows the percentage of improvement of our framework (i.e., the RNN-FFNN) over the LRU, the LFU, and the LRFU mechanisms.

Results: Our proactive caching framework (i.e., the RNN-FFNN) improves LRU from 5.4% to 35.4%. The RNN-FFNN improves LFU from 7.3% to 17.3%. In addition, the RNN-FFNN improves LRFU from 5.6% to 34.7%. The highest percentages of improvements are in the sequence scenario that is 29% on average. The sequence scenario represents the occurrences of queries in a well-defined pattern where the work of the prefetching (i.e., RNN) would excel.

In the batch scenario, our framework improves over the traditional mechanisms by 6% on average. The improvements are lower than the improvements of our framework in the sequence scenario since the traditional mechanisms (LRU, LFU, and LRFU) use a greedy approach to cache that fits the nature of the batch scenario. The batch scenario could fit more repetitions of the same query in a shorter time span than the sequence scenario where the greedy caching attains better results.

For the random scenario, the improvement of our framework over the traditional mechanisms is of 7% on average as it is hard to predict the right upcoming queries in a scenario with no clear patterns.

Summary of RQ3

Using our proactive caching framework, the percentage of improvement in the cache hit ratio ranges from 6% to 29%, on average, over the traditional reactive caching mechanisms (i.e., LRU, LFU, and LRFU).

6 THREATS TO VALIDITY

This section addresses the threats to the validity of our approach as follows. First, our approach estimates the cost of queries by feeding the FFNN with the query represented as a word embedding. This is a generalized approach that can be employed in any database system. However, our approach has a drawback as the generated embeddings may differ when the same query is written using aliases or views (i.e., when queries written in a different syntax).

Second, the benchmark is inducted on three generated datasets. We try to simulate the workloads in a system, whether in a sequential, a batch, or a random manner. This simulation grasps the real-life occurrences of queries to a certain degree, but it is not fully

accurate. In a perfect scenario, we would have based our study on a recorded history of queries, but we could not find any available rich queries history.

Third, to train the FFNN, we labeled the queries by executing them on the Spark framework. The process is strenuous and requires approximately a week to complete the execution of all the queries. Alternatively, if there exists a system that collects the history of the executed queries with their respective runtime and memory consumption, our work can be replicated easily.

Fourth, the prefetching mechanism that uses the RNN predicts the five upcoming queries. The upcoming queries are either cached using the prefetching mechanism solely or fed to the FFNN to prefetch and cache the cost-effective queries. We chose number five as a proof of concept in our work. However, the number of predicted queries could be tuned based on multiple factors, such as the repetitiveness of the executed queries. Further studies can extend our work by optimizing the number of predicted queries based on the nature of the query execution scenario.

Finally, our mechanisms assume that the history is fixed and no unidentifiable incoming queries would occur in the system. That is valid for benchmarking our prefetching and cost estimation approaches. But in real-life scenarios, the two mechanisms should be monitored as with the increasing number of unidentifiable queries the cost estimation and the prefetching might suffer from more errors. Hence, we can monitor the queries in history periodically and find a threshold where the number of new unidentifiable queries would cause a clear degradation in the performance of our mechanisms (tested by the cache hit ratio) that would drive us to retrain our two neural networks.

7 RELATED WORK

The various work in query cache is divided into two main strategies (1) reactive caching and (2) proactive caching. In this section, we discuss the main query caching techniques.

7.1 Reactive Caching

The work on reactive caching focuses on the eviction and replacement mechanisms that are concerned with identifying the stale queries in the cache to be evicted. The eviction relieves the memory overhead in the cache as it frees space for more beneficial queries to be cached. The eviction mechanisms define protocols based on recency and frequency to detect the stale queries in the cache.

Lange et al. [36] introduce the Least Recently Used (LRU) cache eviction mechanism in their work on the CPUs' registers caching. The algorithm maintains an array of timestamps for each register and the eviction happens by removing the data in the register with the lowest timestamp. That simple idea was then reintegrated as a generic caching algorithm for different fields, such as mobile network and database systems [32, 46, 68].

Matick et al. [44] introduce Least Frequently Used (LFU); another caching eviction mechanism following their work on caching in CPUs' registers. The algorithm stores access counts for each register and evicts the data with the least count. This simple approach proved to be efficient and is considered on par with LRU, as frequency and recency are the two most influential factors for identifying stale cache. This mechanism also spanned to be reused in databases and networks [15, 28].

Lee et al. [37] combine the work of LRU and LFU to create the Least Recently Frequently Used (LRFU) mechanism. The comparison against the two algorithms showed that LRFU outperforms the other two in a spectrum of cache capacities. The combination of the recency and frequency is more effective for larger cache sizes.

In-memory database systems such as Redis [41] or Apache Ignite [3, 77] are widely used nowadays for the fast access and data caching/manipulation. Redis and Ignite are often employed as a middleware layer that helps fast access to hot data to eliminate the necessity of data access from the main traditional relational database [34, 75]. Redis and Ignite employ LRU and LFU specifically as cache eviction mechanisms due to their fast reactive nature of eliminating stale cached queries [1, 2, 63].

Hon et al. [27] employ a dynamic web caching protocol. Web pages are stored as HTML data in a cache placed as an intermediate layer between the web server and the client. The protocol uses a synchronization daemon that invalidates and evicts the cached pages that are out of sync with the server.

Different from the aforementioned reactive caching studies, our approach aims to predict queries that need to be cached and proactively cache the cost-efficient queries.

7.2 Proactive Caching

The work in proactive caching revolves mainly on identifying the beneficial data to cache in the system. The main objective is to cache the content that maximizes the reuse of the cache and avoid thrashing in the system. The work in proactive caching branches from studying the popularity of data [11, 39, 40, 66], the data patterns [11, 50, 76, 78], to the structure of the data [62], which eventually leads to maximizing the cache benefits.

Caching Based on Content Popularity. Luo et al. [40] develop a framework for caching the most popular queries in the system. The system follows statistical measures to consider the set of the most used queries that are prioritized for caching.

Liu et al. [39] implement a deep learning approach to cache popular data in ICN networks. The neural network is adaptive and retrainable depending on the network. The popularity prediction is transformed into a discretized prediction problem where multiple classes are predicted. This approach added with the cache replacement scheme LRU showed a 15% to 40% improvement over the LRU mechanism.

Tanzil et al. [66] introduce a proactive predictive approach to cache popular YouTube content in cellular networks. The approach uses a neural network to predict a 10 class popularity, where the content with a higher class inflicts a more popular content. Their work implements a segmented LRU algorithm for cache eviction.

Caching Based on Data Patterns. Chan et al. [11] describe a generic predictive algorithm based on a recurrence based probabilistic method to cache and prefetch the content with high hit rates at the Wireless Edge. The approach is compared with other predictive methods and shows on average a 12.5% improvement over LRU and LFU and a 5% improvement over other predictive methods.

Zeydan et al. [76] work on caching in big data mobile networks (e.g., 5G networks). The work tackles caching the clients' information at the network edge using contextual information such as the browsing history and the spatio-temporal information.

Zhong et al. [78] introduce a Deep Reinforcement Learning approach that takes into consideration the recency and frequency factors to create an adaptive cache eviction mechanism. Their model was trained on network content and compared with the work of LRU and LFU. The reinforcement learning approach shows an improvement of 15% on average over LRU and LFU work over the varying cache capacities.

Caching Based on Data Structure. Shim et al. [62] consider in their work the partial reuse of queries consisting of a single JOIN operation. Their work aims to identify the expensive subqueries to be cached based on a dynamic query cost function that combines the execution time, the memory requirements, and the frequency of usage of the subquery. Their system also evicts from the cache the stale content using the same cost function when newer queries with higher cost need to be cached.

The aforementioned studies use different machine learning techniques to manage the cache based on the predicted popularity or the expected usage patterns of the cached content. Inspired by the existing work, we use machine learning techniques (i.e., RNN) to predict the upcoming queries. Then, our approach prefetches the cost-efficient queries using the FFNN.

8 CONCLUSION

Query caching is an essential technique to ameliorate the performance of a database system. The work on query caching spans from the decision of queries to cache by estimating the execution time savings or by greedy caching mechanisms that focus on the eviction and replacement policies of the cache.

In this work, we combine the two concepts of cache decision and cache eviction to develop a proactive caching framework. Our framework employs cost estimation and prefetching mechanisms combined with a reactive cache eviction policy. We compare our work to the previous cache replacement policies that are employed in current systems. From our experiments, we observe remarkable improvements of 6% to 29% by using our framework comparing with the existing work in terms of the cache hit ratio.

Our work was benchmarked in three scenarios that represent the different workloads in real life, whether queries occurring in sequence, batch, or random. The recursive neural network (RNN) that serves the prefetching solely or with the combination of the feed-forward neural network (FFNN) that estimates the cost of the queries exceeded in performance all the other caching mechanisms (i.e., LRU, LFU, and LRFU).

Moreover, our work is not specific to a particular database system and can be employed for any database system. Our framework can also be used on an intermediate level between the client and the database system to form an in-memory middleware layer that deals with caching the queries as a whole. In the future, we plan to test our approach on real data captured from real-world usage scenarios.

REFERENCES

[1] 2009. Redis cache eviction policies. https://docs.redislabs.com/latest/rs/administering/database-operations/eviction-policy/. Accessed: 2019-12-01.

[2] 2015. Apache Ignite cache eviction policies. https://apacheignite.readme.io/docs/evictions. Accessed: 2019-12-01.

[3] Sujoy Acharya. 2018. Apache Ignite Quick Start Guide: Distributed data caching and processing made easy. Packt Publishing Ltd.

[4] Sibel Adali, K Selçuk Candan, Yannis Papakonstantinou, and VS Subrahmanian. 1996. Query caching and optimization in distributed mediator systems. In ACM SIGMOD Record, Vol. 25. ACM, 137–146.

[5] Awny K Al-omari, Tom C Reyes, and Robert Wehrmeister. 2010. Hybrid database query caching. US Patent 7,743,053.

[6] Ahmad Alwosheel, Sander van Cranenburgh, and Caspar G Chorus. 2018. Is your dataset big enough? sample size requirements when using artificial neural networks for discrete choice analysis. Journal of choice modelling 28 (2018), 167–182.

[7] Garen Arevian. 2007. Recurrent neural networks for robust real-world text classification. In IEEE/WIC/ACM International Conference on Web Intelligence (WI'07). IEEE, 326–329.

[8] Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, and Riccardo Torlone. 1999. Database systems: concepts, languages & architectures. Vol. 1. McGraw-Hill London.

[9] Leif Azzopardi, Mark Girolami, and Keith Van Rijsbergen. 2003. Investigating the relationship between language model perplexity and IR precision-recall measures. (2003).

[10] Matthew Broadbent, Daniel King, Sean Baildon, Nektarios Georgalas, and Nicholas Race. 2015. OpenCache: A software-defined content caching platform. In Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft). IEEE, 1–5.

[11] Chien Aun Chan, Ming Yan, Andre F Gygax, Wenwen Li, Li Li, I Chih-Lin, Jinyao Yan, and Christopher Leckie. 2019. Big Data Driven Predictive Caching at the Wireless Edge. In 2019 IEEE International Conference on Communications Workshops (ICC Workshops). IEEE, 1–6.

[12] Zheng Chang, Lei Lei, Zhenyu Zhou, Shiwen Mao, and Tapani Ristaniemi. 2018. Learn to cache: Machine learning for network edge caching in the big data era. IEEE Wireless Communications 25, 3 (2018), 28–35.

[13] Surajit Chaudhuri, Vivek Narasayya, and Ravishankar Ramamurthy. 2004. Estimating progress of execution for SQL queries. In Proceedings of the 2004 ACM SIGMOD international conference on Management of data. ACM, 803–814.

[14] Arthur F Cochcroft Jr. 1998. Method and apparatus for detecting thrashing in a cache memory. US Patent 5,752,261.

[15] Mieso K Denko and Jun Tian. 2006. Cooperative caching with adaptive prefetching in mobile ad hoc networks. In 2006 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications. IEEE, 38–44.

[16] KR Dittrich, AM Kotz, and JA Mülle. 1985. A multilevel approach to design database systems and its basic mechanisms. In Proc. IEEE COMPINT, Montreal, Vol. 183.

[17] Klaus R Dittrich, Willi Gotthard, and Peter C Lockemann. 1987. DAMOKLES—a database system for software engineering environments. In Advanced programming environments. Springer, 353–371.

[18] Gunhan Dundar and Kenneth Rose. 1995. The effects of quantization on multilayer neural networks. IEEE Transactions on Neural Networks 6, 6 (1995), 1446–1451.

[19] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. 1999. Learning to forget: Continual prediction with LSTM. (1999).

[20] Parke Godfrey and Jarek Gryz. 1997. Semantic Query Caching for Hetereogeneous Databases.. In KRDB. 6–1.

[21] Yoav Goldberg and Omer Levy. 2014. word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. arXiv preprint arXiv:1402.3722 (2014).

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. 6.2. 2.3 softmax units for multinoulli output distributions. In Deep Learning. MIT Press, 180–184.

[23] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. Deep Learning. MIT Press. http://www.deeplearningbook.org.

[24] David J Hand and Robert J Till. 2001. A simple generalisation of the area under the ROC curve for multiple class classification problems. Machine learning 45, 2 (2001), 171–186.

[25] James A Hanley and Barbara J McNeil. 1982. The meaning and use of the area under a receiver operating characteristic (ROC) curve. Radiology 143, 1 (1982), 29–36.

[26] Olaf Hartig and Ralf Heese. 2007. The SPARQL query graph model for query optimization. In European Semantic Web Conference. Springer, 564–578.

[27] Lenny K Hon, Leon Kuperman, Louis S Mau, and Alexander Mohelsky. 2001. Caching dynamic web pages. US Patent 6,185,608.

[28] Jian hua Ran, Na Lv, Ding Zhang, Yuan yuan Ma, and Zhen yong Xie. 2013. On performance of cache policies in named data networking. In 2013 International Conference on Advanced Computer Science and Electronics Information (ICACSEI 2013). Atlantis Press.

[29] Jin Huang and Charles X Ling. 2005. Using AUC and accuracy in evaluating learning algorithms. IEEE Transactions on knowledge and Data Engineering 17, 3 (2005), 299–310.

[30] Frederick Jelinek, Bernard Merialdo, Salim Roukos, and Martin Strauss. 1991. A dynamic language model for speech recognition. In Speech and Natural Language: Proceedings of a Workshop Held at Pacific Grove, California, February 19-22, 1991.

[31] Navin Kabra and David J DeWitt. 1998. Efficient mid-query re-optimization of sub-optimal query execution plans. In ACM SIGMOD Record, Vol. 27. ACM, 106–117.

[32] Harshad N Kamat and David J Clarke. 2012. Email server with enhanced least recently used (LRU) cache. US Patent 8,307,036.

[33] Dmitry Khovratovich, Christian Rechberger, and Alexandra Savelieva. 2012. Bicliques for preimages: attacks on Skein-512 and the SHA-2 family. In International Workshop on Fast Software Encryption. Springer, 244–263.

[34] Doyoung Kim, Won Gi Choi, Hanseung Sung, and Sanghyun Park. 2019. A scalable and persistent key-value store using non-volatile memory. In Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing. ACM, 464–467.

[35] Gary King and Langche Zeng. 2001. Logistic regression in rare events data. Political analysis 9, 2 (2001), 137–163.

[36] Ronald E Lange and Richard J Fisher. 1982. Cache memory utilizing selective clearing and least recently used updating. US Patent 4,322,795.

[37] Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H Noh, Sang Lyul Min, Yookun Cho, and Chong Sang Kim. 2001. LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies. IEEE transactions on Computers 12 (2001), 1352–1361.

[38] Ken CK Lee, Hong Va Leong, and Antonio Si. 1999. Semantic query caching in a mobile environment. ACM SIGMOBILE Mobile Computing and Communications Review 3, 2 (1999), 28–36.

[39] Wai-Xi Liu, Jie Zhang, Zhong-Wei Liang, Ling-Xi Peng, and Jun Cai. 2017. Content popularity prediction and caching for ICN: A deep learning approach with SDN. IEEE access 6 (2017), 5075–5089.

[40] Qiong Luo, Jeffrey F Naughton, Rajasekar Krishnamurthy, Pei Cao, and Yunrui Li. 2000. Active query caching for database web servers. In International Workshop on the World Wide Web and Databases. Springer, 92–104.

[41] Tiago Macedo and Fred Oliveira. 2011. Redis Cookbook: Practical Techniques for Fast Data Manipulation. " O'Reilly Media, Inc.".

[42] Bhushan Mandhani and Dan Suciu. 2005. Query caching and view selection for XML databases. In Proceedings of the 31st international conference on Very large data bases. VLDB Endowment, 469–480.

[43] Michael Martin, Jörg Unbehauen, and Sören Auer. 2010. Improving the performance of semantic web applications with SPARQL query caching. In Extended Semantic Web Conference. Springer, 304–318.

[44] Richard Edward Matick, Jaime H Moreno, and Malcolm Scott Ware. 2006. Cache with selective least frequently used or most frequently used cache line replacement. US Patent 7,133,971.

[45] Geoffrey J McLachlan, Kim-Anh Do, and Christophe Ambroise. 2005. Analyzing microarray gene expression data. Vol. 422. John Wiley & Sons.

[46] Nimrod Megiddo and Dharmendra S Modha. 2004. Outperforming LRU with an adaptive replacement cache algorithm. Computer 37, 4 (2004), 58–65.

[47] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In Eleventh annual conference of the international speech communication association.

[48] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th international conference on machine learning (ICML-10). 807–814.

[49] Raghunath Othayoth Nambiar and Meikel Poess. 2006. The making of TPC-DS. In Proceedings of the 32nd international conference on Very large data bases. VLDB Endowment, 1049–1058.

[50] Arvind Narayanan, Saurabh Verma, Eman Ramadan, Pariya Babaie, and Zhi-Li Zhang. 2018. Deepcache: A deep learning based framework for content caching. In Proceedings of the 2018 Workshop on Network Meets AI & ML. ACM, 48–53.

[51] Sarang Narkhede. 2018. Understanding AUC-ROC Curve. Towards Data Science 26 (2018).

[52] Thomas Neumann and Guido Moerkotte. 2011. Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. In 2011 IEEE 27th International Conference on Data Engineering. IEEE, 984–994.

[53] Hieu V Nguyen and Li Bai. 2010. Cosine similarity metric learning for face verification. In Asian conference on computer vision. Springer, 709–720.

[54] Takayuki Osogami. 2010. A fluid limit for a cache algorithm with general request processes. Advances in Applied Probability 42, 3 (2010), 816–833.

[55] M Tamer Özsu and Patrick Valduriez. 2011. Principles of distributed database systems. Springer Science & Business Media.

[56] Meikel Poess, Raghunath Othayoth Nambiar, and David Walrath. 2007. Why you should run TPC-DS: a workload analysis. In Proceedings of the 33rd international conference on Very large data bases. VLDB Endowment, 1138–1149.

[57] Meikel Poess, Tilmann Rabl, and Hans-Arno Jacobsen. 2017. Analysis of TPC-DS: the first standard benchmark for SQL-based big data systems. In Proceedings of the 2017 Symposium on Cloud Computing. ACM, 573–585.

[58] D Rachmawati, JT Tarigan, and ABC Ginting. 2018. A comparative study of Message Digest 5 (MD5) and SHA256 algorithm. In Journal of Physics: Conference Series, Vol. 978. IOP Publishing, 012116.

[59] Qun Ren, Margaret H Dunham, and Vijay Kumar. 2003. Semantic caching and query processing. IEEE transactions on knowledge and data engineering 15, 1 (2003), 192–210.

[60] Xin Rong. 2014. word2vec parameter learning explained. arXiv preprint arXiv:1411.2738 (2014).

[61] Muhammad Zubair Shafiq, Alex X Liu, and Amir R Khakpour. 2014. Revisiting caching in content delivery networks. In ACM SIGMETRICS Performance Evaluation Review, Vol. 42. ACM, 567–568.

[62] Junho Shim, Peter Scheuermann, and Radek Vingralek. 1999. Dynamic caching of query results for decision support systems. In Proceedings. Eleventh International Conference on Scientific and Statistical Database Management. IEEE, 254–263.

[63] Cristiana-Stefania Stan, Adrian-Eduard Pandelica, Vlad-Andrei Zamfir, Roxana-Gabriela Stan, and Catalin Negru. 2019. Apache Spark and Apache Ignite Performance Analysis. In 2019 22nd International Conference on Control Systems and Computer Science (CSCS). IEEE, 726–733.

[64] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. 2012. LSTM neural networks for language modeling. In Thirteenth annual conference of the international speech communication association.

[65] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. 1997. Introduction to multilayer feed-forward neural networks. Chemometrics and intelligent laboratory systems 39, 1 (1997), 43–62.

[66] SM Shahrear Tanzil, William Hoiles, and Vikram Krishnamurthy. 2017. Adaptive scheme for caching YouTube content in a cellular network: Machine learning approach. Ieee Access 5 (2017), 5870–5881.

[67] Michael J Turmon and Terrence L Fine. 1995. Sample size requirements for feedforward neural networks. In Advances in Neural Information Processing Systems. 327–334.

[68] AI Vakali. 2000. LRU-based algorithms for Web cache replacement. In International conference on electronic commerce and web technologies. Springer, 409–418.

[69] Stijn Van Dongen and Anton J Enright. 2012. Metric distances derived from cosine similarity and Pearson and Spearman correlations. arXiv preprint arXiv:1208.3145 (2012).

[70] Xingxing Wang, LiMin Wang, and Yu Qiao. 2012. A comparative study of encoding, pooling and normalization methods for action recognition. In Asian Conference on Computer Vision. Springer, 572–585.

[71] William E Woods and Arthur Peters. 1982. Hit/miss logic for a cache memory. US Patent 4,363,095.

[72] Tian Xia, Dacheng Tao, Tao Mei, and Yongdong Zhang. 2010. Multiview spectral embedding. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 40, 6 (2010), 1438–1446.

[73] Hongliang Yan, Yukang Ding, Peihua Li, Qilong Wang, Yong Xu, and Wangmeng Zuo. 2017. Mind the class weight bias: Weighted maximum mean discrepancy for unsupervised domain adaptation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2272–2281.

[74] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, Ion Stoica, et al. 2010. Spark: Cluster computing with working sets. HotCloud 10, 10-10 (2010), 95.

[75] Victor Zakhary, Divyakant Agrawal, and Amr El Abbadi. 2017. Caching at the web scale. Proceedings of the VLDB Endowment 10, 12 (2017), 2002–2005.

[76] Engin Zeydan, Ejder Bastug, Mehdi Bennis, Manhal Abdel Kader, Ilyas Alper Karatepe, Ahmet Salih Er, and Mérouane Debbah. 2016. Big data caching for networking: Moving from cloud to edge. IEEE Communications Magazine 54, 9 (2016), 36–42.

[77] Michael Zheludkov, Timur Isachenko, et al. 2017. High Performance in-memory computing with Apache Ignite. Lulu. com.

[78] Chen Zhong, M Cenk Gursoy, and Senem Velipasalar. 2018. A deep reinforcement learning-based framework for content caching. In 2018 52nd Annual Conference on Information Sciences and Systems (CISS). IEEE, 1–6.

Software Evaluation Methodology of Node.js Parallelism under Variabilities in Scalable Systems

Maria Patrou
maria.patrou@unb.ca
University of New Brunswick
Fredericton, Canada

Jacob M. Baird
jbaird2@unb.ca
University of New Brunswick
Fredericton, Canada

Kenneth B. Kent
ken@unb.ca
University of New Brunswick
Fredericton, Canada

Michael Dawson
michael_dawson@ca.ibm.com
IBM Node.js
Ottawa, Canada

ABSTRACT

The backbone of Node.js is a single-threaded event loop, so computationally intensive tasks are bound to the performance of a single core. Modules with different architectures have been built to provide parallelism and scaling. However, their properties differ, making them appropriate for different cases. In order to assist software engineers in choosing the most appropriate module in the most efficient way, we perform an empirical study to investigate the modules' characteristics and functionality, taking into account system variances. Crucially, we present and apply an evaluation methodology focusing on four aspects: compute-intensive task execution, sharing data, communication and overhead. The results suggest that instance type (Node.js thread vs. Node.js process) is not enough to decide the most appropriate one. We find that modules with the highest performance in most cases can sacrifice other aspects, such as support and/or functionality and/or performance in fewer cases, while platform variances play a significant part.

KEYWORDS

Node.js,methodology,parallel,modules,share memory,communicate

ACM Reference Format:
Maria Patrou, Jacob M. Baird, Kenneth B. Kent, and Michael Dawson. 2020. Software Evaluation Methodology of Node.js Parallelism under Variabilities in Scalable Systems. In *Proceedings of 30th International Conference on Computer Science and Software Engineering (CASCON'20).* ACM, New York, NY, USA, 10 pages.

1 INTRODUCTION

An increasing number of applications are built with Node.js [23], a framework for asynchronous I/O, event-driven and server-side JavaScript. Examples include web applications and web servers that take advantage of its asynchronous and highly scalable nature. However, long-running tasks might arise and become a performance bottleneck. As in other programming frameworks/languages, we have to find efficient ways to execute compute-intensive tasks. Node.js, being mostly single-threaded, is not preferred for this type of computation. Fortunately, the framework supports modules that create new instances in which it can offload a task instead of running it in the main thread and blocking the event loop. The major categories include spawning either threads or processes.

There is much discussion on which technique should be preferred over another to achieve optimal performance. In Node.js, there are modules that have implemented Node.js processes and Node.js threads to achieve parallelism. In every case, their internal infrastructure and functionality have several (dis)similarities.

Expanding on our previous work [30], we perform an empirical study on thread and process implementations in Node.js by collecting and analyzing the results from our performance tool. We survey their architectures and explore their support and functionality though performance analysis that leads to better understanding of the modules and their capabilities. We formulate a methodology to evaluate them with benchmarks that describe: compute-intensive task execution by each worker instance, sharing information between parent and child instances and communication through two different message exchange patterns. In our methodology we present metrics for every aspect and include platform variances, as we scale the Node.js instances and their workload on a server with dynamic voltage and frequency scaling (DVFS). This study aims to provide further details of the Node.js modules, including a newer module, and a deeper analysis of their performance than the previous work. Our contributions are listed as follows:

(1) We survey Node.js modules that introduce parallelism. We choose modules that implement Node.js threads and Node.js processes and present their architectures and functionalities. (2) We provide a complete evaluation methodology. For choosing and utilizing a module, four main points are assessed (with metrics): overhead, task execution, sharing memory and communication. (3) We identify software and hardware variabilities. DVFS, evolution of Node.js/V8 between two different versions and execution environment are explored to reveal their impact on module performance. (4) We discuss observations and recommendations on module performance and utilization under scalable conditions.

2 BACKGROUND

The functionality of the hardware and software platform variabilities that we identified is described below. These components should affect the performance and scalability.

CASCON'20, November 10–13, 2020, Toronto, Canada

2.1 Hardware variability

The speed at which the instructions are executed is measured in clock cycles per second (CPU frequency). Modern CPUs can support DVFS in which the clock frequency and/or voltage of the processor changes for efficiency and energy consumption. Various governors, that decide the frequency and voltage values, exist. Intel's P-state driver has governors with P-states (which represent several frequency and voltage points): "performance" and "powersave". In the first case, the driver selects the maximum available P-states (high frequencies) and in the latter, the choice is made based on CPU utilization either relative to the usage or with a rapid increase as the workload increases. In cases that the processors support hardware-managed P-states (HWP), the processors select the P-states and the above algorithms provide indications based on the governor policy [4, 40].

2.2 Software variability

Node.js is a server-side JavaScript framework with asynchronous I/O, event-driven logic [31]. The event-based model depends on the concept of the event loop, achieved through the *libuv* [2, 20] library. The library is responsible for the event loop and thread pool functionality. The main (event loop) thread initializes the application and manages all requests and their responses. It maintains a queue of events and executes their callbacks. I/O network operations are executed through non-blocking sockets, while others, such as the file system, DNS functions, etc. are executed in a thread pool [2, 22].

Node.js runs on top of Google's V8 JavaScript engine [3], which is responsible for transforming JavaScript to machine code and allocating and deallocating memory. Internally, V8 uses an *isolate* structure, a private instance of V8 into which threads can enter one at the time (single-threaded). Multiple isolates can exist in multiple threads, each maintaining its own private heap for object allocations [36]. The *context* structure represents an environment in which unrelated JavaScript code is executed on a single V8 instance [37]. In V8, heap objects are collected though frequent garbage collection (GC) operations that free unused memory. The heap is split into two segments: nursery and old generation. There are two types of collectors: Scavenge that collects dead objects from the nursery and Mark-Sweep-Compact from the full heap. Scavenge creates small pauses in application execution and it happens frequently [19, 34]. However, GC improvements in v6.2 [34] replace Scavenge with a parallel Scavenge and add a parallel Mark-Evacuate algorithm. Node.js v10 [35] adds and enables concurrent marking.

3 RELATED WORK

Research has been conducted on performance evaluation of parallelism, Node.js and V8.

Node.js and V8 Zhu et al. presented an analysis of server-side JavaScript focused on the event-driven nature of Node.js [42]. They used Node.js and SPEC CPU 2006 applications and calculated instruction, cache and event metrics to present implications of the programming model and instruction cache optimizations. Nkenyereye et al. used throughput, response time and error rate to evaluate the performance of a healthcare hub server [29]. The focus was on the concurrent tasks as identified in the architecture of a remote healthcare monitoring system. They compared the single-threaded event-looped Node.js against a multi-threaded approach using Apache Sling and calculated their metrics using performance data from JMeter. Tiwari et al. studied architectural execution characteristics of Sunspider and Google's V8 JavaScript benchmarks [32]. They measured hardware performance counters to analyze and characterize the benchmarks. They also applied statistical techniques using PCA and clustering algorithms on performance metrics to identify similarities between them. Recent studies include the creation of a cloud-based benchmarking framework in Node.js to measure and present scaling performance and regression analysis [41].

Exploring Parallelism Namiot et al. described some JavaScript parallel programming models, Webworkers, WebCL and concurrent frameworks such as River Tail, JAWS and WorkerJS [28]. Vetter et al. evaluated the performance of eight scientific applications [39]. They presented computational, communication and scaling analysis using different tools. Vetter et al. performed a scalability analysis using a correlation coefficient and a rank transformation on the data [38]. More specifically, they evaluated the communication operations of applications from the NAS Parallel Suite and ASCI Compact Benchmarks to identify scaling issues.

Our previous study focused on the performance of modules in Node.js v9.7.1 under a compute-intensive task (Monte Carlo π estimation) looking into similarities and differences in performance terms [30]. We expand our methodology (benchmarks, metrics, test cases), move to a later Node.js version, and perform experiments for two Node.js versions and the modules that achieve parallelism (referred to as parallel modules) they support. This paper provides a survey on parallel modules and a complete methodology that focuses on the modules' infrastructure, performance under micro-benchmarks and factors that affect their performance. To the best of our knowledge, this analysis has not been found before in literature.

4 HYPOTHESES & RESEARCH OBJECTIVES

We identify and evaluate parallel Node.js modules under software and hardware variabilities. First, we survey representative modules. Second, we assess them and the properties which can impact their behavior. Our goal is to stress-test the modules on core operations that we identify as important factors for choosing a module and its functions using the micro-tasks. Considering the impact of the underlying environment and Node.js variances, we investigate the following hypotheses: (H1) Multi-thread modules will have lower overhead and higher performance than multi-process ones. The instance type will be the most important factor for determining their performance. (H2) The Node.js version upgrade will affect positively and evenly all the modules. All modules will either experience better or similar performance on later versions. (H3) The environment will affect the modules. DVFS will affect the scalability of modules. We expect to see cases with higher computation to be faster than others with lower workloads. Also, an impact on task execution when moving from a bare metal host to a virtualized environment is anticipated. (H4) Passing an argument during worker spawning will be the fastest way to share information. Sending a message and using a shared memory module will follow for small data, while for large data the opposite sequence will be more efficient.

The shared memory modules will have the highest overhead. (H5) Communications patterns will affect the system responsiveness with significant differences. The format of message exchange will show that sending one large message is more efficient that sending smaller ones with the data split.

5 SURVEYING PARALLELISM IN NODE.JS

Node.js enables the import of any file or organized folder as a module [1] in order to access its functions. Modules that spawn instances face the challenge of exposing parallelism in an environment that is mainly single-threaded. Node.js implements an event-based infrastructure in which the events are orchestrated on the single-threaded event loop and the embedded V8. The runtime exposes a V8 instance (isolate) which is meant to be used by one thread at a time and a context that is the execution environment of a single instance. Thus, thread implementations do not share the same V8 environment, including the heap (aka objects) and they also need to add and define a structure that supports several threads and the way they fit in the Node.js architecture. However, Node.js provides the concept of *NodePlatform*, created per process, that can point to multiple isolates and provide libuv's background threads [16]. In our opinion, the creation of Node.js processes benefits more from the Node.js/V8 architecture, since by default they share nothing.

The investigation of modules with different implementations for achieving parallelism leads us to five modules. By reading their documentation and code, we survey and collect two process-based modules: Child Process, Cluster and three thread-based modules: WebWorker-Threads, Napa.js and Worker Threads. The multi-process modules are found throughout the Node.js versions, while WebWorker-Threads and Napa.js were not supported in Node.js v12.1.0 in the duration of the study. However, the latest addition is the Worker Threads that appears in later versions. Argument passing, message sending and shared memory modules are supported by Child Process, Cluster and Worker Threads, that are part of the Node.js core. While Napa.js does not support message passing and WebWorker-Threads supports only message passing. Every module has its own architecture and API:

Napa.js is a multi-threaded JavaScript implementation, used as a Node.js module or standalone. It creates symmetrical worker-threads, which share the same configuration settings, defined as a *zone*. The module allows for the creation of multiple zones in the same application. A *napa zone* allows the creation of multiple threads, using a JS thread pool in which they have their own V8 *i*solates and thus, private heaps [9]. The napa zone does not use the libuv APIs within the worker threads; the library is exposed only through the *node zone* for one worker [6, 7]. During the initialization of the zone, each worker is initialized using the zone settings with a worker id. In this step, each worker loads the appropriate modules/libraries, too. The workers are initialized sequentially and they wrap *std::thread*s. The threads share memory through Transportable types, including JavaScript primitive and built-in types such as SharedArrayBuffer (SAB) and ArrayBuffer [5]. The design for making structured data transportable such as SAB is based on solutions that use externalized memory to store/share memory (for SAB objects) and V8's object (de)/serialization system [8].

WebWorker-Threads create JavaScript threads (following the W3C Web Worker API) that run in the background with the main thread [13]. The module supports the creation of thread pools and individual workers. In contrast to Napa.js, the architecture relies on the usage of the *libuv* library. It spawns native threads (*uv_thread_t*) that execute jobs using their shared queues and send the jobs' callbacks back to the event loop. Each thread has its own isolate and thus its own heap [14]. The worker threads share information from the main thread by sending messages. A message is serialized and added as a job into the thread's queue for execution [15]. Finally, the module includes parts of the code written in LiveScript that are transformed to JavaScript minified versions with hexadecimal representations of the ASCII characters [21].

Worker Threads is the most recent built-in Node.js module that allows for parallel task execution. It creates JavaScript threads that support most Node.js APIs [18]. A worker thread is an EventEmitter object, thus it triggers events with listeners to be called upon emission [17]. During the creation of a worker thread, the parent process's environment is cloned, a new MessageChannel is created (for asynchronous communications between parent and child worker), internal modules are loaded, its script (task) is passed as a message and the standard input/output streams are initialized. Every worker thread wraps a *pthread*, maintains its own V8 instance, Node.js environment and event loop, but shares resources with the other threads, such as the thread pool [26]. The information between main and worker threads is passed during the creation of a thread as an argument (*w*orkerData) or by sending a message with the default or a custom MessageChannel (the default global one is used in the experiments). In both cases, the HTML cloning algorithm is used to copy the data using a message event. Also, Worker Threads support SharedArrayBuffers for sharing data that are based on V8's APIs. Finally, the workers can share environment variables with the main thread by setting the special flag: *w*orker.SHARE_ENV [18].

Child Process spawns a new Node.js process. The function child_process.fork spawns a process asynchronously and invokes a module for IPC communication [11]. More specifically, it creates a new *ChildProcess* object, an EventEmitter object, with listeners attached. During creation of the object, the parent process creates the IPC channel, initializes the file descriptors for the standard I/O and passes them as an argument, along with the rest of the environment variables, such as the executable script and args, to the child process. Inter-process communication between the parent and the child process is achieved either through Domain sockets in Unix-based environments or with Named Pipes in Windows [20]. The new Node.js instance/executable is spawned through libuv's *u*v_spawn (that calls *e*xecvp) and thus it has its own environment, event loop [20], V8 instance and private heap. By default, the processes do not share memory, but this can be achieved through shared memory segments. The *shm-typed-array* module (used in the experiments) enables sharing memory between Node.js processes through the IPC mechanism (Unix's shm library) for Unix systems [33].

Cluster creates processes that share server ports. A new process is spawned using the child_process.fork API internally [12]. First, the settings of the cluster module are initialized, including the executable script, arguments, flags and the scheduling policy is set to

default round-robin. Internally, the module calls the child_process's API for each child process and passes the appropriate settings for the cluster. By default, a cluster process will execute the parent's script from the beginning and it is the developer's responsibility to differentiate the execution between master and slave threads. Nevertheless, the module enables the assignment/execution of a separate script through the cluster.setupMaster() API, which was used in the experiments for calculating the metrics related to task execution to reduce the overhead of the child processes.

Our survey reveals that the above modules have some characteristics in common. Each thread/process has its own isolate and thus its own heap. However, the modules can share information regardless of the instance type (thread or process) using different libraries. The only restriction is whether modules support certain ways of sharing and exchanging information.

6 EVALUATING NODE.JS PARALLELISM

In our methodology, we identify four main aspects for testing the modules' performance, we design our benchmark suite based on them and evaluate the modules in a wide range of loads.

6.1 Module Characteristics & Variabilities

We believe that, the way the modules affect the system and their performance on task execution, sharing information and message exchange are the most important characteristics to make a decision on a module and its functions.

First, the overhead from loading the module and spawning the workers and the sharing memory techniques, is explored. Applications that heavily depend on fast execution, such as animations, would consider when and how to spawn a new worker, to avoid significant slowdowns. Also, the overhead caused by the workers, can be a factor for determing the most suitable number.

Regarding task execution, workers are expected to execute a compute-intensive task. Thus, the selection depends a lot on their performance on this task. At the same time, they might require sharing information with the main thread. The performance of the various mechanisms gives an insight on when and how to use each one. Finally, message exchange, reveals the communication impact on the system. Applications that need frequent communication among the main program and the helper instances, should consider the size and number of messages and the communication type.

However, the deployment environment affects the modules, as well. It is not guaranteed that the production environment will always be the same as the development one. Hardware discrepancies, like DVFS affect the frequency of the CPUs and, thus the module performance. Moving from a bare-metal host execution environment to a virtualized one, can cause a further impact on task execution. Even the Node.js version upgrade can have side effects. To this end, our trials and metrics take into consideration the module patterns and the influence from their environment.

6.2 Benchmark suite

The following benchmarks are used to evaluate the modules:

Task execution. The benchmark simulates cases in which workers are called to execute a compute-intensive task and return a result to the main program. The Monte Carlo π estimation algorithm is used. We took the implementation from the Napa.js benchmarks suite [10] and adjusted it to our trials. Each worker executes the algorithm independently by performing n iterations and sends back its calculated value through a message or a promise (for Napa.js). When they have finished, the main program returns a callback with the calculated π value to continue the execution of the application and the aggregated value is printed.

Sharing information. The modules provide different APIs to share data from the parent to the child. Argument passing during spawning the worker, message sending and shared memory techniques are tested, based on the support of each module. Regarding the shared memory techniques (SharedArrayBuffer–SAB for multi-thread modules and shm-typed-array–SHM for multi-process modules), the objects are passed as an argument during the forking process. In all cases, the parent instance is notified by the worker with an event emission: message or promise that the worker has successfully received the data. We examine each technique for each module with two benchmarks that differentiate the range of random integers: any random integer and integers from zero to nine.

Communication cost. Two different patterns for message exchanging between parent and child instances are tested. In the *Batch* pattern, the parent sends a message to all workers at the same time. In the beginning, the parent creates a random array. The workers receive the array, swap the first element with the last and send it back. The main thread/process resends it to all of them, after it receives the array from all the workers and swaps the two elements. In the *Pair* pattern, the main thread resends the array immediately to the worker that it received it from.

The benchmarks are written with each module's API and configured through scripts to change the number of workers and the loads. Each benchmark category/module is configured in a separate Docker container [27], with the exception of the sequential case that is in the same container as the Child Process. Each container has its own Dockerfile with the appropriate libraries and modules.

6.3 Experimental Design & Metrics

We choose the number of workers based on the number of CPUs (eight) on the machine. We scale them based on the number of CPUs used by the workers (one, four, eight and 16) and the number of CPUs used by the workers and the main thread (three, seven and 15) to reach the number of cores (four), CPUs (eight) and double CPUs (16). The trials from task execution and communication use the above configurations, while the experiments that focus on memory scale the workers as follows: one, four, eight and 16. The workloads for every experiment scale in a wide range of loads to simulate small, medium and large ones executed by each worker. We perform 30 runs for every case (with the exception of shared memory case with random numbers with Child Process in version 12) and present the average and standard deviation bars.

On task execution, we measure and reveal variances that impact task execution, including Node.js version and compiler and DVFS. Execution time, speedup, CPU usage/frequency and GC patterns are presented to decide on speed, resource consumption and heap utilization. To compare the shared memory techniques, we measure the time spent spawning each worker until all workers have notified the parent. On communication, we measure execution time and

CPU usage/frequency to decide speed and resource utilization. The execution time does not include the overhead of spawning the workers to show the time spent on message exchange. Regarding overhead, we use the benchmarks from task execution and sharing memory with minimum workloads and we present the speedups against the sequential case.

7 EXPERIMENTAL EVALUATION

The multi-process modules: Child Process (CP), Cluster (C) and multi-thread modules: WebWorker-Threads (WT), Napa.js (N) and Worker Threads (W) on the Node.js versions: 9.7.1 (V8 6.2.414.46) and 12.1.0 (V8 7.4.288.21) and their supporting techniques are tested. We ran our experiments under Ubuntu 16.04.4 LTS, with 8GB RAM, four Intel i7-2600 cores and two hardware threads per core. It supports frequency scaling with minimum and maximum: 1.6GHz and 3.8GHz in "powersave" governor with intel_pstate active mode. We also created a virtual machine using *VBoxManage* and ran Ubuntu 16.04 with 8CPUs and 8GB of RAM and KVM virtualization.

7.1 Overhead

First, we investigate the speedup of the modules in Node.js 12.1.0 and Node.js 9.7.1 with the different techniques for sharing information. In most cases, the argument passing technique has the least overhead followed by the message passing and the shared memory solutions. The results are not surprising as passing an argument while creating a worker instance does not require any additional steps. However, the difference in execution time is minor. In fact, Napa.js shows similar results between argument passing and shared memory with one and eight workers. Also, Worker Threads in v12 show cases where the three techniques overlap including their standard deviation. Since the difference is minor, no one technique has a big overhead difference.

Figures 1a and 1b reveal the cost of spawning a new worker instance for the the Monte Carlo π estimation with one point. The results reveal that speedup increases as workers increase and it starts dropping at 15 workers. Considering that each worker has only one iteration to estimate the π value, the graph shows that it is faster to spawn more instances than one, especially in Node.js v9. In fact, WebWorker-Threads have speedup values higher than one (better performance than the sequential program) in most cases. While in Node.js v12, the overhead of spawning three, four, seven and eight workers is similar. Spawning four workers produces the highest speedup among the above and the most similar speedups are observed in cases with seven and eight workers.

Comparing these results with Figures 1c and 1d in which the server is set to: minimum frequency same as maximum (3.8 GHz) and to "performance" governor, to reduce the changes in CPU frequency, the speedups have the highest value with one worker and decrease as the workers increase. They are always less than one, while the standard deviations are smaller with more consistent results. Thus, DVFS affects the scalability of the modules in this machine. The CPU frequency changes to benefit the spawning of more than one worker (process or thread) with minimum work to execute, as long as they are less than or equal to the machine's hardware threads. In most cases, the highest speedups appear with three and four workers, in which the turbo boost can be activated with speed at least 3.5GHz.

7.2 Task execution

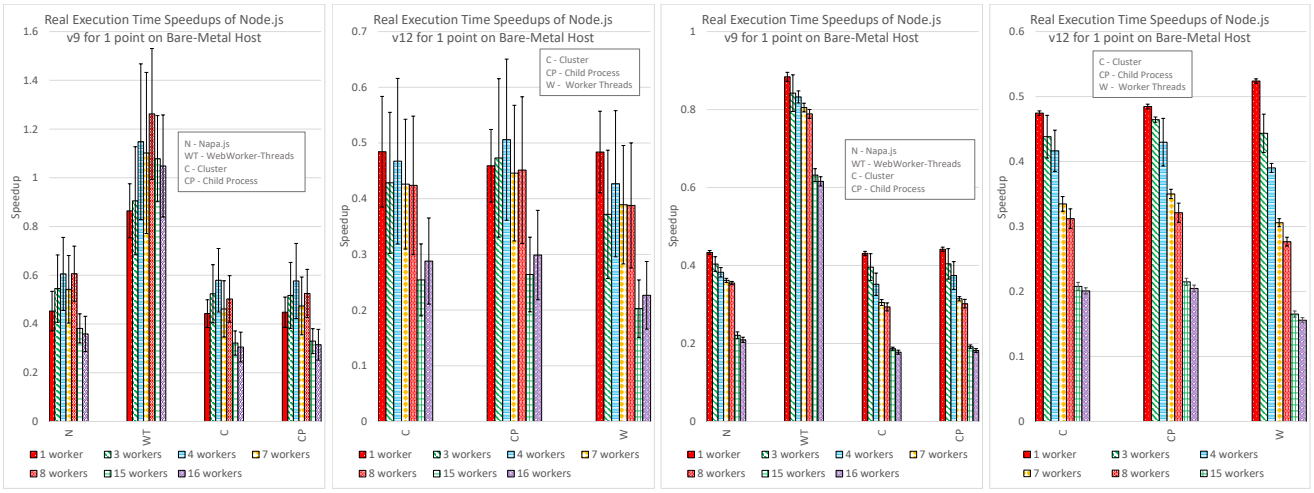
In this trial, we measure the task execution by each worker and report the variances that affected it.

7.2.1 Run times. Figures 2a and 2b present the **speedups** as the workload (iterations) per worker increases in the x-axis and the workers increase in the y-axis. Napa.js and WebWorker-Threads show higher speedups than Child Process and Cluster. WebWorker-Threads have the highest speedup, while Napa.js follows as the workload increases. For small workloads, Napa.js has the lowest speedup. However, by excluding the step of spawning JavaScript threads during the "zone" creation, threads from Napa.js produce speedups closer to WebWorker-Threads for every workload. Focusing on the modules' speedups of task execution, we measured the execution times starting from the assignment of the tasks to the workers until all workers have returned their values to the main thread/process. Napa.js separates the zone creation with the task assignment to each worker and performs it in two steps. The zone creation appears to have such an overhead, that it can change the trend. Napa.js with the zone overhead has the lowest speedup in small workloads, while without it, it has the second best. In Node.js v12, Cluster and Child Process have higher speedups than the Worker Threads. Worker Threads have closer speedups to Cluster, while Child Process outperforms both.

Focusing on the modules' **execution time**, the data shows that they decrease their real execution time as the workers increase from one to three and four workers for workloads up to 4M iterations. The runtimes remain lower in the case of seven and eight workers for the same or fewer points and increase and surpass the single-worker runtime for 15 and 16 workers. These results along with the data on overhead show that in this machine, it is faster to spawn more than one (and less than 15 workers) to perform a small task.

The comparison of the sequential case, the Child Process and Cluster modules between Node.js v9 and Node.js v12, reveals that their execution time differs and improves in the latter **version**. The comparison of their ratios show that they perform better in Node.js v12. In fact, the sequential case and the Child Process module benefit more as the workload increases. On the other hand, the Cluster module benefits in small workloads and then runs with the same speed for workloads more than $2 \cdot 10^7$ and every number of workers. Overall, their execution time in Node.js v12 can be more than 1.5 times faster than in Node.js v9, with the sequential case reaching the v9/v12 ratio 2.48 and Child Process 2.46.

Comparing the performance of **multi-process** modules in both **versions**, the results show that in Node.js v9, Cluster and Child Process have almost identical speedups, while in Node.js v12, Child Process outperforms Cluster. The latter is surprising, given the fact that Cluster encapsulates some of Child Process's functionality. The investigation on Node.js flags coming from V8, reveals performance difference with default settings against the *–no-opt* **flag** which disables any optimizations happening from V8's Just-In-Time compiler, called TurboFan [25]. There is a performance difference with default settings that causes Child Process to be more than twice as fast as Cluster and it disappears when the compiler



(a) Node.js v9 **(b) Node.js v12** **(c) Node.js v9** **(d) Node.js v12**

Figure 1: Speedup for one point - Monte Carlo π estimation - CPU Frequency set to min=max in: (c) and (d)

is disabled. The performance difference increases as the workload per worker increases indicating a connection with the iterations for the π value estimation, in which two numbers are created with the Math library. Thus, this can be connected with the way the compiler optimizes this code in each module for this server.

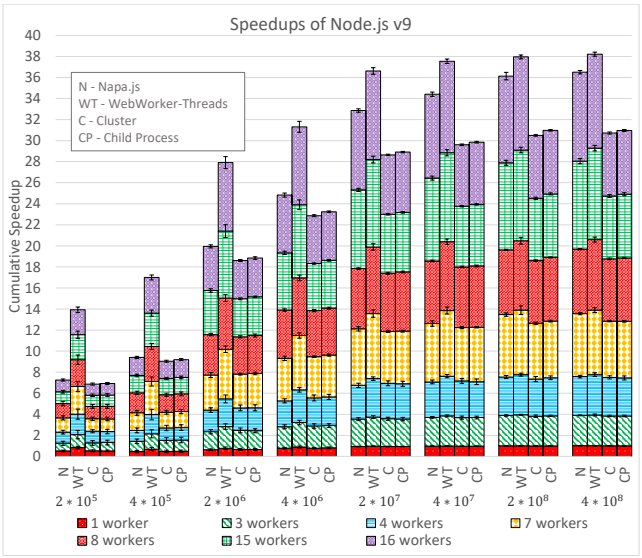
The investigation on the **virtualized environment** shows that the speedups from the experiments on the virtual machine follow very similar trends as in the bare-metal host case. Running a compute-intensive task in a host versus in a virtualized environment on the same host does not affect the execution times of the parallelization modules unevenly for large workloads and many workers. Overall, the performance is either the same or worse in a virtual machine for every module. However, Napa.js and WebWorker-Threads are affected more than Cluster and Child Process by the virtualized environment. WebWorker-Threads and Napa.js reduce their overall speedup in small workloads significantly. In fact, multi-process modules show better speedups than Napa.js in these cases, while Napa.js shows slightly better speedup on the bare-metal host. Also, the ratio of the modules' real execution time in the VM over the host reveals cases that the multi-thread modules experience slowdowns of 2.16x for Napa.js and 2.46x for WebWorker-Threads in the VM, while multi-process show up to 1.34. WebWorker-Threads show the fastest speedup and Napa.js follows in large workloads. Similarly, in Node.js v12, Worker Threads experience higher slowdowns (up to 2.31) than the Cluster (up to 1.5) and Child Process (1.54) modules. The trends do not change—Child Process shows the fastest speedup. Overall, the performance difference occurs in small and medium workloads and it decreases as the workloads increase. Such a difference is significant to highlight, in case an application with a multi-thread module is deployed on a virtualized environment. The transition from a bare metal host to a virtualized environment should have a bigger impact on a multi-thread solution than on a multi-process one for a compute-intensive task.

7.2.2 CPU Metrics. Napa.js and WebWorker-Threads use the least CPU. Multi-process modules have very similar CPU usage and frequency in Node.js v9. In small workloads, the CPU frequency does not always increase as workers increase. The frequency increases rapidly from one worker to three and four workers and then decreases with values either the same with a single worker or even higher. Also, WebWorker-Threads uses lower frequency in $2 \cdot 10^6$ than in $4 \cdot 10^6$ workloads.

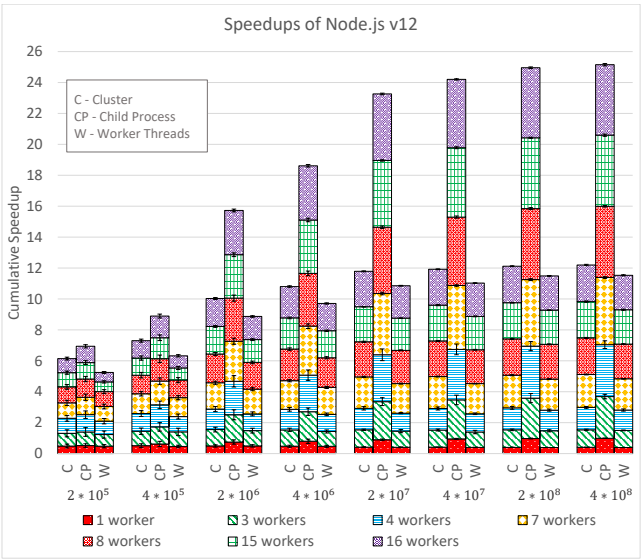
In Node.js v12, the trends are not that clear. Worker Threads use less CPU than Child Process and Cluster. As the workloads increase, the CPU usage of the three modules converges in large workloads. Worker Threads have the lowest CPU frequency. In small workloads, Child Process decreases the frequency as workers increase, while as workloads increase the values are closer together regardless of the number of workers. Similarly in Cluster and Worker Threads, there is no trend that frequency increases or decreases based on the number of workers. However, when the workload per worker is large, it reaches a high value and does not change.

Overall, the multi-thread modules use less CPU and lower frequency for low and medium workloads. At $2 \cdot 10^7$ points, CPU frequency stabilizes at 3.5GHz for all parallelization modules and it does not increase with all modules running on similar CPU frequency. The frequency decreases slightly as workers increase with ranges from 3.69GHZ to 3.47GHz. However, the CPU usage increases steadily as workers increase.

7.2.3 Memory Management - Garbage Collection. In Node.js v9, Sequential, Napa.js, Cluster and Child Process perform almost the same **number of GCs** especially as the workloads increase. They also produce almost the same number of heap garbage collected objects, WebWorker-Threads included. More specifically, the sequential case has the fewest **heap allocations** and number of GCs in lower loads and Napa.js the highest among these four cases. This result suggests a higher memory footprint in Napa.js. On the other hand, WebWorker-Threads has the most GCs, almost twice



(a) Node.js v9



(b) Node.js v12

Figure 2: Monte Carlo Speedups on Bare-Metal Host

more in some cases. The results could be explained by the different starting heap sizes for each technique. Napa.js threads have 9.1 MB, Sequential, Cluster and Child Process processes have 3.4 and WebWorker-Threads have 2.6 MB.

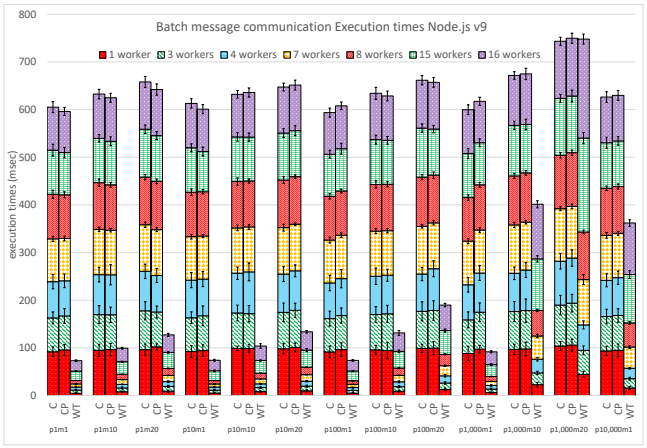
In Node.js v12, the trend is similar. In low workloads, the sequential case does the fewest GCs, while the parallelization modules perform almost the same number of GCs for every workload and approach the sequential case as the workloads increase. The Worker Threads start with heap size 2.19MB, the Cluster and Child Process processes start with 2.18MB and the sequential 2.19MB. Investigation on the number of GCs and the heap allocations until the last GC show that among the sequential, Cluster and Child Process isolates, the number of GCs are similar in both Node.js/V8 **versions**, with v12 producing fewer GCs as the workloads increase. Regarding heap usage, the results are similar, but v12 modules have fewer heap allocations for small workloads. In every case, the number of GCs and heap allocations are consistent revealing that they can be predicted using the information for one isolate and applying when scaling out.

Finally, we measure the **duration** of all GCs using the logs that show the GC pause [24] with the user and system execution time of each technique. The ratio shows the percentage of time spent in GC and reveals the time spent on freeing unused memory during application execution. For up to three workers, Napa.js spends the most time in GCs and WebWorker-Threads spends the least. As the workers and load increase, Napa.js approaches the performance of Cluster and Child Process, which spend the least time in GC. More specifically, as workers increase, the multi-process techniques spend the least time for small and medium loads. When they reach 15 and 16 workers, they spend the least GC% time for tiny loads and then increase at a high rate as the load increases. They reach up to 100% GC duration, meaning that there is always a GC happening throughout the whole application duration. On the other hand, Sequential has the lowest GC% duration. This shows that most of the application time is spent in executing the task rather than freeing up the heap. In Node.js v12, the modules show similar percentages in GC% duration. In cases with large workloads, Worker Threads have the lowest percentage, while in small workloads Cluster and Child Process have lower. For all modules in both versions the GC% duration is less than 6% up to eight workers. The percentages increase at 15 and 16 workers.

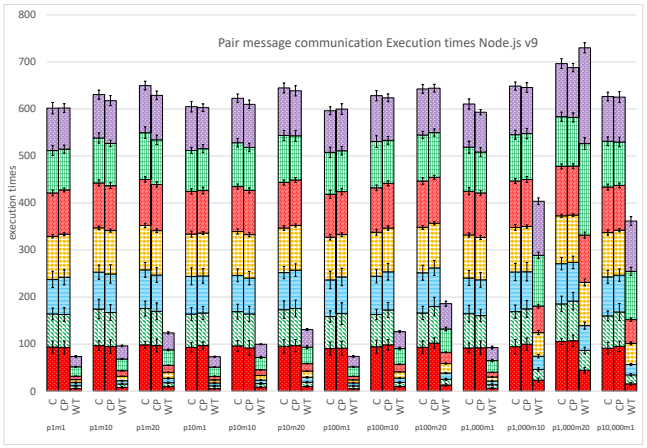
7.3 Sharing Information

The execution time of the modules sharing an array among workers that contains **random integers** with a random number of digits is measured. For four workers and small arrays, argument passing is the fastest technique among the modules. As the arrays increase, Napa.js performs better with shared memory, while Cluster and Child Process, with argument or message passing. For more workers, argument passing is still faster, except for Napa.js where the shared memory technique is faster for large arrays. However, the difference in execution time among each technique for every module (except for Napa.js with large arrays) is minor, regardless of the number of workers and the size of the array. In Node.js v12, argument passing shows the least execution time for small arrays. As the size of the arrays increases the shared memory technique proves to be more efficient. A similar outcome happens for 16 workers. WebWorker-Threads shows the least execution time followed by Napa.js in Node.js v9 and Child Process and Cluster show the least in Node.js v12.

The results show that each technique (with the exception of Napa.js) does not increase substantially as the array increases. The execution time increases more as more workers are spawned and share the array. Also, there are a few cases that sending a message is faster than the other two methods. If all of them are supported the usual order is: argument passing, message sending and shared memory. Shared memory is faster in cases that large arrays are

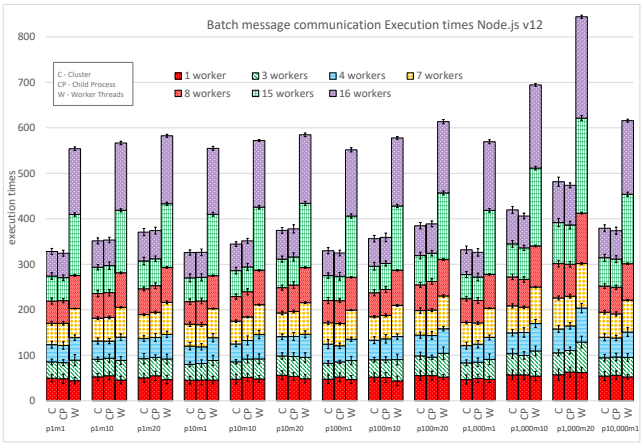


(a) Batch

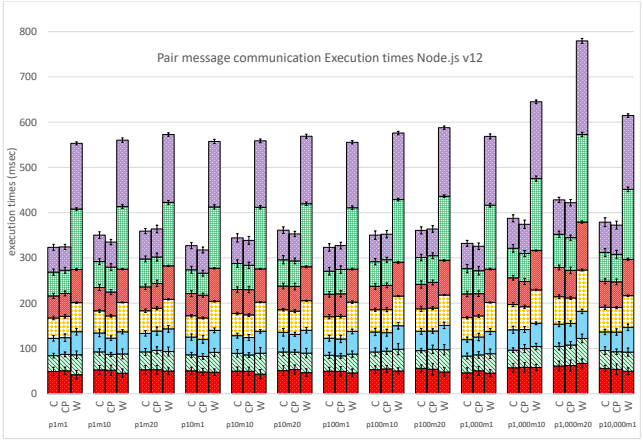


(b) Pair

Figure 3: Batch-Pair Patterns Execution times Node.js v9



(a) Batch



(b) Pair

Figure 4: Batch-Pair Patterns Execution times Node.js v12

shared. However, the difference is minor in most cases, with the exception of Napa.js for large arrays.

The modules are, also, tested with arrays that hold integers with **one digit** (0-9). Larger arrays are used to evaluate the performance of each technique, too. Here, the trends are very similar to the previous benchmark. However, Cluster and Child Process receive the data in argument passing technique as strings. In this case, there appears to be a limit on the number of integers shared among the worker processes. Arrays with size 10^5 and 10^6 cannot be passed as arguments.

Overall, argument passing appears to be a fast way to share data. In cases with large arrays, the shared memory technique creates a big performance gap, especially in multi-process implementations, and shows to be the fastest solution. Message passing has the least execution time in very few cases.

7.4 Communication Cost

Figures 3 and 4 present the execution time for Batch and Pair patterns for cases up to $p10,000m1$. The x-axis shows the array size with random-sized integers-points (pn) followed by the number of messages sent from the parent to each worker (mk).

The modules in **Node.js v9** have very similar **execution time** for all cases between $p1m1$ and $p100m20$. In both patterns, the trends are similar. The execution time increases slightly as the number of messages increases, while there is no significant difference in the performance as the message size is increased. Even though, the number of messages exchanged affects the performance more than the number of points sent, within each message, the gap remains small. As the workers increase, WebWorker-Threads increase their execution time. Child Process and Cluster have the shortest duration with three and four workers and the execution time increases gradually as workers increase. Overall, WebWorker-Threads have the least runtime and that creates a big performance gap with Cluster and Child Process. However, as the array size increases, WebWorker-Threads increase the execution time as such that exceeds the multi-process modules. In both patterns, WebWorker-Threads show very small execution time with small message size and as the size and number of messages increase show the slowest execution time. Also, for $p100,000$ with 10 and 20 messages the module is not supported.

In **Node.js v12**, Cluster and Child Process have the best execution time for any message size and number. However, Worker Threads perform best for one worker with small and medium array sizes and decrease their performance as the points increase ($p10,000$). For small and medium size arrays, Cluster and Child Process have their least execution time for three and four workers and Worker Threads have their least execution time in three workers with small points (more than one point).

In both versions, the modules show a slight performance improvement when parent and child exchange one big message, rather than ten smaller ones. The graphs show that in cases with pnm10 and p$10n$m1 the overall execution time is less with one message exchanged for any pattern. Additionally, in cases: $p100,000m10$ and $p100,000m20$, Child Process and Cluster modules have their execution time around twice more in the Batch pattern than in the Pair one, while for smaller cases the runtimes are similar. Worker Threads are not affected by the pattern change in the same cases.

Regarding **CPU utilization**, Cluster and Child Process use the most CPU in the majority of cases. For cases with less than 10,000 points (excluding $p1,000m20$), the CPU usage of all modules maintains very similar values for the same number of messages. However, the CPU usage for the multi-process modules drops afterwards. Considering that the execution time increases as points and messages increase, sending a message becomes a bottleneck. In cases with the largest array with 10 and 20 messages, Child Process and Cluster spend twice as much time in Batch pattern compared to the Pair pattern and use less than two CPUs in Batch pattern. Thus, waiting to receive everyone's array and then sending it back becomes inefficient. For the other cases with smaller messages, the patterns do not have a significant difference. In Node.js v9, WebWorker-Threads use less CPU than Cluster and Child Process. In Node.js v12, Worker Threads use the least CPU in the Pair pattern for every case and in the Batch pattern, all cases except the two with the largest array and 10 and 20 messages.

The **CPU frequency** is also higher for multi-process modules. However, as the message size and the number increase the multi-thread modules reach and sometimes overcome their values. The frequency values of Child Process, Cluster, in both Node.js versions, and Worker Threads show that the frequency increases significantly from one worker to three workers and four workers. On the other hand, WebWorker-Threads decrease their frequency as workers increase for small messages.

8 DISCUSSION

The performance analysis within our test environment reveals aspects that need to be taken into consideration during software design and bottleneck analysis. The outcomes related to our hypotheses in parentheses are as follows:

The first observation is the overhead of spawning new workers. The results show that this overhead can be high in both Node.js thread and process implementations. In contrast to our first hypothesis, the creation of a Node.js thread does not guarantee lower overhead. In fact, the overhead can be similar or higher than the multi-process implementations. WebWorker-Threads, followed by Napa.js, shows the best performance in Node.js v9.7.1, while Child Process, followed by Cluster, shows the best in Node.js v12.1.0. In both versions, the multi-process modules are faster when communicating with very large arrays, than the multi-thread ones. However, the above modules have some limitations on their support and functionality. They either support fewer APIs for information exchange (argument passing and shared memory techniques were not supported in WebWorker-Threads and message passing was not supported in Napa.js) and/or less data exchanged in some cases (arrays larger than 10^5 could not be sent as arguments in Cluster and Child Process and arrays of 100,000 points with 10 and 20 messages could not be sent in WebWorker-Threads). On the other hand, Worker Threads do not have any limitation on the same experiments. (H1)

The performance and the module support depends on the Node.js version. Thus, there is not a clear winner for both versions. The modules are presented and compared within their versions. Among versions, changes are introduced to Node.js, that affect the whole system's performance and make it unfair to draw any conclusions on modules for both versions. Also, the Node.js version appears to be an important factor for choosing a module, because of version incompatibilities. The transition from Node.js v9.7.1 to Node.js v12.1.0, shows benefits in the performance with lower execution time and fewer GCs. Thus, applications that use the above modules, can experience better execution time in a newer Node.js version. However, findings show that in the latter version, the Cluster and Child module are not optimized evenly in certain cases. Thus, the code compilation might benefit the performance differently. (H2)

Argument passing has the least execution time in many cases, but the difference with other techniques is minor. Also, there is a minor overhead difference among them. The performance in Napa.js between argument passing and shared memory is obvious in several cases, while for the other modules the gap appears in large arrays, in which shared memory techniques outperform. However, argument and message passing allow for more flexibility on the type and size of the data sent than in the shared memory solutions. In the latter cases, the developers need to define the data size and type that is shared across instances. (H4)

In most cases, the communication pattern does not affect performance. Only multi-process modules are affected for large message exchanges. The gap with different message numbers and sizes is minor within each module, but the modules show slightly better performance with one larger message exchanged than ten smaller. A big performance gap appears for messages with large arrays. (H5)

The environment affects the performance of all modules. DVFS affects the scalability of the parallelization modules, by changing the "speed" of CPUs as more work is executed by them. Spawning more than one worker can be faster with the current CPU scaling driver/governor and CPU boost limits. Usually, Node.js threads use less CPU and lower frequency. However, the execution environment affects the multi thread implementations more than the multi process ones. The execution of a compute-intensive task in the virtualized environment creates higher slowdowns in the multi thread implementations compared with multi process ones. The difference, though, does not change the trend on which module has the highest speedup, since all modules experience slowdowns. (H3)

9 CONCLUSION AND FUTURE WORK

Node.js allows the creation of parallelization modules with different architectures. Understanding and investigating key functionalities can give insight on which one is more appropriate in every case. Thus, we provide an evaluation methodology and survey Node.js parallelization modules, while we consider variabilities in a software and hardware level. We formulate hypotheses and perform an experimental study to conclude on these statements and provide better module utilization.

However, our focus is on task parallelism on one machine. In the future, we can use our methodology for evaluating modules on distributed systems with the latest Node.js/module version and considering performance interference. Also, our study is based on micro-tasks that while we focus on specific individual aspects, the outcomes might deviate in complex real-world cases. Following a similar methodology with more elaborated scenarios can reveal further observations and the connection between controlled micro-tasks with real-world scenarios, including more variabilities.

ACKNOWLEDGMENTS

This research was conducted within the Centre for Advanced Studies—Atlantic, Faculty of Computer Science, University of New Brunswick. The authors are grateful for the colleagues and facilities of CAS Atlantic in supporting our research. The authors would like to acknowledge the funding support of the Natural Sciences and Engineering Research Council of Canada (NSERC), 501197-16. Furthermore, we would also like to thank the New Brunswick Innovation Foundation for contributing to this project. We also thank Stephen MacKay for his careful proofreading and editing the paper to improve its quality.

REFERENCES

[1] [n.d.]. Node.js v14.4.0 Documentation - Modules. https://nodejs.org/api/modules.html. [Online; accessed 15-June-2020].
[2] 2014. Libuv 1.20.4-dev Documentation. http://docs.libuv.org/en/v1.x/design.html. [Online; accessed 13-July-2018].
[3] 2015. Chrome V8. https://developers.google.com/v8/. [Online; accessed 13-July-2018].
[4] 2017. Intel P-State Driver. https://www.kernel.org/doc/Documentation/cpu-freq/intel-pstate.txt. [Online; accessed 19-December-2019].
[5] 2018. Napa.js - Namespace Transport. https://github.com/microsoft/napajs/blob/7e934ecc125a02088c25e1a58a8eb1b6fad80f6d/docs/api/transport.md. [Online; accessed 13-November-2019].
[6] 2018. Napa.js - Namespace Zone. https://github.com/microsoft/napajs/blob/73460d244438e91d302194ee02de2a911a9bb731/docs/api/zone.md. [Online; accessed 13-November-2019].
[7] 2018. Napa.js - Napa.js Module. https://github.com/microsoft/napajs/blob/master/docs/api/module.md#js-vs-cpp. [Online; accessed 13-November-2019].
[8] 2018. Napa.js - Transport JavaScript Standard Built-in Objects. https://github.com/microsoft/napajs/blob/7e934ecc125a02088c25e1a58a8eb1b6fad80f6d/docs/design/transport-js-builtins.md. [Online; accessed 13-November-2019].
[9] 2018. Napa.js: A Multi-threaded JavaScript Runtime. https://github.com/microsoft/napajs. [Online; accessed 20-July-2018].
[10] 2018. Napa.js: estimate-pi-in-parallel.js. https://github.com/microsoft/napajs/blob/master/examples/tutorial/estimate-pi-in-parallel/estimate-pi-in-parallel.js. [Online; accessed 22-July-2018].
[11] 2018. Node.js v10.4.1 Documentation - Child Processes. https://nodejs.org/api/child_process.html. [Online; accessed 20-July-2018].
[12] 2018. Node.js v10.4.1 Documentation - Cluster. https://nodejs.org/api/cluster.html. [Online; accessed 20-July-2018].
[13] 2018. WebWorker Threads. https://www.npmjs.com/package/webworker-threads. [Online; accessed 7-July-2018].
[14] 2018. WebWorker Threads - Lightweight Web Worker API Implementation with Native Threads - Repository. https://github.com/audreyt/node-webworker-threads. [Online; accessed 13-November-2019].
[15] 2018. WebWorker Threads - WebWorkerThreads.cc. https://github.com/audreyt/node-webworker-threads/blob/dfbbe267b83224e0d7eb82957fe07dcdc2122464/src/WebWorkerThreads.cc. [Online; accessed 13-November-2019].
[16] 2019. Node.js C++ Codebase. https://github.com/nodejs/node/blob/master/src/README.md. [Online; accessed 19-December-2019].
[17] 2019. Node.js v12.1.0 Documentation - Events. https://nodejs.org/dist/v12.1.0/docs/api/events.html#events_class_eventemitter. [Online; accessed 11-November-2019].
[18] 2019. Node.js v12.1.0 Documentation - Worker Threads. https://nodejs.org/dist/v12.1.0/docs/api/worker_threads.html. [Online; accessed 11-November-2019].
[19] Jay Conrod. 2014. A tour of V8: Garbage Collection. http://jayconrod.com/posts/55/a-tour-of-v8-garbage-collection. [Online; accessed 6-July-2018].
[20] Libuv contributors. 2019. Libuv Documentation Release 1.33.1. https://buildmedia.readthedocs.org/media/pdf/libuv/stable/libuv.pdf. [Online; accessed 18-November-2019].
[21] Jamie Davis. 2017. WebWorker Threads - Developer guide. https://github.com/audreyt/node-webworker-threads/wiki/Developer-guide. [Online; accessed 14-November-2019].
[22] OpenJS Foundation. [n.d.]. Don't Block the Event Loop (or the Worker Pool). https://nodejs.org/uk/docs/guides/dont-block-the-event-loop/. [Online; accessed 13-December-2019].
[23] OpenJS Foundation. 2020. Node.js. https://nodejs.org/en/. [Online; accessed 17-August-2020].
[24] Joyee Cheung Alibaba Cloud(Alibaba Group). 2017. Are your V8 garbage collection logs speaking to you? https://www.slideshare.net/NodejsFoundation/are-your-v8-garbage-collection-logs-speaking-to-youjoyee-cheung-alibaba-cloudalibaba-group. [Online; accessed 8-January-2020].
[25] Jakob Gruber. 2019. JIT-less V8. https://v8.dev/blog/jitless. [Online; accessed 12-December-2019].
[26] Anna Henningsen. 2018. Worker: Initial Implementation #20876. https://github.com/nodejs/node/pull/20876. [Online; accessed 15-November-2019].
[27] Docker Inc. 2018. Docker. https://www.docker.com/. [Online; accessed 07-July-2018].
[28] Dmitry Namiot and Vladimir Sukhomlin. 2015. JavaScript Concurrency Models. *International Journal of Open Information Technologies* 3, 6 (2015), 21–24.
[29] Lionel Nkenyereye and Jong-Wook Jang. 2016. Performance Evaluation of Server-side JavaScript for Healthcare Hub Server in Remote Healthcare Monitoring System. *Procedia Computer Science* 98 (2016), 382–387.
[30] Maria Patrou, Kenneth B Kent, and Michael Dawson. 2019. Scaling Parallelism Under CPU-Intensive Loads in Node.js. In *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, 205–210.
[31] Stefan Tilkov and Steve Vinoski. 2010. Node. js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing* 14, 6 (2010), 80–83.
[32] Devesh Tiwari and Yan Solihin. 2012. Architectural characterization and similarity analysis of sunspider and Google's V8 Javascript benchmarks. In *Performance Analysis of Systems and Software (ISPASS), 2012 IEEE International Symposium on*. IEEE, 221–232.
[33] ukrbublik. 2019. shm-typed-array. https://www.npmjs.com/package/shm-typed-array. [Online; accessed 19-November-2019].
[34] Michael Lippautz Ulan Degenbaev and Hannes Payer. 2017. Orinoco: Young Generation Garbage Collection. https://v8.dev/blog/orinoco-parallel-scavenger. [Online; accessed 18-December-2019].
[35] V8. 2018. Concurrent Marking in V8. https://v8.dev/blog/concurrent-marking. [Online; accessed 18-December-2019].
[36] V8. 2019. 3.11.10(node0.8.28) - v8::Isolate Class Reference. https://v8docs.nodesource.com/node-0.8/d5/dda/classv8_1_1_isolate.html. [Online; accessed 25-November-2019].
[37] V8. 2019. Getting started with embedding V8. https://v8.dev/docs/embed. [Online; accessed 18-December-2019].
[38] Jeffrey S Vetter and Michael O McCracken. 2001. Statistical scalability analysis of communication operations in distributed applications. In *ACM SIGPLAN Notices*, Vol. 36. ACM, 123–132.
[39] Jeffrey S Vetter and Andy Yoo. 2002. An empirical performance evaluation of scalable scientific applications. In *Supercomputing, ACM/IEEE 2002 Conference*. IEEE, 16–16.
[40] Rafael J. Wysocki. 2017. intel_pstate CPU Performance Scaling Driver. https://www.kernel.org/doc/html/v4.12/admin-guide/pm/intel_pstate.html. [Online; accessed 19-December-2019].
[41] Jiapeng Zhu, Panagiotis Patros, Kenneth B Kent, and Michael Dawson. 2018. Node. js scalability investigation in the cloud. In *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering*. IBM Corp., 201–212.
[42] Yuhao Zhu, Daniel Richins, Matthew Halpern, and Vijay Janapa Reddi. 2015. Microarchitectural implications of event-driven server-side web applications. In *Microarchitecture (MICRO), 2015 48th Annual IEEE/ACM International Symposium on*. IEEE, 762–774.

The Weakest Link: Revealing and Modeling the Architectural Patterns of Microservice Applications

Vladimir Podolskiy
v.podolskiy@tum.de
Chair of Computer Architecture & Parallel Systems
Technical University of Munich
Germany

Maria Patrou
maria.patrou@unb.ca
Faculty of Computer Science
University of New Brunswick
Canada

Panos Patros
panos.patros@waikato.ac.nz
Department of Software Engineering
University of Waikato
New Zealand

Michael Gerndt
gerndt@in.tum.de
Chair of Computer Architecture & Parallel Systems
Technical University of Munich
Germany

Kenneth B. Kent
ken@unb.ca
Faculty of Computer Science
University of New Brunswick
Canada

ABSTRACT

Cloud microservice applications comprise interconnected services packed into containers. Such applications generate complex communication patterns among their microservices. Studying such patterns can support assuring various quality attributes, such as autoscaling for satisfying performance, availability and scalability, or targeted penetration testing for satisfying security and correctness. We study the structure of containerized microservice applications via providing the methodology and the results of a structural graph-based analysis of 103 Docker Compose deployment files from open-sourced Github repositories. Our findings indicate the dominance of a power-law distribution of microservice interconnections. Further analysis highlights the suitability of the Barabási-Albert model for generating large random graphs that model the architecture of real microservice applications. The exhibited structures and their usage for engineering microservice applications are discussed.

CCS CONCEPTS

• **Computer systems organization → Cloud computing**; • **Software and its engineering → Extra-functional properties**.

KEYWORDS

cloud-native application, microservice, software vulnerability, application topology

ACM Reference Format:
Vladimir Podolskiy, Maria Patrou, Panos Patros, Michael Gerndt, and Kenneth B. Kent. 2020. The Weakest Link: Revealing and Modeling the Architectural Patterns of Microservice Applications. In *Proceedings of 30th International Conference on Computer Science and Software Engineering (CASCON'20).* IBM, USA, 10 pages.

CASCON'20, November 10–13, 2020, Toronto, Canada

1 INTRODUCTION

Cloud-native applications comprise containerized microservices, each implementing a narrow part of the application's functionality to enable fine-grain elasticity. However, finding the right number of containerized microservice instances to guarantee quality of service, reduce resource consumption and identify bottlenecks is not trivial: communication between microservices can happen in a number of ways depending primarily on the application's topology.

Analyzing the structure of real microservice applications unveils chains of microservices (in a producer-consumer relationship) that utilize various communication protocols with as many as 100-300 services [7]. When scaling a particular logical service in such a chain, one may face the necessity of cascading capacity changes for the downstream services to avoid such services becoming a new bottleneck [19]. Knowing the topology of a microservice application could help identify such a bottleneck service, *the weakest link*, in advance. This allows predictive scaling that can dynamically meet demand [26] and protect against malicious entities exploiting *a weakest link vulnerability*, which can happen via a targeted denial of service attack affecting the availability of the cloud application.

Besides scaling and security, knowing a microservice topology can assist in assuring other software quality attributes. For example, realistic benchmarks can be created using the service topology as a generic template. Also, deployment based on the weakest link-services to assist the weakest one and determining the application capacity can lead to better performance.

The lack of publicly available industry-scale microservice applications precluded the research of this type of applications [7]. The study of public code repositories allows us to overcome this challenge to an extent as individuals and companies tend to open source their production code or community projects. The strong positive correlation between the number of employees and number of services supported in an application [7] allows us to assume that it is more likely to find an application encompassing a high number of services and with a more complex topology in the public repository of a large company, such as Google or Uber, than in the repository of a recent startup or an individual.

Studying the structure of open source microservice applications can disclose common topological patterns. Generating versatile real-like application structures from these patterns can further be used to assemble microservice applications of similar structure but with a larger size to enable practice-relevant research or realistic stress-testing for such applications. We focus on the architectural patterns of microservice applications contributing in:

- Performing an empirical study of the structure of 103 microservice applications available on Github.
- Modeling the structure of the over-represented microservice application type with a power-law distribution of vertices' degree using random graph models, which we evaluate.
- Outlining an overall methodology for performing such empirical studies, including the identification of an appropriate random graph generation model and tuning its parameters.

The paper is organized as follows: background and related work in Sections 2 and 3; architectural pattern inspection of the applications in Section 4; modeling of the structure of applications with power-law service degree distribution in Section 5, observations in Section 6 and conclusion and future work in Section 7.

2 BACKGROUND

A microservice implements a limited functionality, is independently deployable and often communicates with other microservices via the network. OS-level virtualization with containers allows one to implement microservices easily—the software developer needs to add the necessary libraries and the software to the container image, which can be used to deploy multiple containers implementing the same function. In a microservice application that follows a Service-Oriented Architecture (SOA) software design style, the communication between the microservices is usually done via API calls over the network—this supports loosely coupled applications and allows fine-grained application elasticity.

Microservice applications tend to serve users' requests, a pattern commonly used in web-shops and online-portals, because this architecture addresses multiple requirements: the response time of a microservice application deployed on the cloud can be relatively short and predictable by scaling individual microservices; high availability is ensured by negligible microservice deployment times; and there are multiple orchestration tools available for microservice applications, which make management and autoscaling easy tasks (e.g., Docker Swarm and Kubernetes) [16].

The application deployment in Docker Swarm requires the use of a Compose file [18] in YAML [2] standard, which describes the components of the application and their interconnections at a software architecture and deployment level. Docker Swarm initializes the cluster with the container-services described in the YAML file. The file includes configuration settings for each service that resides in a container, including the container image, which has the executable code, and its dependencies with other containers that affect the order of starting and stopping the services. Furthermore, information on the cluster's networking for intercommunication and reachability among containers and their data storage is defined.

While in Docker Swarm, applications are organized into containers, Kubernetes leverages Pods. Each Pod has one or multiple containers, while groups of pods are deployed to create an application on a (cloud) cluster [27].

2.1 Graph Theory Essentials

A graph is a discrete mathematical abstraction that encompasses a set of objects (*vertices* of a graph) and a set of relations between these objects (*edges* of a graph). If the set of vertices is denoted by V and the set of edges is denoted by E with an edge between i^{th} and j^{th} vertices being e_{ij}, then a graph can be denoted as an ordered pair $G = (V, E)$. Graphs are usually depicted with circles being the vertices and lines being the edges; if a graph is directed (the order of vertices in edge matters), arrows are used instead of simple lines. A graph can be quantified by parameters. The most basic quantification is through the number of vertices, $|V|$, and the number of edges, $|E|$. In addition, each vertex could be quantified by the number of edges that connect to it; this parameter is called the *degree* of a vertex, $deg(v)$. In a directed graph one can further divide the notion of degree into outdegree, i.e.,the number of edges that start at this vertex, and indegree, i.e., the number of edges that end at this vertex. The degree sum for the undirected graph could be computed as $\sum_{v \in V} deg(v) = 2|E|$. The degree of a vertex is a primary characteristic of structural patterns in the graph as it can be used to describe the connectivity of a particular part of a graph by relating vertices to edges in a quantifiable way. Thus, graph theory is used in the paper as a formalism to analyze the structure of microservice applications.

2.2 Network Theory Essentials

Network theory emerged to address the complexity and vulnerability of real-world structures like power grids or the Internet [5]. In essence, modelling real a structures, a network is a graph with labelled vertices and/or edges.

It is often necessary to understand which nodes in the network are more important than the others. The importance could be denoted differently, but the most common way is to associate the number of connections with a node's importance. The identification of such nodes is addressed by *centrality indices* that are computed differently [9]. Degree centrality is one of the simplest centrality measures and is defined as the number of links incident upon a node, thus the degree centrality of a vertex v is $C_D(v) = deg(v)$; it characterizes the immediate importance of the node.

Degree distribution is a probability distribution of degrees in the network used to describe the whole network. Degree distribution shows how often nodes with a particular degree are encountered—different degree distributions correspond to different structures. For example, if the degree distribution has a long tail for higher degrees, then the network contains only a few nodes of high importance, i.e., numerous connections with other nodes. This fact can have significant implications in such cases as developing a network structure that is resilient to cyber attacks. Hence, certain structural properties of the network can be conveyed with the degree distribution.

The degree distribution of a network can be approximated by a formula; this allows the in-depth study of the network's properties. For example, one of the most common types of networks is a scale-free network with the probability distribution described roughly by $P(d) \propto d^{-\alpha}$, where the fraction of nodes with d connections is

defined as $P(d)$ and drops exponentially with the growth of the degree (α is usually between 2 and 3). Various models exist to describe the properties of the networks and to generate new ones. In particular, scale-free networks are best described by the Barabási-Albert (BA) model that uses a preferential attachment method to generate networks with a power-like degree distribution [4].

We employ degree distribution on the microservice connectivity as described by the configuration files. The metric exposes the connectivity across the microservices, thus revealing the application's structure model. We conduct an analysis of the microservice applications that allows us to use random graph models to generate networks with realistic microservice structural properties.

3 RELATED WORK

Despite the absence of work devoted to the study of the structural aspects of microservice applications, the importance of such research is recognized in the literature [14].

Contributions to the study of application's structure were made for conventional multi-tier application architectures, such as ones with a front-end, an application service and a database. The necessity to incorporate such knowledge to identify application bottlenecks was recognized by Malkowski et al. as the result of experimental studies of N-tier applications using the RUBiS and RUBBoS benchmarks [24]. Wang et al. approached the challenge of detecting the transient bottlenecks in multi-tier applications that contribute to the latency long-tail problem in clouds via elaborate load-throughput analysis on multiple tiers of application [32, 33]. Liu et al. applied queuing network theory-based application modeling to wide-spread 3-tier web-applications to derive accurate predictions for response time and throughput [23]. Workload scaling as a method to scale multi-tier cloud applications via replicating the processing of the same request and sending the results of the fastest VM to the user was proposed by Pérez at al. [28]; the same work marks application topology and tier-specific workload scaling models as a research challenge. sPARE is the first known partial replication system that takes into account the structure of a multi-tier application to coordinate the replication levels on all tiers [6].

Similar to us, Márquez et al. performed an empirical study on scalability aspects of microservice-based applications by investigating 30 open-sourced projects. They analyzed three types of configuration files found in the projects: YAML files for Docker Compose, POM files for Apache Maven and Gradle files (build.gradle). Their main focus was to answer research questions towards scalability using their pattern language that focuses on scalability dimensions they have previously identified. Their goal is to identify the frameworks that meet the scalability dimensions and provide recommendations on microservice architecture [25].

In contrast, our study focuses on applying graph theory to microservice-based applications' structures. Every service, which is defined in the Compose file, is treated as a node and keywords that reveal their dependencies are used to extract the connections among services. We identify and generate models that best fit the structures and discuss their potential usage in software engineering. Any observations on software qualities, such as scalability, security etc., are made based on the architecture of the service interconnection and can be used to improve software quality. Finally, unlike other empirical studies who analyze source code, such as [11], our focus is entirely on the interconnections at the software architecture/component level.

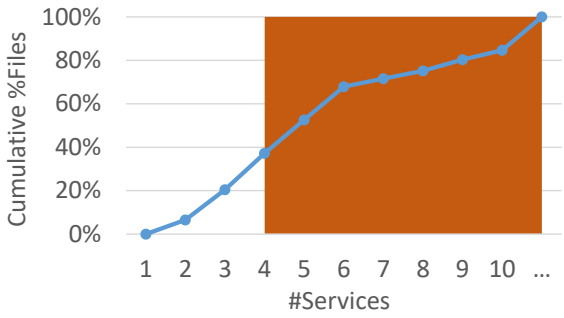


Figure 1: Cumulative distribution of configuration files per number of services they declare. The shaded area designates the files having four or more services for further analysis.

4 ARCHITECTURAL PATTERN INSPECTION

Various public Github repositories of IT companies, organizations and individuals were explored manually to obtain their configuration YAML files. The files were processed to reveal the type of services and how they tend to be interconnected. A statistical analysis with graph and network metrics was applied to find how common distributions model real-world application structures.

4.1 Dataset

Although following a manual data collection is a limitation of the study, it was adopted since the automatic exploration of Github repositories' excerpts available on Kaggle[1] would result in meaningless sample pet-projects polluting the results and thus, biasing the resulting structural models. The exploration resulted in a collection of 137 Docker Compose configuration files taken from 107 Github repositories, which we made available [1]. The collected configuration files represent a variety of applications, including web-shops and web-portals, cloud platforms for IoT, technology stacks, etc. Their version distribution was as follows: 18 files were version 1.0; 66 were version 2.x; and 53 were version 3.x.

These YAML files define the start up sequence of microservices via special keywords, such as *depends_on*, *links*, or *external_links*[2]. We used this formal specification to create service-dependency graphs. However, to ensure our dataset contained complex-enough points, it was decided to exclude those that had three or fewer connected microservices (Figure 1).

After the above filtering was processed, 103 Compose configuration files were selected and analyzed for their microservice topology. Based on these configuration files, the results show that at least 26% of the applications contain more than eight microservices. For each of the 826 services defined, we extracted the number of ports exposed to other services and to outside clients, the number of persistent volumes, the number of services a service depends on and the number of services that are depended upon a service.

[1] https://www.kaggle.com/github/github-repos
[2] https://docs.docker.com/compose/compose-file/#links

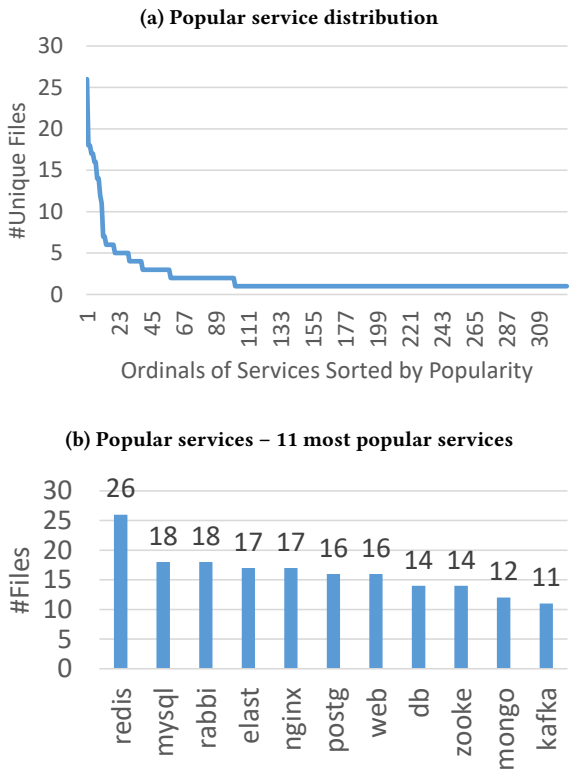


Figure 2: Service popularity in the filtered dataset

General observations were made on the filtered service data using the Pearson Coefficient to show the relationship among the metrics (as values move away from zero, the statistical relationship among the metrics is stronger):

- No large correlations were measured among the extracted metrics. A coefficient of -0.19 was recorded between the numbers of depending and depended services, which indicates the presence of leaves and roots in the tree-structure of an application; and a coefficient of 0.17 for the number of volumes and dependent services, which indicates that persistence-related functionality was less frequent on leafs.
- No trend was observed for an increased number of microservices being used as versions progressed. The correlation between number of services and file version was 0.06.
- Certain services were more popular than others in a way that resembles a power law distribution (Figure 2). In particular, popular services used in at least ten different files revealing various databases, cloud elasticity services and load balancers—not surprising, given the types of applications commonly deployed on the cloud.

4.2 Microservice Degree Distribution

Next, an adjacency matrix representation of the underlying directed graph was produced for each of the deployment files of the dataset. The matrices were analyzed to identify patterns in the structure of the microservice applications using the observed degree metric, i.e., the number of services that are connected to a service.

4.2.1 Degree Distributions. Visual inspection identified three prevalent degree distribution types in the dataset:

- **Uniform distribution**. The number of vertices N for a degree d in a range $[a, b]$ is *const* and 0 otherwise.
- **Power law (Pareto) distribution**. The number of vertices N for a degree d is $N \propto d^{-\alpha}$.
- **Normal distribution**. The number of vertices N for a degree d is $N \propto e^{-\frac{(d-\mu)^2}{2\sigma^2}}$.

Furthermore, an automated machine learning-based approach, agglomerative clustering, that leveraged hierarchical clustering, also identified three distinct clusters in our dataset, confirming our visual inspection findings. Each application was assigned a vector $v = (p_0, p_1, ..., p_{24})$ that represents the probability p_i of a service being connected by i services in a specific application, where 24 is the maximum observed number of services that a service is connected by. Each connection is represented as a directed line from one service to the others that depends, as indicated by the Compose file's keywords: *links*, *external_links* and *depends_on*. As an example, consider Figure 4C, which shows the graph of a batch scheduling system by Yelp with four services and three connections. The calculated service dependence probability vector is: 0.25,0.75,0,0,..,0, where as 25% of the services (one) have no incoming line to them and 75% (three) have one.

The service dependence probability vectors were averaged, clustered pairwise and recursively based on the smallest Euclidean distance among their probabilities. The dendrogram of the results is displayed in Figure 3 and shows the three clusters that the samples were automatically grouped into. When averaging out all members of each of the clusters, the aggregate distributions (which are omitted for brevity) appear to be primarily following the power law with some other distribution added on top.

However, every cluster shows different properties. Cluster 1 has more than half (57%) of its services with one service to be depended upon and 28% zero. Thus, the majority of the services were acting as leaf-services and less than a third as roots. Cluster 2 has 71% of the application's services with no dependencies, indicating minimum dependency among the majority of the services, while the dependencies should be concentrated to a few services. Cluster 3 is the most representative in the dataset, comprising 44% of the applications. It indicates that 40% of the services had zero dependencies, 13% and 27% had one and two, respectively. The cluster shows more services with zero dependencies that Cluster 2, revealing fewer independent services and more services with less than two dependencies. All three clusters show that dependencies among services do not exceed two connections for most services: finding a service that depends on more than two services is rare.

4.2.2 Degree Distribution Methodology for Small Graphs. With a maximum of 24 vertices in the largest microservice application, determining the form of the degree distributions with statistical tests can be inaccurate [20]. A graph may be attributed to several distribution types. To improve the quality of such tests, we devised an appropriate testing technique.

The proposed approach combines *conventional statistical distribution tests* with fallback *heuristics*. Preliminary tests on randomly generated distributions showed high inaccuracy of statistical tests

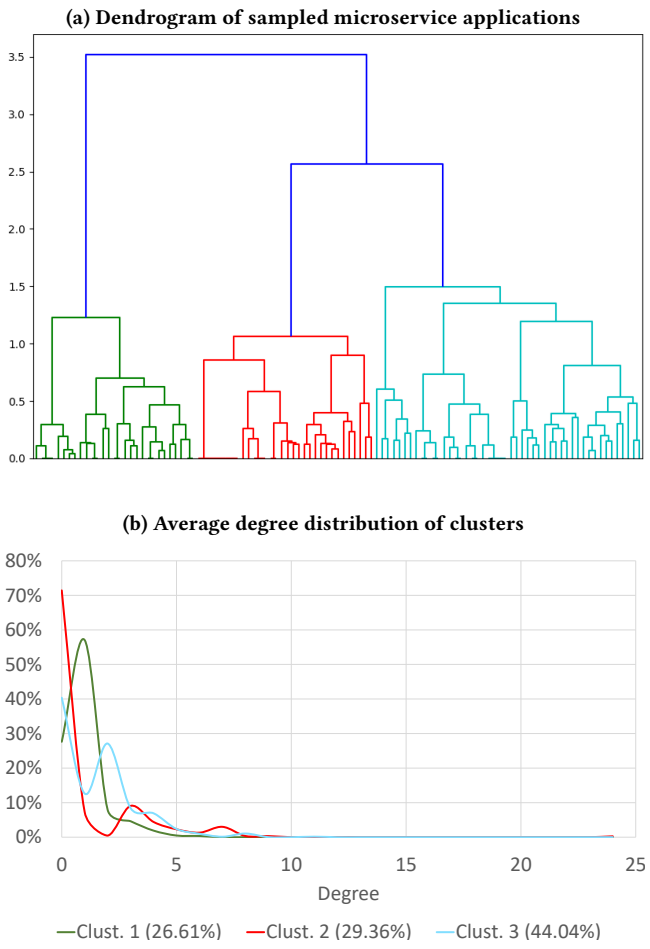


Figure 3: Unsupervised learning of clusters of distributions in the dataset.

for a number of samples less than six; hence we applied fallback heuristics when graphs had fewer than six vertices or when the corresponding statistical test could not be applied to the degree distribution. Although the merger of the statistics and heuristics-based analysis approaches is limited compared to the pure statistics, statistics offers relatively few methods available for the small population sizes. Omitting the small applications (between 3 and 6 services) from the consideration would have added a significant flaw to the research since there exist industry applications of such "small" sizes, e.g. at companies where IT plays only the support role for the operations [7]. The designed heuristics are as follows:

Uniform distribution heuristic. A small number of distinct degrees in graphs makes the direct application of uniform distribution tests impractical. However, it is possible to transform the data such that statistical testing would provide meaningful results. First, the initial degree distribution is transformed into a histogram. Following, the Pearson's chi-squared test is applied to test the degree distribution based on a Monte Carlo test with 500 replicates [17]. The value of 500 replicates was determined by conducting multiple tests on randomly generated distributions. The fallback heuristic for uniform distribution checks the single outcome not covered by the statistical test: when all vertices have the same degree.

Power law distribution heuristic. The Kolmogorov-Smirnov test was used to determine if the degree distribution of a graph is close to a power law (Pareto) distribution. Computed parameters of power law distribution allow us to determine if the fallback test should be invoked. Usually, it is necessary for borderline graphs with 6–7 vertices. The fallback heuristic computes the mean degree and checks if the number of vertices with a degree lower than the computed mean is higher than the number of vertices with a degree higher than the mean:

$$|\{v_i | d \leq \mu\}| - |\{v_j | d > \mu\}| > T$$

Based on the threshold T for such a comparison, more or fewer cases can be classified as following the power law; the threshold values 1 or 2 were good for the collected dataset.

Normal distribution heuristic. To determine if the degree distribution of a graph follows a normal distribution, the Shapiro-Wilk test of normality [29] was used. This test was shown to be more powerful when testing for normality in comparison to Kolmogorov-Smirnov [31]. Its associated fallback heuristic checks 1) if the most frequent degree in a graph d_f is between the minimal (d_m) and maximal (d_M) degrees, and 2) if the number of vertices with degrees higher than the most frequent degree and the number of vertices with degrees lower than the most frequent degree are almost equal (discrepancy by a threshold $T = 1$ was allowed):

$$\left(d_m < d_f < d_M\right) \wedge \left(|\{v_i | d \leq d_f\}| - |\{v_j | d > d_f\}| \leq T\right)$$

4.3 Service Degree Distribution Analysis

We compared the known distributions: power-law, uniform and normal with the application topologies using the statistical tests and the heuristics described above. Table 1 shows the applications that fit in the corresponding distribution type under the graph-based threshold parameters. To account for the limitations of the statistical analysis with fallback heuristics, we adapted the distribution types names accordingly. Both absolute numbers and percentages in dataset are reported. The *Total* column presents the applications that have the distribution type, while the *Pure* column shows the applications that fit only in the underlying distribution type.

Microservice applications with the power law degree distribution of the underlying structure graph prevail. The applications with such a degree distribution cover around 87% of the whole data set with a loose threshold of 1 for the fallback heuristic and around 78% with a tighter threshold of 2. The uniform and normal distribution cases amount to only around 42% and 19% of cases correspondingly. Considering only the cases that were associated with a single distribution type, a similar picture of power law distribution emerges, being the most frequent with around 47% of all the cases, and followed by the uniform distribution with around a 30%-wide gap. For the small number of unique degrees, the uniform degree distribution might be overrepresented. Hence, for the examined dataset, the dominance of the power law-like distribution becomes even more apparent. Samples of the discussed graphs can be found in Figure 4.

Distr. Type	Threshold = 1[a]		Threshold = 2[a]	
	Total[b]	*Pure*[c]	*Total*[b]	*Pure*[c]
Skewed	90 (87.4%)	48 (46.6%)	80 (77.7%)	42 (40.8%)
Near-uniform	43 (41.8%)	11 (10.7%)	43 (41.7%)	14 (13.6%)
Central	20 (19.4%)	0 (0.0%)	20 (19.4%)	0 (0.0%)
Other	- (-%)	2 (1.9%)	- (-%)	8 (7.8%)

[a]Threshold is set for the fallback test.
[b]Positive outcomes for other types are possible.
[c]Only negative outcomes for other distribution types.

Table 1: Degree Distribution types

Table 1 shows several distributions that are different from those tested. The thresholds increase from one to two for the power-law heuristic test yields an increase in the number of unclassified cases by six, which might be hybrids between the skewed and some other types . The two other cases should be quite different from the power law distribution. Indeed, these two examples show the prevalence of vertices of a higher degree in comparison to vertices of a lower degree; this type of distribution could be described as $N \propto e^d$.

The main outcome of the analysis is that most applications have a structure of a *scale-free network* [3]. The skewed degree distribution with a long tail implies a presence of services that have significantly more connections than others; there are at least several types of such microservices, e.g., PostgreSQL, Zookeeper, RabbitMQ and Elasticsearch. This is not surprising as these microservices implement common functions, such as logging, configuration management, message brokering and data storage. This also means that most microservice applications *tend to form bottlenecks* and *are susceptible to targeted attacks.*

5 ARCHITECTURAL PATTERNS MODELING

Our dataset provides hints on how cloud-native applications tend to be structured. Understanding these tendencies can result in models that capture structural properties of real-world applications for further structure-driven capacity balancing research. To evaluate what types of models better fit our data, we use several models that can generate random graphs. Then, we compare the similarities between the real and randomly generated graphs to determine how well each model (and its parameters) fits for the empirically collected data. The study was conducted for 42 microservice applications, which were attributed to the power law degree distribution with the strictest conditions according to Table 1.

5.1 Structural Models Identification

A large percentage of the applications exhibited skewed degree distribution. Thus, five random graph models, which we believe describe applications that model scale-free networks, are chosen. Distance metrics are computed for every application and the results for each each metric reveal the model types that best describe the majority of the applications.

5.1.1 Considered Models. The following models were considered to identify the architectural patterns:

(1) **Erdős-Rényi random graph (ER)** in its $G(n, m)$ and $G(n, p)$ forms was used as a baseline [13]. The number of vertices n and the number of edges m are equal to that of the application graph, whereas the probability of an edge to be included in the generated graph p varies throughout the tests.

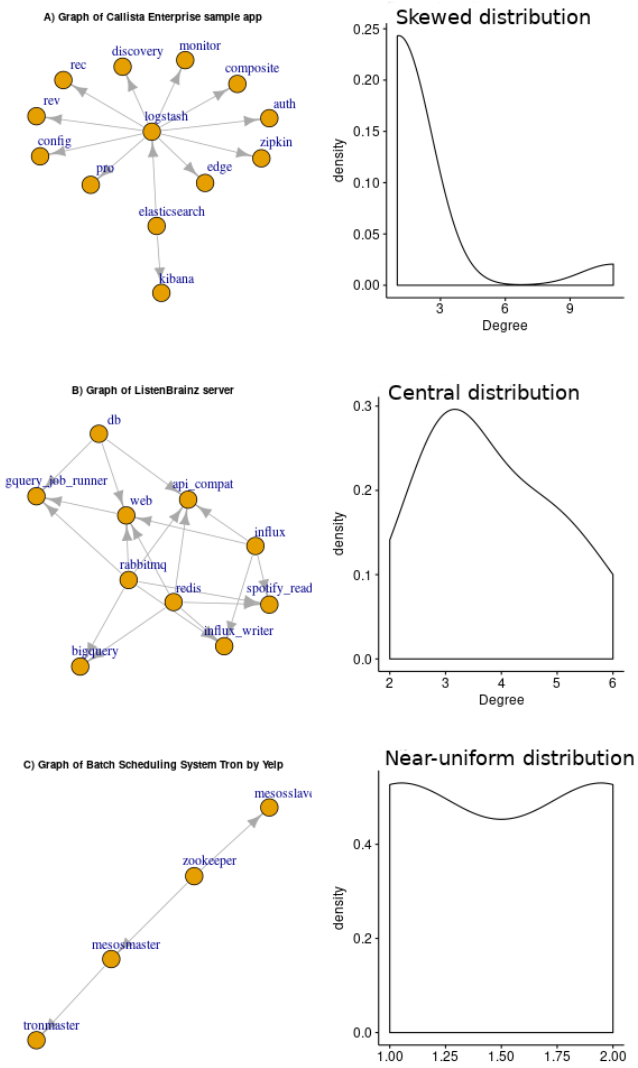


Figure 4: Samples following the proposed distributions.

(2) **Barabási-Albert (BA)** with the varying parameters: power of the preferential attachment, number of edges to add per timestep, attractiveness of vertices without edges [4];
(3) **Forest Fire (FF)** with the varying parameters: forward burning probability, backward burning ratio, number of ambassador vertices [21];
(4) **Fitness Score (FS)** that generates a graph with edge probabilities proportional to node fitness scores with the power used to generate the vector containing the fitness of each vertex as the only varying parameter [15];
(5) **Simple Power Law (SPL)** that generates a graph with a desired power law degree distribution varying only the in-degree and outdegree power law exponents [10, 15].

We used the the R package **igraph** [12] to implement these models.

5.1.2 Approach. The identification of a structural model for a single application graph starts with the generation of multiple random graphs (300) for each discussed model type (five types) with all

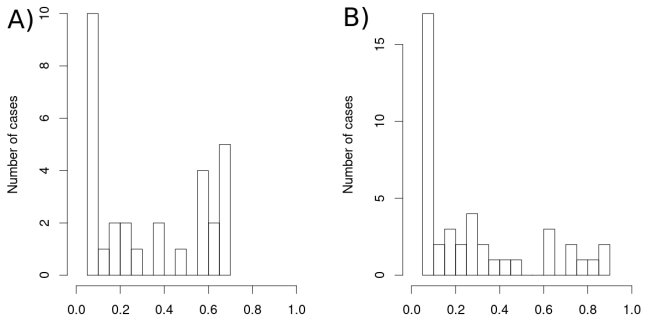


Figure 5: Distribution of the cases with the optimal parameters' values over the values of *Power of preferential attachment* (BA)

possible combinations of model parameter values *from the meaningful subspace* determined by the preliminary experiments. Such parameters as the number of vertices/edges are taken directly from the application graph.

All distance metrics are computed, for each pair of an application graph with one of the generated random graphs. Each distance metric (e.g., Hamming) for the given model type (e.g., ER) and the current set of model parameters (e.g., $n = 17$ and $p = 0.5$) is computed as the average of all pairwise distance values between the application graph and the random graphs generated from that model. Averaging ensures the stability of the results.

We tuned the model parameters via running our approach with different parameter limits multiple times. We adjusted the limits of each parameter by studying the form of the distribution of the cases with the minimal value of a distance metric over each parameter's values. If the histogram is skewed, it might be necessary to increase the upper boundary on the parameter and continue the tuning.

An experiment with 30 application graphs showed that the upper bound on the BA model's parameter *Power of preferential attachment* originally set to values from the interval $[0.05, 0.7]$ was too small as the number of cases with the optimal parameters' values increased to the end of the interval (see plot **A** in Figure 5). With the upper bound of the same parameter increased to 0.9 for the experiment involving the full set of 42 application graphs, we did not observe any increase in the number of cases towards the end of the interval (see plot **B** in Figure 5). Hence, with an exhaustive search being unfeasible, the parameters' bounds tuned with this method, cover random graphs models close to real application graphs.

5.1.3 Test Settings. The pilot test covered 30 applications. Then, we conducted three tests on 42 applications: The first returned results for all 42 applications, the second returned results for 41 applications, and the last one only for 36 applications. The last test was conducted for the case of undirected graphs; the partial results returned are due to particular distance metrics relying in their computation on matrix invertibility, which is not always the case for the given data set. Further, we discuss test settings and results of the second experiment as it covers all 42 applications. The bounds on the parameters' values are given in Table 2. The number of random graphs generated for each model type and each parameters values combination is 300. The number of vertices for each experiment was taken directly from the application graph.

Model type	Parameter	Start[a]	End[a]	Step[a]
ER $G(n, m)$	Edges number	-	-	-
ER $G(n, p)$	Edge inclusion prob.	0.05	0.65	0.05
BA	Power of the preferential attachment	0.05	0.90	0.05
	Number of edges to add per timestep	1	2	1
	Attractiveness of vertices with no edges	0.01	3.5	0.01
FF	Forward burning probability	0.05	0.65	0.05
	Backward burning ratio	1	3	1
	Number of ambassador vertices	1	2	1
FS	Power to generate fitness vector	2	3.5	0.1
SPL	Power law expon. of the out-degree distr.	2	3	0.1
	Power law expon. of the in-degree distr.	2	3	0.1

[a] "-" value is taken from the application graph.

Table 2: Studied parameter values

Distance type	Random Graphs Model Types					
	ER $G(n, m)$	ER $G(n, p)$	BA	FF	FS	SPL
Degree Centrality	0 0.00%	1 2.38%	**29** **69.05%**	3 7.14%	0 0.00%	*9* *21.43%*
Closeness Centrality	0 0.00%	3 7.14%	**32** **76.19%**	3 7.14%	1 2.38%	3 7.14%
Between. Centrality	0 0.00%	**26** **61.90%**	*11* *26.19%*	0 0.00%	4 9.52%	1 2.38%
Edge Difference	0 0.00%	**32** **76.19%**	*9* *21.43%*	0 0.00%	1 2.38%	0 0.00%
Graph Diffusion	0 0.00%	1 2.38%	**38** **90.48%**	0 0.00%	2 4.76%	1 2.38%
Hamming	0 0.00%	**32** **76.19%**	*10* *23.81%*	0 0.00%	0 0.00%	0 0.00%

Table 3: Cases with minimal network distance

5.1.4 Results. Network distance metrics were used to determine which one of the studied model types allows us to generate random graphs that are close to the real applications. Each metric captures different structural properties, e.g., *Degree Centrality-based distance metric* tends to mark graphs having close degree distributions as similar, whereas *Edge Difference distance metric* is small for pairs of graphs that have similar connections. These differences between metrics become apparent when looking at Table 3. Here, each row corresponds to one of the network distance types, and each column contains the number and percentage of cases with the minimal distance to the random graphs generated with the model type specified in the column header.

Since the distances were averaged over 300 generated graphs for each selected application graph from the dataset, the analysis of the cases with larger network distances is not provided as the observed gap between the model exhibiting the minimal distance and the model with the second smallest distance was higher than what would be meaningful to consider.

BA excels at capturing structural characteristics used for comparison by *Degree Centrality*, *Closeness Centrality*, and by *Graph Diffusion distance*. ER in its $G(n, p)$ form shows good results for *Betweenness Centrality*, *Edge Difference distance*, and *Hamming distance*. However, BA is still in second place with 11, 9, and 10 cases out of 42 for these distance types correspondingly. In contrast, ER in its $G(n, p)$ form has less than 6 cases in total marked as similar to real graphs by *Degree Centrality*, *Closeness Centrality*, and *Graph Diffusion distance*. Hence, BA captures the properties of the microservice applications structure nicely.

Recalling that the 42 application graphs selected for this study exhibited power law-like degree distribution, we might find it significant that for some metrics, numerous cases result in the ER in its $G(n, p)$ form. Essentially that means that a combination of BA with ER in its $G(n, p)$ form could capture the structural properties of microservice applications better than each of these model types individually. Such combinations can be enabled by generative models of graphs acquired with machine learning techniques [8, 22].

Nevertheless, further application-wise study of minimal network distances demonstrates that *Edge Difference distance* values for different models vary weakly; in 32 cases this type of distance demonstrated the smallest variability when computed for different models. Thus, we select the BA type as the best representative type for microservice application graphs.

5.2 Structural Model Generation

We then proceeded to create models that best fit the structures of our dataset. Studying the parameters of the BA model leading to minimizing the network distances shows that the change only in two parameters influences how close the generated graph is to the real one. These parameters are *power of preferential attachment*, α, and *attractiveness of vertices with no edges*, a. According to BA, a single vertex is added to the graph at each time step; a new vertex is attached to old vertices with one or more edges. The probability of i^{th} vertex to be chosen is given by $P_i = d_i^{\alpha} + a$, where d_i is the in-degree of this vertex. As we see, higher values of α favor vertices with more connections, whereas higher a values give vertices with no connections a chance to establish new ones.

Study of parameters α and a distributions for graphs with minimal network distances from the Subsection 5.1.4 allowed us to find two perspective intervals for each of these: $\alpha \in [0.01; 0.10] \cup [0.80; 1.00]$, $a \in [0.00; 0.05] \cup [3.00; 3.50]$. For each interval marked either as *LOW* or *HIGH*, a value close to its middle was selected, then four possible combinations of these values were acquired to generate example random graphs according to BA. Parameter *edges to add per time step* was set to 1. Generated samples with 18 vertices are shown in Figure 6.

Visual study shows that sample **B** in Figure 6 corresponds to the applications that rely on the common logging service, whereas sample **C** represents an application with several auxiliary services used, e.g., to maintain configurations. Sample **D** in that sense is close to applications organized in the conventional multi-tier fashion. Sample **A** in Figure 6 also finds peers among microservice applications—these exhibit highly-centralized hierarchical architectures with most of the services using the configuration service.

6 DISCUSSION

The above results lead to several observations on the structure of microservice applications and how it could be used to assure software quality attributes.

6.1 Implications of the Microservice Applications Structure

Studying 103 open-sourced Docker Compose configuration files discovered the prevalence of microservice applications with *a power law distribution of degrees* in the application graph. This structural feature implies the presence of one or several highly-connected microservices. Such a microservice application design pattern might lead to highly vulnerable applications in case microservices with a high number of connections implement a critical functionality.

In some cases, the microservice with the highest number of connections is just a logging service, hence its failure won't influence SLOs. Thus, structural analysis and modeling of microservice applications *should be enhanced with the analysis of the functional context* such that critical microservices are clearly recognized and are not mixed with ones that are not critical but are still highly relied upon. Such information can be used to ensure that the availability, throughput and resource requirements are satisfied by helping decide the appropriate number of critical microservices' replicas.

Among several graph generation models studied, the BA-model demonstrated an ability to capture the degree distribution of the microservice application using relatively small intervals of values for its parameters *power of preferential attachment* and *attractiveness of vertices with no edges*. Changing these parameters means modifying the number of connections that few nodes have (first parameter) and changing the number of nodes central to some local clusters of nodes (second parameter). A high value of the parameter *attractiveness of vertices with no edges* allows us to model fairly complex graphs with several "centers of attraction".

The study of network distances between generated random graphs and 42 microservice applications graphs underlines that one model cannot convey all the properties of the microservice application structure. This can be solved via analytical models that generate random graphs exhibiting characteristics of several models: consider similar work by Solé et al. [30] or by learning a deep generative graph model on a representative set of examples [22].

Both simple and hybrid random graph models can be employed to synthesize structures that correspond to real microservice applications. Varying the parameters of such models would enable capturing the peculiarities of a microservice application's structure. As one can select the number of vertices and edges for such models arbitrarily, the absence of large open-sourced microservice applications does not hinder the design and evaluation of algorithms utilizing in some way the information on the applications' structures. However, with the simplifications that could be made when identifying the appropriate random graph model (e.g., omitting information on types of services), it may become necessary to validate the model manually by developing a sample large-scale microservice application with limited functionality.

The analysis of the microservice application's structures in the paper is based on the degree of graph vertices. This could be viewed as a limiting factor as the graph abstraction offers a rich set of parameters to study the microservice structure in-depth, e.g., vertex connectivity or isoperimetric number. For example, one could think of studying the vertex connectivity of the microservice applications' graphs to identify the cornerstone services whose removal, say due to failure, damages the functionality of the application. An isoperimetric number can be used in studies of potential bottleneck services. Consideration of these parameters was deemed beyond the scope of this paper.

The conducted structural analysis makes a strong assumption that the application is static, which in practice does not always hold true. Addition and removal of microservices over time is a

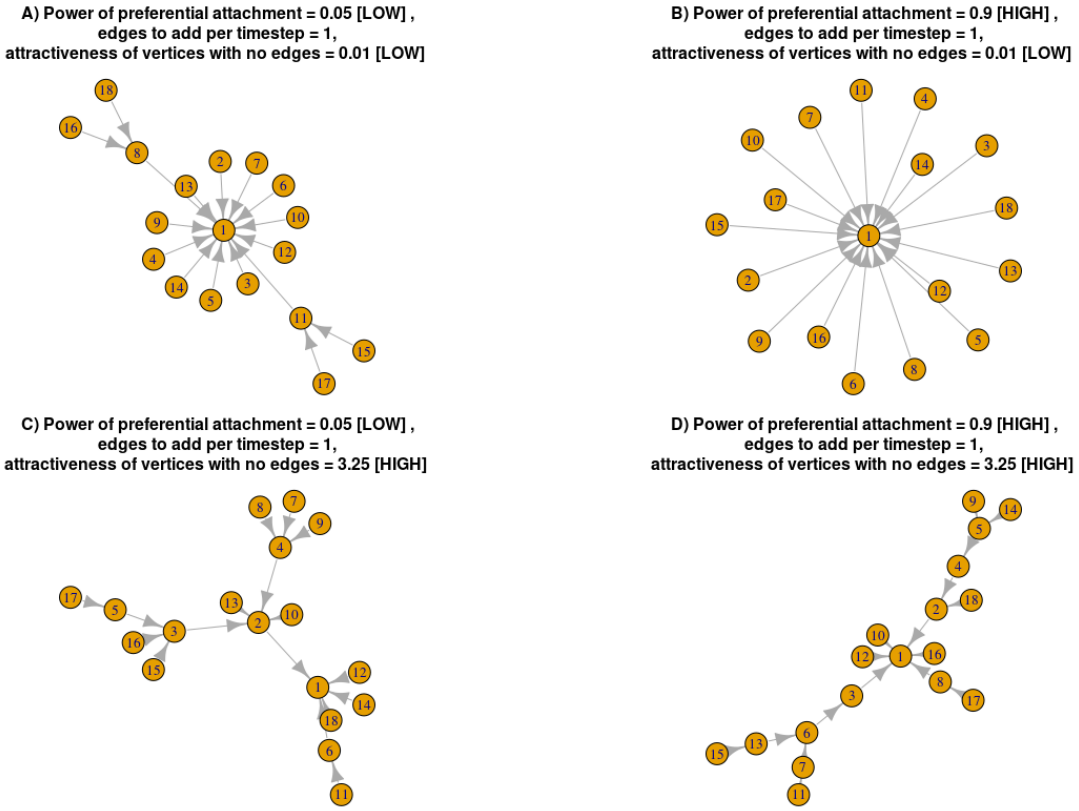


Figure 6: Random Graphs generated using BA with four parameter. Number of nodes: 18.

normal practice for such applications. Dynamic graph analysis of the microservice application will lead to models that capture the evolution of the application. In turn, such models could contribute to increasing the accuracy of predictive autoscaling by incorporating the knowledge of potential structural changes in the model.

6.2 Application Structure towards assuring Software Quality Attributes

Knowing an application's structure can contribute to quality assurance of the application across the software life cycle:

Scalability and Availability. Revealing the relationship between scaling events and applications' capacity will lead to the fine tuning of the scaling actions; instead of individual scaling actions one might speak of *scaling action cascades* directed by the structure of an application and capacities of microservices. We believe that the adaptation of the microservice applications to changing workloads can be improved by including the application structure into the set of autoscaling parameters. Such improvements for real elastic microservice applications hosted in the cloud can result in better quality of service and budget savings, therefore it seems necessary to consider the application structure when scaling.

Testability and Correctness. We identified and replicated the applications' architectures. To this end, realistic benchmarks can be created using these models as a generic template. The templates can be used in the testing process for the product or for cases that the product acts as an input for other applications. Additionally, computationally expensive quality assurance methodologies, such as formal verification, could be better targeted towards the various soft points in a topology of an application.

Security and Reliability. The identification of the *weakest link* service with the most services that depend on it can help to make precautions for protecting the applications in advance or making changes in the infrastructure to make it safer from attacks. More specifically, certain rollback policies can be applied based on the service dependencies in case they go offline.

Performance efficiency. The *weakest link* services can be deployed based on their connectivity. Certain resources can favour certain types of services to achieve better response times and thus better performance. The configuration of a service-container can be set to allow for more hardware resources on critical services than on less critical ones.

Adaptivity. From a self-adaptive systems perspective, being able to create and analyze models of one's composition is a crucial self-* property that can be used to analyze and plan adaptation such that various quality attributes (or setpoints/goals) are satisfied.

Finally, the application topology reveals the *strongest link*. By making certain design choices that will shift the load from the strongest to the weakest service can help towards the application quality, as well.

7 CONCLUSION AND FUTURE WORK

The study discovered degree distributions that are widely-present in graphs of 103 open-sourced microservice applications: power law, uniform, and normal. Looking closer at 42 applications that exhibited power law-like degree distribution allowed us to discover that BA-based random graphs capture the structure of real microservice applications well. By employing this model, one can synthesize random graphs with a large number of vertices that capture the structural properties of microservice applications.

The study paves the way towards larger and systematic empirical studies of how microservice applications tend to be structured, resulting in new heuristic algorithms for improved scaling, self-protection from targeted attacks, testing and system administration. Revealing and generating models based on their connectivity, while viewing an application as a directed graph of services, can be very helpful for application evolution.

The following future research directions appear to have significant utility in microservice applications deployment and management: customized analytic and machine learning-based graph models to generate random graphs; extension of the structural modeling and analysis with microservice types; extending graph models capturing properties of microservice applications with other graph characteristics and building dynamic graph models to predict structural changes. The main limiting factor for the research of microservice application structures is the novelty of the concept and limited public availability of real microservice applications. With the continuing adoption of the microservice architecture for cloud-native applications, more data would become available in public repositories and more mining-based research can be done. With more publicly available knowledge, we aim to explore further types of applications that use certain programming languages and frameworks to reveal even more aspects of the status quo of software products.

ACKNOWLEDGMENTS

This work was supported by AWS research program Cloud Credits, STRATUS, a project funded by New Zealand's Ministry of Business, Innovation and Employment (MBIE), the Natural Sciences and Engineering Research Council of Canada (NSERC) and Canada's New Brunswick Innovation Fund (NBIF). We also thank Stephen MacKay for his careful proofreading and editing the paper to improve its quality. We also thank anonymous reviewers for their comments which we tried to address in the final version of the paper.

REFERENCES

[1] 2020. Docker compose files to analyze structural patterns of containerized microservice applications. https://doi.org/10.5281/zenodo.3573846. [Online; accessed 8-June-2020].
[2] 2020. The Official YAML Web Site. https://yaml.org/. [Online; accessed 26-March-2020].
[3] Réka Albert and Albert-László Barabási. 2002. Statistical mechanics of complex networks. *Rev. Mod. Phys.* 74 (Jan 2002), 47–97. Issue 1.
[4] Albert-László Barabási and Réka Albert. 1999. Emergence of Scaling in Random Networks. *Science* 286, 5439 (1999), 509–512.
[5] Albert-László Barabási and Márton Pósfai. 2016. *Network science.* Cambridge University Press, Cambridge. http://barabasi.com/networksciencebook/
[6] R. Birke, J. F. Perez, Z. Qiu, M. Borkqvist, and L. Y. Chen. [n.d.]. sPARE: Partial Replication for Multi-tier Applications in the Cloud. *IEEE Transactions on Services Computing* ([n. d.]), 1.
[7] J. Bogner, J. Fritzsch, S. Wagner, and A. Zimmermann. 2019. Microservices in Industry: Insights into Technologies, Characteristics, and Software Quality. In *2019 IEEE International Conference on Software Architecture Companion (ICSA-C).* 187–195.
[8] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. 2018. NetGAN: Generating Graphs via Random Walks. In *ICML.*
[9] Phillip Bonacich. 1987. Power and Centrality: A Family of Measures. *Amer. J. Sociology* 92, 5 (1987), 1170–1182.
[10] Fan Chung and Linyuan Lu. 2002. Connected Components in Random Graphs with Given Expected Degree Sequences. *Annals of Combinatorics* 6, 2 (01 Nov 2002), 125–145.
[11] Rhys Compton, Eibe Frank, Panos Patros, and Abigail Koay. 2020. Embedding Java classes with code2vec: improvements from variable obfuscation. In *IEEE/ACM 17th International Conference on Mining Software Repositories (MSR 2020).* ACM.
[12] Gabor Csardi and Tamas Nepusz. 2006. The igraph software package for complex network research. *InterJournal* Complex Systems (2006), 1695. http://igraph.org
[13] Paul Erdős and Alfréd Rényi. 1959. On Random Graphs I. *Publicationes Mathematicae (Debrecen)* 6 (1959 1959), 290–297.
[14] M. Fazio, A. Celesti, R. Ranjan, C. Liu, L. Chen, and M. Villari. 2016. Open Issues in Scheduling Microservices in the Cloud. *IEEE Cloud Computing* 3, 5 (Sep. 2016), 81–88.
[15] K.-I. Goh, B. Kahng, and D. Kim. 2001. Universal Behavior of Load Distribution in Scale-Free Networks. *Phys. Rev. Lett.* 87 (Dec 2001), 278701. Issue 27.
[16] W. Hasselbring and G. Steinacker. 2017. Microservice Architectures for Scalability, Agility and Reliability in E-Commerce. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW).* 243–246.
[17] Adery C. A. Hope. 1968. A Simplified Monte Carlo Significance Test Procedure. *Journal of the Royal Statistical Society. Series B (Methodological)* 30, 3 (1968), 582–598. http://www.jstor.org/stable/2984263
[18] Docker Inc. 2020. Compose file version 3 reference. https://docs.docker.com/compose/compose-file/. [Online; accessed 26-June-2020].
[19] Steffen Kächele and Franz J. Hauck. 2013. Component-based Scalability for Cloud Applications. In *Proceedings of the 3rd International Workshop on Cloud Data and Platforms (CloudDP '13).* ACM, New York, NY, USA, 19–24.
[20] Robert V. Krejcie and Daryle W. Morgan. 1970. Determining Sample Size for Research Activities. *Educational and Psychological Measurement* 30, 3 (1970), 607–610.
[21] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD '05).* ACM, New York, NY, USA, 177–187.
[22] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. 2018. Learning Deep Generative Models of Graphs. arXiv:cs.LG/1803.03324
[23] X. Liu, J. Heo, and L. Sha. 2005. Modeling 3-tiered Web applications. In *13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems.* 307–310.
[24] S. Malkowski, M. Hedwig, and C. Pu. 2009. Experimental evaluation of N-tier systems: Observation and analysis of multi-bottlenecks. In *2009 IEEE International Symposium on Workload Characterization (IISWC).* 118–127.
[25] G. Márquez, M. M. Villegas, and H. Astudillo. 2018. An Empirical Study of Scalability Frameworks in Open Source Microservices-based Systems. In *2018 37th International Conference of the Chilean Computer Science Society (SCCC).* 1–8.
[26] V. Podolskiy, M. Mayo, A. Koay, M. Gerndt, and P. Patros. 2019. Maintaining SLOs of Cloud-Native Applications Via Self-Adaptive Resource Sharing. In *2019 IEEE 13th International Conference on Self-Adaptive and Self-Organizing Systems (SASO).* 72–81.
[27] Nigel Poulton and Pushkar Joglekar. 2019. *The Kubernetes Book* (fourth ed.).
[28] J. F. Pérez, L. Y. Chen, M. Villari, and R. Ranjan. 2018. Holistic Workload Scaling: A New Approach to Compute Acceleration in the Cloud. *IEEE Cloud Computing* 5, 1 (Jan 2018), 20–30.
[29] S. S. Shapiro and M. B. Wilk. 1965. An analysis of variance test for normality (complete samples)†. *Biometrika* 52, 3-4 (1965), 591–611.
[30] Ricard V Solé, Romualdo Pastor-Satorras, Eric Smith, and Thomas B Kepler. 2002. A model of large-scale proteome evolution. *Advances in Complex Systems* 05, 01 (2002), 43–54.
[31] M. A. Stephens. 1974. EDF Statistics for Goodness of Fit and Some Comparisons. *J. Amer. Statist. Assoc.* 69, 347 (1974), 730–737.
[32] Q. Wang, Y. Kanemasa, J. Li, D. Jayasinghe, T. Shimizu, M. Matsubara, M. Kawaba, and C. Pu. 2013. Detecting Transient Bottlenecks in n-Tier Applications through Fine-Grained Analysis. In *2013 IEEE 33rd International Conference on Distributed Computing Systems.* 31–40.
[33] Qingyang Wang, Y. Kanemasa, J. Li, D. Jayasinghe, T. Shimizu, M. Matsubara, M. Kawaba, and C. Pu. 2013. An Experimental Study of Rapidly Alternating Bottlenecks in n-Tier Applications. In *2013 IEEE 6th International Conference on Cloud Computing (CLOUD),* Vol. 00. 171–178.

Report on Evaluation Experiments Using Different Machine Learning Techniques for Defect Prediction

Marios-Stavros Grigoriou
Dept. of Computer Science
Western University
London, ON, Canada
mgrigori@uwo.ca

Kostas Kontogiannis
Dept. of Computer Science
Western University
London, ON, Canada
kostas@csd.uwo.ca

Alberto Giammaria
Austin Laboratory
IBM
Austin, TX, USA
agiammaria@us.ibm.com

Chris Brealey
Toronto Laboratory
IBM
Toronto, ON, Canada
cbrealey@ca.ibm.com

ABSTRACT

With the emergence of AI, it is of no surprise that the application of Machine Learning techniques has attracted the attention of numerous software maintenance groups around the world. For defect proneness classification in particular, the use of Machine Learning classifiers has been touted as a promising approach. As a consequence, a large volume of research works has been published in the related research literature, utilizing either proprietary data sets or the PROMISE data repository which, for the purposes of this study, focuses only on the use of source code metrics as defect prediction training features. It has been argued though by several researchers, that process metrics may provide a better option as training features than source code metrics. For this paper, we have conducted a detailed extraction of GitHub process metrics from 148 open source systems, and we report on the findings of experiments conducted by using different Machine Learning classification algorithms for defect proneness classification. The main purpose of the paper is not to propose yet another Machine Learning technique for defect proneness classification, but to present to the community a very large data set using process metrics as opposed to source code metrics, and draw some initial interesting conclusions from this statistically significant data set.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**; *Maintaining software*; *Software version control*.

ACM Reference Format:
Marios-Stavros Grigoriou, Kostas Kontogiannis, Alberto Giammaria, and Chris Brealey. 2020. Report on Evaluation Experiments Using Different Machine Learning Techniques for Defect Prediction. In *Proceedings of CASCON'20:*. ACM, New York, NY, USA, 10 pages.

CASCON'20:, Nov 10-13, 2020, Toronto, Canada

1 INTRODUCTION

The problem of classifying a file as failure prone or not has attracted the attention of the software engineering community early on. Early approaches focused on the use of software metrics to compute maintainability and software health indexes [2] [18]. These approaches were based on the compilation of linear or non-linear formulas to yield maintainability indexes which were assumed to be associated with the overall health of a component or a system. In this respect, the assumption was that a higher maintainability index would indicate a software component (function, method, file, or module) that has a low probability of exhibiting a failure. As research progressed in this field, the software engineering community experimented with approaches focusing on the static and dynamic analysis as well as the analysis of project data, such as the number, type and time interval between bug fixes [7] [16]. These approaches utilized statistical analyses and heuristics to experimentally yield predictions related to the fault-proneness of a software component. However, over the past few years, research in this area has decisively shifted towards the use of Machine Learning *(ML)* techniques. These techniques aim to first identify a collection of source code related and process related features which can serve as classifiers for fault-proneness, and second apply these features for training *ML* models using a variety of *ML* algorithms (see Section 2). Once such models are trained the premise is that they can be used to classify whether a newly seen software component is defect prone or not.

The challenge that arises using such *ML* techniques is that they yield models which perform as black boxes and do not provide any explanation on how their results have been reached, as they are purely dependent on the training data set provided, and the *ML* algorithm used. Another challenge that arises is when *ML* models are trained on source code metrics alone. Large software systems are rarely implemented using a single programming language and are often composed of a collection of different frameworks, configuration scripts and dynamically linked components. That makes the extraction of accurate source code metrics an almost impossible task. On the contrary, process related metrics can be extracted quite accurately and easily from various DevOps tools such as GitHub, Jira, Jenkins, and Slack.

In this context, this paper aims to shed light on two major issues.The first issue is to identify, through the use of process metrics and extensive experimentation, the technique, or the combination of techniques and features, that best classify whether a software component (i.e. file) can be considered defect prone or not. The second issue is whether process metrics can be used instead of source code metrics and whether these can be used to train models that yield similar of better classification results in a single project or across projects. These two issues can be formalized by four related research questions as follows:

RQ1: *By using a very large set of open source projects to experiment with, which is the best combination of classifiers which are fast, easily trainable and able to yield the best results as these are measured in terms of accuracy, precision, recall, F1, and AUC*?

RQ2: *What is an optimal subset of available process metrics which can be easily calculated and at the same time yield the best results when provided as input to different classifiers*?

RQ3: *Is it possible to perform defect-proneness classification using process metrics while maintaining classification performance measures comparable to similar techniques reported in the literature which use source code metrics*?

RQ4: *Is it possible to perform cross project defect proneness classification in the sense that data from different projects can be used to train a model which will then be used to perform defect proneness prediction on other unknown projects for which not enough training data may be available*?

For this paper we take an experimental approach, aiming to draw conclusions by applying the techniques under examination to a very large collection of open source projects. More specifically, we have considered a collection of 148 open source systems from which we have extracted various process metrics utilising a custom-made extraction tool. The open source projects were selected based on their complexity, size, prevalence, and the quality and availability of process repository data. The importance of the work reported in this paper lies on two parts. First, in the best of our knowledge, it is the first work which utilises such a large data set of 148 open source systems, providing thus a much more statistically significant result than previously reported works, and second providing answers to research questions which can assist researchers advance the state-of-the-art in the area.

The paper is organised as follows. Section 2 presents related work. Section 3 discusses the features and the feature extraction process. Section 4 presents the different Machine Learning techniques which we have evaluated. Section 5 presents the results obtained, while Section 6 discusses and interprets the obtained results. Finally, Section 7 concludes the paper and offers pointers for future work.

2 BACKGROUND

In the related literature there is a wealth of approaches for defect prediction using Machine Learning techniques. Two widely-used defect prediction techniques are regression and classification. The main purpose of regression techniques is to estimate the number of software defects on a software component. In contrast, classification techniques aim to tag whether a software module is faulty or not. It has been shown that classification models can be trained from defect data on earlier versions of the system being analyzed. Some of the most commonly used supervised learning techniques for defect prediction are outlined below.

Decision Trees (DT): Decision tree algorithms use tree structures to model decisions and their possible consequences. In decision trees, each leaf node corresponds to a class label while attributes are represented as internal .

Logistic Regression (LR): Logistic regression is a supervised classification algorithm whereby the target variable O (i.e output), can take on values in the interval $[0, 1]$ representing the probability for a given set of input features I to belong to class 1 or 0.

Random Forest (RF): RF is an ensemble type of learning method used for both classification and regression problems. The key idea behind RF is the construction of several decision trees at training time and outputting the mode/mean prediction of the individual trees.

Support Vector Machine (SVM): SVM is a discriminative classifier formally defined by a separating hyperplane. In SVMs, given a labeled training data set whereby each data item is marked as belonging to one or the other of two categories, the algorithm outputs an optimal hyperplane, which classifies new unseen data in one of these two categories.

k-Nearest Neighbors (k-NN): k-NN is a non-parametric method that can be used for both classification and regression problems. In both cases, the input consists of the k closest training examples in a feature space. The output depends on whether k-NN is used for classification or regression. In classification, the output is to categorize an input to one of equivalence classes. In regression, the output is to assign a value to the input, usually the average of the values of its closest k-neighbors.

Naive Bayes Classifiers (NB): These classifiers refer to a family of simple "probabilistic classifiers" based on applying Bayes' theorem and by considering a strong independence assumption between features, that is the presence or absence of a particular feature of a class is not related to the presence or absence of any other feature.

Neural Networks (NN): Neural Networks are nonlinear predictive structures that consist of interconnected processing elements called neurons that work together in parallel within a network to produce output, often simulating an unknown function or phenomenon.

Multi-layer Perceptron (MLP): MLPs refer to a class of feedforward artificial neural network (ANN). An MLP comprised of a directed graph of multiple layers of nodes which are fully connected to the nodes of the next layer. For training purposes, MLP utilizes a supervised learning technique defined as *backpropagation*.

Radial Basis Function (RBF) Networks: RBF Networks are a type of ANNs used to approximate through training the value of an unknown function. They are different from MLPs in the sense that they are feedforward networks comprising of only three layers, the input layer, the hidden layer and the output layer.

2.1 Defect Prediction using Machine Learning

A variety of machine learning methods have been proposed and assessed for addressing the software bug prediction problem. These methods include decision trees [24], neural networks [32, 39], Naive Bayes [12, 15, 34], support vector machines [5], Bayesian networks [27] and Random Forests [1].

2.1.1 Source CodeMetrics Approaches. Deciding whether a component has a high likelihood to be defective or not has been proved to have a strong correlation with a number of software metrics. Identifying and measuring software metrics is vital for various reasons, including estimating program execution, measuring the effectiveness of software processes, estimating required efforts for processes, estimating the number of defects during software development as well as monitoring and controlling software project processes [29] [6]. Various software metrics have been commonly used for defect prediction, including lines of code (LOC) metrics, McCabe metrics, Halstead metrics, and object-oriented software metrics. Hence, the automated prediction of defective components from extracted software metrics evolved as a very active research area. [14]. In [26], Nagappan aims to find the best code metric to predict bugs. The conclusion of this work is that complexity metrics can successfully predict post-release defects, but there is no single set of metrics that is applicable to all systems. Hassan et. al have investigated the impact of different aspects of the modelling process to the end results and the interpretation of the models [4] [3] [11] [8].

2.1.2 Process Metrics Approaches. In [19], Venkata et. al compared different machine learning models for identifying faulty software modules and they found that there is no particular learning technique that performs the best for all the data sets. In [36], Wang and Yao aim to find bugs without decreasing the overall performance of the model. In this process, they find that imbalanced distribution between classes in bug prediction is the root cause of its learning difficulty. Likewise, in our paper, we noted the issue and used re-sampling as described in detail in the section 4.3 in an effort to minimise the impact of class imbalance to the quality of our results. Similarly, in [17], Zimmermann et. al propose an approach to predict bugs on cross-language systems. The work examined a large number of such systems and concluded that only 3.4% of the systems had precision and recall prediction levels above 75% . The authors also tested the influence of several factors on the success of cross-language prediction and concluded that there was no single factor which led to such successful predictions. The authors used decision trees to train the model and to estimate precision, recall, and accuracy before attempting a prediction across systems. Lastly, in [23], Hassan discusses how frequent source code "commits" in the repository negatively affect the quality of the software system, meaning that the more changes incurred to a file, the higher the chance that the file will contain critical errors. Furthermore, the author in [23] presents a model which can be used to quantify the overall system complexity using historical code-change data, instead of plain source code features.

3 DATA MODELING

For the purposes of this study we have designed two separate data models. The first data model denotes the raw information which can be mined from software repositories, while the second data model denotes the post-processed raw data which are in a form that can be consumed by the machine learning algorithms we have experimented with.

The design goal of the first data model was to have a structure which would be easy to populate while maintaining a low memory profile, would facilitate data reconciliation of data entries originating from different devOps tools (e.g. GitHub, Jenkins, Jira), would be scalable, and would be able to support preprocessing workflows of varying complexity at high speeds. The schema for this data model is depicted in Fig. 1.

The design of the second data model was to have a simple relational structure which can be easily imported as a tab or comma delimited file in various machine learning tools and which can be easily manipulated so that aggregate features can be easily computed. The features in this second data model are depicted in Table 1.

3.1 Raw Data Model

For this study we have exhaustively collected process related metrics from 148 open source systems of various sizes and complexities. The list of the systems along with all the data obtained or computed are listed in the anonymous repository [35].[1] The profile of the data set we have considered is depicted in Table 2. The data acquisition process is based on two steps. The first step is to utilize a custom made client-side extractor tool which is able to connect to and reconcile data obtained using various tools and namely GitHub, Bugzilla, Jira, and Jenkins. However, for this study we report results on data acquired only from the GitHub repositories of these 148 open source systems. The second step of the raw data acquisition process is to fuse the information extracted by each repository record into one repository which conforms to the raw data schema depicted in Fig. 1. The extractor application and its data fusion module is implemented using Python 3.

As depicted in Fig. 1, the raw data model is founded on the concept of a *Commit*, the concept of a *File*, and the concept of a *CommitProperties*. The extracted information is represented as a Json file stored in a Mongo DB server. As such, a run-time model of a GitHub repository was created which held the information of the unique commit records. Every commit contained a list of *fileChanges* and the details for each of these files' change. This data model represents a GitHub record structure utilizing simple Python 3 objects which have a very low memory profile and initialization time.

In this data model a commit is uniquely identified by it's *commitID*, it contains attributes specific to it, including the *author*, the *commit-time*, *the files committed* as these are denoted by their *FileIds*, a *commit message*, the *overall lines added, deleted* as well as a tag field maintaining information about whether the *Commit*'s status is indicated as a *bug fixing* or as a *clean* Commit. For each *File* within

[1] Please note that the repository is anonymous for the time being, in order to protect the double blind review process and to facilitate the assessment of this work by the reviewers. Please do not distribute or use without prior arrangements with the authors.

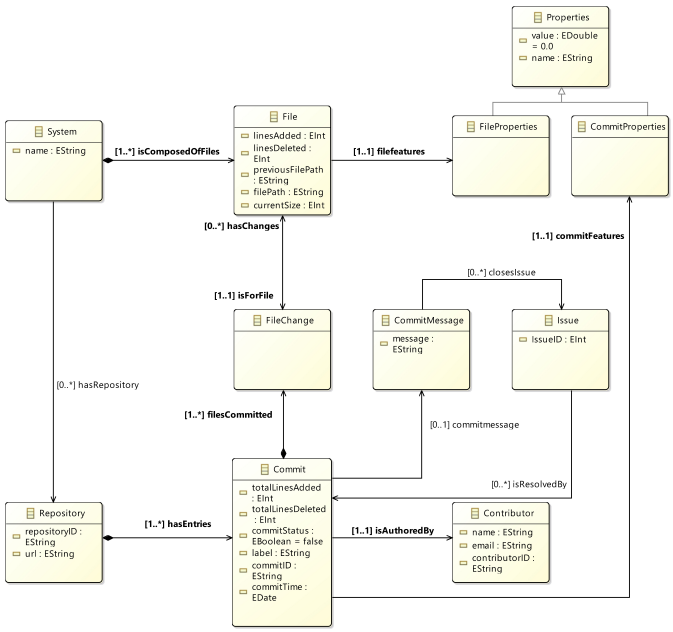


Figure 1: Data Model for Raw Repository Data

a commit, the added, and deleted lines as well as the current size of the file are maintained. Storing the *fileID* is necessary since in case of a file changing locations the *fileID* remains the same even if the file changes name (the names are fully qualified names with respect to the root folder of a project). In addition to the version control model, another important component of this extractor system is the issue tracker component. This component is far simpler. It is a simple object maintaining a specific *issueID* together with the issue tag, its message, and its referenced *commitID*s. The data model is populated by initially downloading a complete repository from the corresponding GitHub site and then moving through each commit on the master branch adding the relative data iteratively in it, thus maintaining the initial structure. Once the model is populated it undergoes several steps of preprocessing. The first task is to remove all files that cannot contribute to a defect, such as any non-compilable and non-configuration related files. The next task is to use a simple heuristic to clean up the extracted commits so that only actual code changing commits remain. This entails removing entries that are clearly annotated as *refactoring* commits, because of words stemming from *refactoring, reorganize, restructure* in the commit message, and also removing all files which have been eventually removed from the system from all past commits. Finally all merge commits are also removed since they contain change information pertaining to different branches and will therefore introduce large amount of noisy data points to the dataset. Given that this study is not focusing on defect introducing software changes, the removal of refactoring and merge commits from the dataset will not impact its ability to discern between faulty and healthy files. After the cleanup stage is completed, the most important remaining task is that of assigning the class label for each commit. This task is accomplished by parsing the commit message for terms that

Table 1: Features Used for System Training

F1: NoOfCommits (CF)	F2: LateNightCommits (LNC)
F3: TotalAddedLines (TAL)	F4: MaxAddedLines (MAL)
F5: AvgAddedLines (AAL)	F6: TotalDeletedLines (TDL)
F7: MaxDeletedLines (MDL)	F8: AvgDeletedLines (ADL)
F9: TotalChurn (TCF)	F10: MaxChurn (MC)
F11: AvgChurn (AC)	F12: TotalCoCommitSize (TCS)
F13: MaxCoCommitSize (MCS)	F14: AvgCoCommitSize (ACS)
F15: TotalDistinctAuthors (TDA)	F16: AgeInMonths (AIM)
F17: FractalValue (FRV)	F18: FailureIntensity (FI)

may indicate that it is a bug-fixing commit as opposed to a clean one, linking commits to issue tracker entries labeled as faults and optionally, applying the same parsing as above to the issue tracker messages. The heuristic terms used to tag a commit as a bug-fixing one are presented in Section 4.2 below. This is an approach for automatically generating datasets for such applications which has been known to work in the related literature [21]. This approach however has the potential to produce a high number of false positives in the dataset because of the commit granularity level at which it is applied. This is due to the fact that some commits may be only partially defective leading to wrong labels for the non-defective part [9]. In this case the entirety of the commit and its modifications are annotated as bug-fixes, which is not acceptable for commits that contain a significant percentage of a systems' files.

3.2 Post-Processed Data Model

Once the raw data are extracted they are post-processed in order to yield a data model suitable for input to various Machine Learning classification tools. Table 1 provides a list of the features considered for our study which represent a subset of all features available in the literature, pertaining only to file properties and ommiting any features stemming from organisational or social structures.

3.3 Explanation of the Features

The *FractalValue* [33] provides a measure for the contribution of different authors to a file. It can take any value in the range (0, 1] where a value of 1 means that a file has had a single author whereas a value close to 0 means that the file has had similar contributions from multiple different authors. *TotalCoCommitSize* for a file F_i in a system S is the count of all files $F_j \in S$ which have been committed alongside F_i, counting multiple occurrences of the same files.

4 MACHINE LEARNING FRAMEWORK

In this section we discuss the machine learning algorithms used and technical details on how training and testing were conducted for this study.

4.1 Machine Learning Models Considered

For our experiments we have considered six different classification algorithms, and namely *(1)* Logistic Regression; *(2)* Support Vector Machines; *(3)* Multi-Layer Perceptron; *(4)* Decision Trees; *(5)* Random Forests and; *(6)* Naïve Bayes. The selection of these algorithms

is based on the fact that these are the algorithms most commonly used in the related literature [31], [13], [14] [10].

Each of the aforementioned algorithms comes with its own benefits and drawbacks. The most important benefit was the speed at which these can be trained and evaluated while the most important drawback of all approaches except for Logistic Regression was the lack of explainability. Nevertheless, the combination of these algorithms is currently the de-facto standard in the related literature as base classifiers [14] [31].

4.2 Commit Tagging

As discussed in Section 3.2 the raw data repository is considered as a container of commits. Each commit is tagged in the repository as a *bug fixing commit* or a *clean commit*. This tagging is based *a)* on the label of the GitHub commit record itself, or in the absence of such a label by analyzing the *comments* section of the commit record. More specifically, if the *comments* section of the commit record contains any of the keywords *'bug', 'bugs', 'defect', 'defects', 'error', 'errors' 'fail', 'fails', 'failed', 'failing', 'failure', 'failures', 'fault', 'faults', 'fix', 'fixes', 'fixed', 'fixing', 'problem', 'problems', 'wrong'* which *may indicate* a bug fixing *intention*, then the commit is tagged as a *buggy commit* (label 1), otherwise as a *clean commit* (label 0).

4.3 File Tagging

In its turn, a commit is considered itself as a container of files. If a commit is tagged as a bug fixing one (see above), then all the files in the commit are also tagged as buggy. This is a heuristic that can introduce many false positives, but unfortunately in the absence of a gold standard this is the best approximation and it is also a heuristic which is used in most papers appearing in the related literature [21]. In our data model each file also contains details about the contribution of that file to the commit in terms of the lines of code added or deleted as a percentage of the overall number of lines of code added or deleted in the commit.

In the related research literature there is no authoritative set of tagged files which can serve as a gold standard. The only such data set is PROMISE which relates only to source code metrics and does not include process metrics. We have identified an intersection of 11 projects available in PROMISE [30] which have a tagging (buggy or clean) and for which we can extract process metrics. We have used these 11 projects to answer research question Q_3.

As most of the 148 systems which we have considered for this study are open source systems the operational life of which spans several years, we have split the commits into two eras. The rationale behind this split is that very old commits (e.g. commits which may be several years old) should not bear significant weight to the overall computation. The first era consists of the past 70% of the commits and the second era of most recent 30% of the commits. The experimentation set-up proceeds then as follows.

Feature Entries

Let $F_{i,j} =< m_1, m_2, \ldots, m_{18} >^{i,j}$ denote a feature value vector entry for file F_i participating in commit C_j, where m_k is the value of a feature f_k $k \in \{1, 2, ..18\}$ (see Table 1) related to file F_i in commit C_j.

Let also $F_{i,S} = < v_1, v_2, ..., v_{18} >^{i,S}$ be the feature vector entry for file F_i across all commits C_j, $C_j \in S$, in which F_i appears, and where each value v_p, $p \in \{1, 2, ..18\}$, is obtained by combining all correcponding m_p's appearing in feature value vector entries $F_{i,j}$.

Let us also assume that the commits C_1 C_2, ... $C_{j-1} = S_1$ belong to the first era of 70% of commits and C_j C_{j+1}, ... $C_n = S_2$ belong to the era of the most recent 30% of system commits. Then the resulting feature value vector, for all commits $S = S_1 \cup S_2$ containing changes for this file F_i, will be $F_{i,S} = F_{i,S_1 \cup S_2}$ will be $<< v_1, v_2, ..., v_{18} >^{i,S_1}$, $< v_1, v_2, ..., v_{18} >^{i,S_2} >$.

Tagging Process

Let S_R be the set of commits in the most recent 30%, and S_P be the set of commits in the past 70%, of the total commits of the system. Let also F_i be a file modified in commit C_k. If $C_k \in S_R$ has been identified as a *bug fixing commit* then the commit's metrics vector becomes $F_{i,k} =< m_1, m_2, \ldots, m_{18}, \mathbf{1} >^{i,k}$ and the vector across all commits of the file becomes $<< v_1, v_2, ..., v_{18} >^{i,S_P}, < v'_1, v'_2, ..., v'_{18} >^{i,S_R}, \mathbf{1} >>$.

Accordingly for file F_i, C_k and S_R, S_P as above, if all $C_k \in S_R$ have been tagged as a *clean commit* then the overall feature vector of F_i becomes $<< v_1, v_2, ..., v_{18} >^{i,S_P}, < v'_1, v'_2, ..., v'_{18} >^{i,S_R}, \mathbf{0} >>$. Finally for a file F_i, C_k and S_R, S_P as above, if there is no $C_k \in S_R$ then the overall feature vector of F_i becomes $< v_1, v_2, ..., v_{18} >^{i,S_P}$, $< NULL, NULL, ...NULL >^{i,S_R}, \mathbf{0} >>$.

Example

As an example, consider the file F_{10} which appears in commits C_5, C_{50}, and C_{1000} where C_5, C_{50} belong to the first era (past 70%) while C_{1000} belongs to the recent 30% and is tagged as a bug fixing commit. Then the file F_{10} will have the following feature vector:

$<< v_1, v_2, \ldots, v_{18} >^{10,\{5,50\}},$
$< v_1, v_2, ..., v_{18}, 1 >^{10,\{1000\}} >>$

and which is produced by aggregating the feature value vectors:

$<< m_1, m_2, ..., m_{18}, 0 >^{10,5}$
$< m'_1, m'_2, \ldots, m'_{18}, 0 >^{10,50}$
$< m''_1, m''_2, \ldots, m''_{18}, 1 >^{10,1000}$

The post processed data model is essentially a relational table where each line in the table is the feature vector entry of the file $< v_1, v_2, \ldots, v_{18}, Tag >^{i,S}$.

The obtained results are then the data considered on this study in order to answer questions $Q_1 - Q_4$.

Rebalancing

In the related research literature rebalancing is applied in order to avoid overfitting classifiers or other *ML* models to the majority class. Likewise we used the *random oversampling* technique [25] to tackle this threat, accepting that it may contribute to drift bias in the generated models [3]. The technique involves randomly selecting instances from the minority class with replacement before the training stage, to create a new set of the minority class' instances which will resemble in cardinality that of the majority class.

Table 2: Project Statistics

Project Size (in files)	Avg. LOC	No. of Projects	Avg. Age (in years)	Avg. No. of Commits
148,740 – 20,000	15 MLOC	4	13	77,230
19,999 – 10,000	2.3 MLOC	4	11	18,870
9,999 – 5,000	1.3 MLOC	5	14	13,400
4,999 – 2,000	500 KLOC	18	10	7,288
1,999 – 1,000	457 KLOC	12	15	13,906
999 – 500	231 KLOC	26	10	3,900
499 – 200	86 KLOC	32	9	2,213
199 – 12	50 KLOC	47	9	1,325

4.4 Training and Test Set Data and Bias

Once we have calculated the feature vector entries for each file, we follow the 80-20 split rule for testing and training, and we consider all the entities in the post processed data (i.e. entries from the fisrt and second era of the commits). This was done so that the produced results will be comparable to the studies published in the relevant research literature [38]. Aiming at reducing the effect of outliers across all 148 systems considered, we trained and applied a scaler on the training data to decrease the effective range of all feature values, and also applied rebalancing on the minority and majority classes as explained in Section 4.3.

4.5 Evaluation and Performance metrics

The norm for extracting meaningful results from *ML* models is the use of the stratified k-fold cross validation technique [38]. The bootstrap and leave-one-out validation techniques were also used to provide a better understanding of how the models would perform during the application of the trained model. The performance metrics used in this study are the ones most frequently mentioned in the literature [3] [20]. In total, 7 different performance measures were calculated for each one of the variations of the technique and namely *precision*, *recall*, *accuracy*, *F1-score*, *Brier-score*, *Receiver-operator-characteristic/area-under-curve*, and where possible, *support*. In the context of academic research it has been argued that an overall high *F1-score* as well as a high *area-under-curve* are good indicators to identify whether a classifier is a successful one or not. Given, however, that F1 is a combination of Precision and Recall, it means that a satisfactory value for it is not necessarily the result of an optimal combination of it's constituent values. However, in industry, it is often preferred to maintain high precision even at the expense of recall. The rationale is that investigating false positives may require significant effort, or may result to the prediction system not being easily adopted by developers.

5 EVALUATION STUDIES

In this section we present the details of the studies we have conducted for answering the research questions Q_1 - Q_4. As stated, some basic statistics of the 148 open source projects, are depicted in Table 2. Due to space limitations only the highlights of the results will be presented in the *Obtained Results* subsection for each *RQ*. The full list of results for all *RQs* can be found on the accompanying repository [35].

5.1 Study 1: Identification of Best Classifiers

The first study aims to identify through experimentation the combination of the best classifiers to be used for defect prediction (classification). Our study here is by far not the first study of its kind. In [14] the authors have reviewed various research approaches and concluded that the best results are consistently given by the application of simple modelling techniques such as Logistic Regression and Naive Bayes. Similarly, the authors in [28] reported that the best features to use are *owner experience, overall developer experience, owner contributed lines, minor contributor count, and distinct dev count* of which only the last one is used in this approach. However, to our knowledge the study in this paper is the first of its kind in that it uses *a)* a very large, comprehensive, and statistically significant sample of 148 systems, in a quest to obtain conclusive results in the topic, and *b)* repository process metrics as opposed to source code metrics which as mentioned above do not require the use of specialised parsers and source code metrics calculators.

5.1.1 Study Set-up. For this study we have obtained feature vector entries collected from post-processing raw data extracted from 148 GitHub [22] repositories, as discussed in Section 3.1 and Section 3.2. The data were split into k-folds to apply k-fold cross validation where k=5. For each of the folds the training data were rebalanced to overcome class imbalance issues, using minority class oversampling. A scaler was trained using the training data, normalizing values to the $[0, 1]$ interval, and applied on both train and test datasets.

5.1.2 Obtained Results. For this study we used all combinations of the following classifiers: Decision Trees (DT), Random Forests (RF), Linear Regression (LR), Multi-Layer Perceptron (MLP), Support Vector Machines (SVM), and Gaussian Naive Bayes (GNB) to train a hard-voting classifier. These yielded a total of 9,324 individual results for 148 projects. Those are summarized in Table 3 where the range depicts obtained *min - max* scores.

The results indicate that if the Accuracy criterion is to be considered first amongst the top ranked combinations with respect to AUC, and F1 score, then the best combination of classifiers across all projects are the DT_LR_RF, followed by the DT_GNB_LR_MLP_RF. Furthermore, this observation becomes more pronounced for projects for which the Buggy/Clean file ratio is more that 0.5 (see Section 6). The best combination of classifiers as grouped by the ratio Buggy/Clean files is depicted in Table 4.

5.2 Study 2: Identification of Best Features

This part of the study aims to identify the optimal combination of the selected features that can be used for creating defect proneness classification models, similarly to the work in [28], but here we are investigating a different set of process metrics.

5.2.1 Study Set-up. For this study the classifier used was the DT_LR _RF which was identified as optimal as shown in Table 3, and was applied on all possible combinations of the features. For the evaluation of the trained models the k-fold cross validation with k=5

Table 3: Results of Classifier Combinations

Classifier Combination	Accuracy Range (Min - Max) Median Accuracy Avg. Accuracy	AUC Range (Min - Max) Median AUC Avg. AUC	F1 Range (Min - Max) Median F1 Avg. F1
DT_LR_RF	0.7 - 1.0 0.89 0.89	0.61 - 1.0 0.85 0.85	0.0 - 1.0 0.81 0.75
DT_GNB_LR_MLP_RF	0.69 - 1.0 0.88 0.88	0.64 - 1.0 0.85 0.86	0.0 - 1.0 0.79 0.75
DT_GNB_MLP_RF_SVM	0.61 - 1.0 0.87 0.88	0.64 - 1.0 0.85 0.85	0.0 - 1.0 0.79 0.75
DT_GNB_LR_MLP_RF_SVM	0.61 - 1.0 0.87 0.87	0.61 - 1.0 0.84 0.85	0.0 - 1.0 0.77 0.74
DT_GNB_LR_RF_SVM	0.65 - 1.0 0.87 0.87	0.62 - 1.0 0.84 0.85	0.0 - 1.0 0.77 0.74
LR_MLP_RF	0.61 - 1.0 0.87 0.87	0.63 - 1.0 0.85 0.86	0.0 - 1.0 0.79 0.75
DT_LR_MLP_RF_SVM	0.65 - 1.0 0.87 0.87	0.65 - 1.0 0.86 0.86	0.0 - 1.0 0.79 0.75
DT_LR_MLP	0.57 - 1.0 0.87 0.87	0.62 - 1.0 0.86 0.86	0.0 - 1.0 0.79 0.75
LR_MLP_RF_SVM	0.53 - 1.0 0.86 0.87	0.6 - 1.0 0.85 0.86	0.0 - 1.0 0.78 0.73
MLP_RF_SVM	0.65 - 1.0 0.86 0.87	0.65 - 1.0 0.85 0.86	0.1 - 1.0 0.79 0.74
DT_LR_MLP_SVM	0.57 - 1.0 0.86 0.87	0.62 - 1.0 0.85 0.85	0.0 - 1.0 0.77 0.73
GNB_LR_MLP_RF_SVM	0.65 - 1.0 0.86 0.87	0.64 - 1.0 0.85 0.86	0.0 - 1.0 0.78 0.74
DT_GNB_LR_MLP_SVM	0.65 - 1.0 0.86 0.87	0.6 - 1.0 0.85 0.86	0.0 - 1.0 0.78 0.74
DT_MLP_SVM	0.61 - 1.0 0.86 0.87	0.65 - 1.0 0.86 0.86	0.0 - 1.0 0.79 0.74
LR_RF_SVM	0.65 - 1.0 0.86 0.86	0.65 - 1.0 0.84 0.86	0.0 - 1.0 0.77 0.74
MLP	0.61 - 1.0 0.85 0.86	0.64 - 1.0 0.85 0.86	0.1 - 1.0 0.78 0.74
DT_LR_SVM	0.53 - 1.0 0.86 0.86	0.6 - 1.0 0.85 0.86	0.0 - 1.0 0.77 0.73
GNB_LR_MLP	0.61 - 1.0 0.86 0.86	0.59 - 1.0 0.84 0.85	0.04 - 1.0 0.77 0.73
GNB_MLP_SVM	0.57 - 1.0 0.85 0.86	0.63 - 1.0 0.85 0.86	0.0 - 1.0 0.77 0.73
LR_MLP_SVM	0.61 - 1.0 0.85 0.86	0.63 - 1.0 0.85 0.86	0.1 - 1.0 0.77 0.74

was used. The extracted results represent the average over the 5 folds. Rebalancing was used to equalise the instances pertaining to the minority and majority classes, and scaling applied to normalise the data points as to facilitate a better fitting of the models to the data prior to training. The testing data were not rebalanced but the same scaling was applied to them as well.

5.2.2 Obtained Results. The summary of the obtained results for all projects, per feature combination can be found in Table 5, where the range depicts obtained *Min - Max* scores. For this study we have used all combinations of the following Features: Number of Commits Feature(CF), Total Distinct Authors (TDA), Total Churn Feature (TCF), and Total CoCommits Size (TCS). These combinations in all 148 projects produced 2,220 individual results.

Table 4: AUC for the 10 Best Classifier Combinations Compared to Buggy/Clean Ratio Range

Classifier	Avg. F1 and avg. AUC for B/C < 0.25	Avg. F1 and avg. AUC for B/C = [0.25 - 0.5)	Avg. F1 and avg. AUC for B/C ≥ 0.51
DT_LR_RF	0.55 0.87	0.74 0.85	0.84 0.85
DT_GNB_LR_MLP_RF	0.55 0.87	0.74 0.85	0.84 0.85
DT_GNB_MLP_RF_SVM	0.55 0.87	0.74 0.85	0.84 0.85
DT_GNB_LR_MLP_RF_SVM	0.55 0.87	0.74 0.85	0.84 0.85
DT_GNB_LR_RF_SVM	0.55 0.87	0.74 0.85	0.84 0.85
LR_MLP_RF	0.55 0.87	0.74 0.85	0.84 0.85
DT_LR_MLP_RF_SVM	0.55 0.87	0.74 0.85	0.84 0.85
DT_LR_MLP	0.55 0.87	0.74 0.85	0.84 0.85
LR_MLP_RF_SVM	0.55 0.87	0.74 0.85	0.84 0.85
MLP_RF_SVM	0.55 0.87	0.74 0.85	0.84 0.85

5.3 Study 3: Comparison of Process and Source Code Metrics

This study aims at performing a comparison between the efficiency of using Source Code Metrics versus using Process Metrics for carrying out Fault-Proneness prediction. It is carried out on a subset of data made available for software engineering research as part of the PROMISE repository [30].

5.3.1 Study Set-up. All combinations of the available classifiers were used for this experiment and the Feature combination used was the one identified as optimal in Section 5.2. To select the subset of systems on which to conduct this study we manually investigated the contents of the PROMISE repository to identify projects for which a valid Git repository is still available. Given the age of the repository and it's specific structure this process yielded only 11 systems for which process metrics could be mined. For these 11 systems the data between the PROMISE dataset and our system were reconciled. The reconciliation process consisted of only using data pertaining to files present both in data extracted from Git repositories and in the PROMISE dataset. In addition, the files had to be active with at least a single commit in the latest 30% of the total commits of the system. The files' classes were set from the manually curated PROMISE dataset. The process used afterwards is the same as in Sections 5.1 and 5.2. The data were rebalanced in both cases and independently scaled. The process presented so far was designed to give both approaches an equal amount of data and to be easily replicable by other researchers. The evaluation of the models was implemented using k-fold cross validation where k=5 for each of the approaches, Source-Code Metrics and Process Metrics respectively, and the presented results depict the average over all 5-folds of this evaluation.

Table 5: Results of Feature Combinations

Feature Combination	Accuracy Range (Min - Max) Median Accuracy Average Accuracy	AUC Range (Min - Max) Median AUC Average AIC	F1 range (Min - Max) Median F1 Average F1
CF_TCF_TCS_TDA	0.68 - 1.0 0.89 0.89	0.58 - 1.0 0.86 0.86	0.0 - 1.0 0.8 0.76
CF_TCF_TCS	0.7 - 1.0 0.89 0.89	0.55 - 1.0 0.85 0.85	0.0 - 1.0 0.81 0.75
CF_TCS_TDA	0.63 - 1.0 0.88 0.88	0.59 - 1.0 0.85 0.85	0.0 - 1.0 0.8 0.76
CF_TCS	0.66 - 1.0 0.88 0.88	0.6 - 1.0 0.85 0.85	0.0 - 1.0 0.81 0.76
TCF_TCS_TDA	0.72 - 1.0 0.88 0.88	0.64 - 1.0 0.84 0.84	0.0 - 1.0 0.8 0.75
TCF_TCS	0.7 - 1.0 0.87 0.87	0.55 - 1.0 0.82 0.83	0.0 - 1.0 0.78 0.73
TCS_TDA	0.6 - 1.0 0.88 0.87	0.46 - 1.0 0.83 0.84	0.0 - 1.0 0.79 0.74
CF_TCF_TDA	0.73 - 1.0 0.86 0.87	0.6 - 1.0 0.83 0.83	0.0 - 1.0 0.78 0.72
CF_TCF	0.62 - 1.0 0.85 0.86	0.5 - 1.0 0.81 0.82	0.0 - 1.0 0.76 0.71
TCF_TDA	0.69 - 1.0 0.85 0.85	0.61 - 1.0 0.8 0.81	0.0 - 1.0 0.75 0.7
CF_TDA	0.5 - 1.0 0.84 0.85	0.6 - 1.0 0.81 0.82	0.13 - 1.0 0.77 0.71
CF	0.33 - 1.0 0.82 0.84	0.4 - 1.0 0.8 0.81	0.13 - 1.0 0.75 0.69
TCS	0.47 - 1.0 0.84 0.83	0.44 - 1.0 0.79 0.79	0.0 - 1.0 0.71 0.67
TCF	0.63 - 1.0 0.82 0.83	0.5 - 1.0 0.76 0.77	0.0 - 1.0 0.69 0.65
TDA	0.39 - 1.0 0.8 0.81	0.4 - 1.0 0.81 0.81	0.07 - 1.0 0.73 0.68

5.3.2 Obtained Results. For this study a hard-voting classifier was utilised using all combinations of the following classifiers: Decision Trees (DT), Random Forests (RF), Linear Regression (LR), Multi-Layer Perceptron (MLP), Support Vector Machines(SVM), and Gaussian Naive Bayes(GNB). The Features used were: Number of Commits Feature(CF), Total Distinct Authors (TDA), Total Churn (TC), and Total CoCommitsSize Feature (TCS) for extracting process metrics and all features available were used from the PROMISE dataset. This process was applied on 11 projects and yielded a total of 756 results for each metric type. The results are shown in Table 6.

5.4 Study 4: Cross Project Validation

For this study, we have trained the classifiers in a collection of projects (training set) and we have applied them to another set projects (testing set) for comparing the obtained results with the ones obtained when the classifiers are trained and applied only in one project.

5.4.1 Study Set-up. For this study the optimal classifier identified in Section 5.1 and the optimal feature combination identified in Section 5.2 were used. To prepare the data we filtered the available systems selecting only those having less than 20K and more than 250 files and then split these into three performance classes using the ratio of fault-prone over healthy files in the system. This yielded a total of 26 projects with a B/C ratio in the interval [0, 0.25), 20 projects with a ratio in the interval [0.25, 0.5) and 44 projects with a B/C ratio ≥ 0.5 (see also Table 4). These groups were then divided into two randomly selected groups, and one group was used for training a model while the other group was used for evaluation. Given the uneven size of the different systems it was necessary to upsample the data available for each one of them so as to have all projects represented approximately equally in the training set and avoiding it being dominated by the largest systems. Rebalancing of the minority and majority classes was carried out on the upsampled data separately for each project to provide the training algorithm with equal amounts of positive and negative instances. In this study there was no reason to use the k-fold cross validation technique as the evaluation of the trained model happened on other systems than the ones used for training.

Table 6: Process Metrics vs Source Code Metrics

Classifier Combination	Process Metrics Avg. F1 Median F1 Avg. AUC Median AUC	Source Code Metrics Avg. F1 Median F1 Avg. AUC Median AUC
DT_GNB_RF	0.8 0.93 0.77 0.74	0.58 0.58 0.67 0.63
DT_MLP_RF	0.81 0.93 0.77 0.76	0.6 0.61 0.67 0.65
DT_RF_SVM	0.81 0.93 0.75 0.74	0.6 0.62 0.67 0.65
RF	0.8 0.93 0.79 0.76	0.59 0.61 0.66 0.63
DT_LR_RF	0.81 0.92 0.76 0.77	0.61 0.61 0.68 0.65
DT_GNB_MLP_RF_SVM	0.8 0.84 0.75 0.73	0.59 0.59 0.68 0.65
DT_GNB_LR_MLP_SVM	0.79 0.82 0.75 0.75	0.59 0.59 0.68 0.67
GNB_LR_MLP_RF_SVM	0.79 0.82 0.76 0.74	0.58 0.6 0.68 0.66
DT_LR_SVM	0.79 0.82 0.75 0.74	0.58 0.6 0.65 0.65
SVM	0.79 0.82 0.76 0.75	0.57 0.59 0.65 0.64

5.4.2 Obtained Results. For this study a total of 45 results were obtained one for each system used for testing. The results are presented grouped by faulty over healthy ratio in Table 7.

Table 7: Cross Project with Rebalancing

	Cross vs. Within Project	Cross vs. Within Project	Cross vs. Within Project
B/C Value	Avg. Accuracy Accuracy Median	Avg. AUC AUC Median	Avg. F1 F1 Median
B/C < 0.25	0.75 vs. 0.60 0.81 vs. 0.63	0.77 vs. 0.94 0.78 vs. 0.94	0.4 vs. 0.63 0.43 vs. 0.64
B/C $\in [0.25,0.5)$	0.69 vs. 0.76 0.74 vs. 0.76	0.73 vs. 0.86 0.76 vs. 0.86	0.62 vs. 0.78 0.66 vs. 0.77
$B/C \geq 0.5$	0.73 vs. 0.86 0.74 vs. 0.84	0.75 vs. 0.88 0.74 vs. 0.85	0.76 vs. 0.85 0.77 vs 0.84

6 DISCUSSION

6.1 General Observations

The first observation is that each software system is unique, and there is no single best classifier which can be used to provide accurate defect proneness classification results for all projects. What we have observed is that a classifier or a collection of classifiers can produce high performance scores (i.e. high accuracy, high F1, and high AUC) for one project, and poor scores on another (see also "No Free Lunch Theorem" [37]).

The second observation is that we were not able to identify a feature, or a collection of features, that guarantee (i.e. with certainty) that such a classifier or a set of classifiers can be trained to always yield high performance scores. The only measure we have found to be a very good indicator of quality results is the Buggy/Clean ratio, where in the vast majority of cases, if a project has a Buggy/Clean ratio ≥ 0.5 it is almost certain that there exist a classifier or a combination of classifiers which can produce F1 and AUC scores higher that 0.85. This essentially means that classifiers work best when the data (number of buggy vs. clean files) are balanced. Classifiers on projects with Buggy/Clean ratio values less than 0.2 almost certainly perform poorly.

The third general observation is that there is a need to devise techniques for accurate tagging (i.e. to assign a *buggy* or *clean* label to a file) to be used for training purposes. The heuristics used so far in the literature and in this study, may introduce many false positives for training, skewing thus the obtained results. Furthermore, there is a need to devise techniques for introducing a temporal effect on the data, meaning that distant past commits should carry less weight than recent commits.

Overall, *ML* shows to be an interesting technique for defect proneness classification but we have not reached a point yet to identify how these classifiers can be trained effectively to yield trustworthy results for an arbitrary given project.

6.2 Detailed Observations

For research question RQ_1: The combinations of classifiers which included one or more tree-based models performed on average better than any other combination of classifiers. This may have to do with the nature of the problem which lends itself more naturally to a binary type of classification (i.e. *Buggy* or *Clean*) in which tree-based classifiers may perform better. We can say that combinations of classifiers which include Decision Trees and Random Forests, on average, outperform other combinations. Having said that, our observation is that there are no guarantees that these classifiers will perform as well when applied to a new arbitrary project, but there is a higher likelihood they will.

For research question RQ_2: By examining the obtained results the features having the highest probability of generating high quality results is the combination of all four features, followed by the combination of CF (number of commits) and TCS (total co-commit size) and TCF (total churn). However, looking at the minimal set of features which can be used and still produce high quality results is any combination of two of CF (number of commits) and TCS (total co-commit size) and TCF (total churn). Another observation is that the TDA (total distinct authors) feature on its own does not provide high quality results, but only when combined with other features.

For research question RQ_3: Here we can say that process metrics clerly outperform the source code metrics for defect-proneness classification purposes. This is an valuable observation as process metrics are language agnostic and their compilation does not require specialised parsers and metrics extractors. This result can be explained, as software metrics may often exhibit an unjustified variability in their values over different commits of the same file even if it maintains its status (i.e. Buggy or Clean). Vice versa, it can also be the case that software metrics may exhibit no variability of their values over different commits even if the file changes its tag value. This behaviour of software metrics may generate conflicting data for the classifier. In contrast, process metrics may exhibit a variability too, but maintain a better type of "history" feature values as the project evolves.

For research question RQ_4: Here our observation is that cross project classifier training and application does not yield high quality results. When classifiers are trained in some projects and applied to other projects the classification results are not as accurate as when the classifier is applied to the same project as the one it is trained on. This may be explained from the fact that the profiles of process metric values are kind of project specific as they depict the history of the project and the activity on each file. This is an interesting result, as it disputes the case of one trained model fits all.

6.3 Threats to Validity

We identify three threats. The first threat has to do with the way tagging is performed in order to create a training set. For our study we consider for tagging purposes the most recent 30% of the commits, while we maintain historical information from the past 70% of commits. For the feature value vectors of the distant past 70% of the commits we are either providing a DC value or no value (e.g. see Section 4.3). This creates the potential for false positives to be generated during tagging. The second threat has to do with how files are tagged within a single commit. For this study, if a file participates on a bug fixing commit then we consider all files in the commit as buggy. This is an overestimate and introduces the possibility of false positives, similarly fixes-in-passing can introduce false negatives which again degrade the overall quality of the data. The most accurate approach would be being able to tag all files in all commits in the history of the project with their correct label. However, this would be almost impossible for such large projects as we have considered in the course of this study. It would be though a valid approach for new projects, where accurate labeling can commence on early stages of the project. The third

threat has to do with the used features. Since the purpose of the study was to provide results from a large data set and not to propose new features, we have considered features which are commonly used in the research literature. There may be other features which relate to process metrics and which the community has not considered yet, which may produce good defect proneness classification results. This can also be considered an open problem for further investigation.

7 CONCLUSION

This paper reports on the results of a set of experiments conducted in order to evaluate the use of Machine Learning for defect proneness classification. Over the past few years we have seen a tremendous growth on research and publications related to Machine Learning for software maintenance, and in particular for defect proneness classification and defect prediction. Even though there is a significant body of work conducted on evaluating Machine Learning techniques for defect proneness classification, the significance of this paper is that it is the first work to our knowledge that examines such a large corpus of open source data aiming to concretely address four key research questions which relate to experimentally identifying the best classifiers, the best features, whether process metrics outperform software metrics as predictors, and whether cross project training and application can be a viable option with respect to the quality of the obtained results. The experiments conducted revealed a number of observations. First, Machine Learning techniques are not guaranteed to perform well in all projects. Each software project has a specific life-cycle and "personality" profile of its own, and "a one classifier fits all" approach is not feasible. Second, we have seen that Machine Learning techniques are more likely to perform well when the buggy/clean ratio of the system files is between 0.5 and 2. Note that a ratio of 1 indicates that there are as many buggy files and clean files. Third, there was a clear indication that process metrics perform better or, in some cases, at least as well as software metrics. This implies that there is a strong indication that process related metrics can safely be used as predictors. Fourth, training the classifiers in one set of systems and applying on another is not a good approach, as the best classification results are obtained when the classifier is applied on the same system it is trained on. Overall, *ML* for defect proneness classification is a promising area of work that still has a number of open problems to investigate. including, identifying new features, creating better tagging tools, combining *ML* with static and dynamic analysis to increase the performance of the classifiers, and introducing a means of grouping similar projects according to project-wide metrics.

REFERENCES

[1] C. Catal and B. Diri. 2009. Investigating the Effect of Dataset Size, Metrics Sets, and Feature Selection Techniques on Software Fault Prediction Problem. *Inf. Sci.* 179, 8 (2009), 1040–1058.

[2] S. R. Chidamber and C. F. Kemerer. 1994. A metrics suite for object oriented design. *IEEE TSE* 20, 6 (1994), 476–493.

[3] C. Tantithamthavorn et al. 2018. The Impact of Class Rebalancing Techniques on the Performance and Interpretation of Defect Prediction Models. *CoRR* abs/1801.10269 (2018).

[4] C. Tantithamthavorn et al. 2019. The Impact of Automated Parameter Optimization on Defect Prediction Models. *IEEE TSE* 45, 7 (2019), 683–711.

[5] D. Gray et al. 2009. Using the Support Vector Machine as a Classification Method for Software Defect Prediction with Static Code Metrics. In *Engineering Applications of Neural Networks*. 223–234.

[6] F. Touré et al. 2017. Predicting different levels of the unit testing effort of classes using source code metrics: a multiple case study on open-source software. *Innov. in Systems and Software Eng.* 14 (2017).

[7] F. Zhang et al. 2012. An Empirical Study on Factors Impacting Bug Fixing Time. In *19th WCRE*. 225–234.

[8] J. Jiarpakdee et al. 2019. *The Impact of Correlated Metrics on the Interpretation of Defect Models,*. IEEE TSE.

[9] L. Pascarella et al. 2019. Fine-grained just-in-time defect prediction. *JSS* 150 (2019), 22–36.

[10] M. D'Ambros et al. 2012. *Evaluating Defect Prediction Approaches: A Benchmark and an Extensive Comparison.* EMSE Vol. 17 pp.531-577.

[11] M. Kondo et al. 2019. *The Impact of Feature Reduction Techniques on Defect Prediction Models.* EMSE 24:(4) pp. 1925-1963.

[12] P. He et al. 2015. *An Empirical Study on Software Defect Prediction with a Simplified Metric Set.* Inf. Softw. Technol. 59C pp. 170-190.

[13] S. Lessmann et al. 2008. Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings. *IEEE TSE* 34, 4 (2008), 485–496.

[14] T. Hall et al. 2012. A Systematic Literature Review on Fault Prediction Performance in Software Engineering. *IEEE TSE* 38, 6 (2012), 1276–1304.

[15] T. Menzies et al. 2007. Data Mining Static Code Attributes to Learn Defect Predictors. *IEEE TSE* 33, 1 (2007), 2–13.

[16] T. Menzies et al. 2010. Defect prediction from static code features: current results, limitations, new approaches. *Automated Soft. Engineering* 17, 4 (2010), 375–407.

[17] T. Zimmermann et al. 2009. Cross-Project Defect Prediction: A Large Scale Experiment on Data vs. Domain vs. Process. In *7th ESEC/FSE*. 91–100.

[18] V. R. Basili et al. 1996. A validation of object-oriented design metrics as quality indicators. *IEEE TSE* 22, 10 (1996), 751–761.

[19] V. U. B. Challagulla et al. 2005. Empirical assessment of machine learning based software defect prediction techniques. In *10th IEEE International Workshop on OO Real-Time Dependable Systems*. 263–270.

[20] Y. Kamei et al. 2013. A large-scale empirical study of just-in-time quality assurance. *IEEE TSE* 39, 6 (2013), 757–773.

[21] Z. Tóth et al. 2016. A Public Bug Database of GitHub Projects and Its Application in Bug Prediction. In *ICCSA*. 625–638.

[22] Github. 2008. Github, Build software better, together - https://github.com.

[23] A. E. Hassan. 2009. Predicting faults using the complexity of code changes. In *31st ICSE*. 78–88.

[24] T. M. Khoshgoftaar and N. Seliya. 2002. Tree-based software quality estimation models for fault prediction. In *Proceedings Eighth IEEE Symposium on Software Metrics*. 203–214.

[25] G. Lemaître, F. Nogueira, and C. K. Aridas. 2017. Imbalanced-Learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. *J. Mach. Learn. Res.* 18, 1 (2017), 559–563.

[26] Nagappan et al. N. 2006. Mining Metrics to Predict Component Failures. In *28th ICSE*. 452–461.

[27] A. Okutan and O.T. Yundefinedldundefinedz. 2014. *Software Defect Prediction Using Bayesian Networks*. Vol. 19. Kluwer Academic Publ., USA. 154–181 pages.

[28] F. Rahman and P. Devanbu. 2013. How, and why, process metrics are better. In *35th ICSE*. 432–441.

[29] D. Romano and M. Pinzger. 2011. Using source code metrics to predict change-prone Java interfaces. In *27th ICSM*. 303–312.

[30] J. Sayyad Shirabad and T.J. Menzies. 2005. The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada. http://promise.site.uottawa.ca/SERepository

[31] C. Tantithamthavorn and A.E. Hassan. 2018. An Experience Report on Defect Modelling in Practice: Pitfalls and Challenges. In *40th ICSE*. 286–295.

[32] M.M.T. Thwin and T.S. Quah. 2005. Application of Neural Networks for Software Quality Prediction Using Object-Oriented Metrics. *JSS* 76, 2 (2005), 147–156.

[33] A. Tornhill. 2015. *Your Code as a Crime Scene: Use Forensic Techniques to Arrest Defects, Bottlenecks, and Bad Design in Your Programs.* Pragmatic Bookshelf.

[34] B. Turhan and A. Bener. 2009. Analysis of Naive Bayes' Assumptions on Software Fault Data: An Empirical Study. *Data Knowl. Eng.* 68, 2 (Feb. 2009), 278–290.

[35] Unnamed. 2020. Study data. https://figshare.com/s/cc5bfdefd91442712b0b

[36] S. Wang and X. Yao. 2013. Using Class Imbalance Learning for Software Defect Prediction. *IEEE Trans. on Reliability* 62, 2 (2013), 434–443.

[37] D. H. Wolpert and W. G. Macready. 1997. No free lunch theorems for optimization. *IEEE Trans. on Evolutionary Computation* 1, 1 (1997), 67–82.

[38] T.T. Wong. 2015. Performance Evaluation of Classification Algorithms by K-Fold and Leave-One-out Cross Validation. *Pattern Recogn.* 48, 9 (2015), 2839–2846.

[39] J. Zheng. 2010. Cost-Sensitive Boosting Neural Networks for Software Defect Prediction. *Expert Syst. Appl.* 37, 6 (June 2010), 4537–4543.

Moving from Cross-Project Defect Prediction to Heterogeneous Defect Prediction: A Partial Replication Study

Hadi Jahanshahi
Data Science Lab
Ryerson University
Toronto, Ontario, Canada
hadi.jahanshahi@ryerson.ca

Mucahit Cevik
Data Science Lab
Ryerson University
Toronto, Ontario, Canada
mcevik@ryerson.ca

Ayşe Başar
Data Science Lab
Ryerson University
Toronto, Ontario, Canada
ayse.bener@ryerson.ca

ABSTRACT

Software defect prediction heavily relies on the metrics collected from software projects. Earlier studies often used machine learning techniques to build, validate, and improve bug prediction models using either a set of metrics collected within a project or across different projects. However, techniques applied and conclusions derived by those models are restricted by how identical those metrics are. Knowledge coming from those models will not be extensible to a target project if no sufficient ov erlapping me trics have been collected in the source projects. To explore the feasibility of transferring knowledge across projects without common labeled metrics, we systematically integrated Heterogeneous Defect Prediction (HDP) by replicating and validating the obtained results. Our main goal is to extend prior research and explore the feasibility of HDP and finally t o c ompare i ts p erformance w ith that of its predecessor, Cross-Project Defect Prediction. We construct an HDP model on different publicly available datasets. Moreover, we propose a new ensemble voting approach in the HDP context to utilize the predictive power of multiple available datasets. The result of our experiment is comparable to that of the original study. However, we also explored the feasibility of HDP in real cases. Our results shed light on the infeasibility of many cases for the HDP algorithm due to its sensitivity to the parameter selection. In general, our analysis gives a deep insight into why and how to perform transfer learning from one domain to another, and in particular, provides a set of guidelines to help researchers and practitioners to disseminate knowledge to the defect prediction domain.

KEYWORDS

Defect Prediction; Heterogeneous Metrics; Transfer Learning; Software Quality

ACM Reference Format:
Hadi Jahanshahi, Mucahit Cevik, and Ayşe Başar. 2020. Moving from Cross-Project Defect Prediction to Heterogeneous Defect Prediction: A Partial Replication Study. In *Proceedings of 30th Annual International Conference on Computer Science and Software Engineering (CASCON'20).* IBM Corp., Riverton, NJ, USA, 10 pages.

CASCON'20, November 10-13 2020, Toronto, Canada

1 INTRODUCTION

Defect prediction is a powerful technique proposed to allocate limited testing resources efficiently and minimise the risk associated with incurring post-release defects. This technique helps software teams to prioritise potential defective modules (e.g., files or functions) in the software systems. In most cases, the metrics extracted from the historical defect dataset of a project are used to train a binary classifier to predict the defects of new software modules from the same project. This type of prediction is called Within-Project Defect Prediction (WPDP). However, this approach does not apply to the case where new software is launched, and there is no significant evidence about the characteristics of bugs in the system.

In Cross-Project Defect Prediction (CPDP) [27, 33], we may use another project, with the same metrics as those of the testing set, to build a model and create a promising prediction. CPDP aims to predict bugs in a new project that lacks historical defect data. This process constructs a model from a similar but not identical project. Nevertheless, in practice, collecting datasets with the same metrics is not feasible for all projects. To alleviate the limitations due to heterogeneous metric sets in CPDP, Nam et al. [17] proposed Transfer Learning or Heterogeneous Defect Prediction (HDP). They presented an HDP method exploiting gain ratio feature selection and metrics matching based on the Kolmogorov-Smirnov test (KS-test) to choose the metrics with similar distribution in the training set. There are other variations of the HDP method. Gong et al. [8] suggested an unsupervised deep domain adaption method to overcome the issues with heterogeneous metrics and unbalanced datasets. Li et al. [15] proposed a two-stage ensemble learning for HDP to alleviate the problem of linear inseparability and class imbalance.

We first partially replicate the work by Nam et al. [17], and then investigate new research questions on the possibility of transferring the lessons learned from traditional defect prediction to a new project with different features. To systematically explore the issue, we construct our study along with the following four research questions:

- **RQ1**: How do models selected using domain-agnostic similarity perform in a cross-project context?
- **RQ2**: How do HDP methods predict the defect in the system compared to WPDP methods?
- **RQ3**: How do the ensembles of models built from several projects perform in an HDP context?
- **RQ4**: How feasible is HDP in terms of target prediction coverage?

The remainder of the paper is organised as follows. Section 2 presents a brief background of the original study. In Section 3, we provide an overview of the replication study. Afterward, In Section 4, the comparison of the result with the original study has been discussed. Moreover, we report the performance of the approach under different scenarios and investigate the coverage ability of HDP. Section 5 discusses the threats to the validity of the paper, and finally, Section 6 concludes the paper.

2 INFORMATION ABOUT THE ORIGINAL STUDY

This paper reports on the replication and extension of a paper, established upon the proposed guidelines for experimental replications [1].

Nam et al. [17] proposed HDP to circumvent the severe limitation of CPDP: the need for a homogeneous set of metrics; i.e., different projects should have the same (or at least sufficient common) metrics to make CPDP work. The notion of transfer learning [18, 30] highlights the importance of exploiting all available resources, even if the input feature space and data distribution characteristics are not identical. In the original study [17], authors examine the possibility of quickly transferring lessons learned from defect datasets to new, unseen datasets.

The proposed methodology aims to employ the similarity between the distribution of heterogeneous metrics to address the restrictions on CPDP. After applying feature selection on the training set (i.e., source project), they use metrics matching to find the best-matched metrics in the testing set (i.e., target project) for each selected metric of the training set. This step may lead to infeasible HDP if the method is unable to find any paired metrics. They discussed the HDP target coverage to report the success rate of their method. After obtaining the best-matched set of metrics, they build a classifier on the source project and predict the defect-proneness of the target project. Nam et al. [17] evaluate the performance of HDP based on the Area Under the ROC curve metric. They reported the representative HDP results using Gain Ratio for feature selection (top 15% metrics), KS-test with the cutoff threshold of 5% ($p = 0.05$) for similarity finding, and the Logistic Regression (LR) as the classifier. Table 1 gives a summary of the research questions and the steps followed in their paper. More details on their method are provided in Section 4.2.

3 INFORMATION ABOUT THE REPLICATION

3.1 Benchmark datasets

In this paper, we abide by the publicly available datasets that were used in earlier studies. The summary of the datasets is provided in Table 2. Nonetheless, in some cases, e.g., NASA datasets, more than one version of the datasets were available and some concerns about the quality of the dataset have been reported [9, 22]. Hence, we established three important inclusion criteria to filter out unreliable datasets [23]:

(1) **Criterion 1- different corpora**: Since we are to implement HDP, we need to choose our datasets from a multitude of sources where there is a chance of inconsistency between column names. Furthermore, this factor will augment the generalizability of our conclusions.
(2) **Criterion 2- Sufficient EPV**: The number of Events Per Variable (EPV) is the ratio of buggy instances to the number of predictors (metrics). For instance, in JDT dataset, EPV is $206/19 = 10.85$. To avoid problems related to the overfitting, underfitting, and misleading associations, general guidelines have been suggested for the minimum EPV required in multivariate analysis to be from 10 to 20 [24]. We set the minimum acceptable EPV to 10 and discard any dataset whose EPV is less than 10 since few events relative to the number of independent variables produce unreliable results.
(3) **Criterion 3- defect ratio**: It is implausible to have more defective software modules than those that are free of bugs. Accordingly, we remove datasets whose defective rate is more than half ($> 50\%$).

Each group has a different set of measures in terms of the number and type, making CPDP infeasible in many cases. Therefore, we consider HDP as a remedy for incongruous metrics.

We analyse 136 publicly available defect datasets from various sources [5, 6, 10, 12, 19, 22, 26, 31, 32]. We apply the inclusion criteria to sift qualified datasets out. 114 datasets have EPV less than 10, and 5 datasets have defective ratio greater than 50%; consequently, we arrive at 17 eligible datasets. In Table 2, the number of instances, the number and the percentage of buggy instances per dataset, the number of metrics, and the EPV score of those datasets are reported.

3.2 Level of interaction with authors of original study

In this replication study, the original researchers, [17], provided us most of the datasets; we queried about the detail of some parts that were not clearly mentioned in the paper, and the authors responded to our email. Since the scripts of the paper were not available, all the codes have been written from scratch, and we did not request the related codes from the authors.

3.3 Changes to the original experiment

We had original datasets, to some extent, to create our benchmark; however, several new datasets are incorporated to validate the generalizability of the original paper. The replicated research questions and the design of the experiments are similar to those of the original study with the following modifications:

- We define three new Research Questions to first investigate whether HDP has the transferability to be applied on a dataset with a totally different nature, and second whether the performance of the HDP can be improved through pooling and voting approaches.
- Different points of view into the problem for each Research Question are embodied in research.
- The original study used only Logistic Regression (LR) as a classifier, whereas we implement both LR and Random Forest (RF) with the number of trees equal to 100.
- Unlike the original study, we consider 136 different datasets, and based on the predefined criteria we choose only 17 reliable projects. Hence, only 5 datasets, namely JDT, Mylyn,

Table 1: The overview of research questions and the steps taken in paper by Nam et al. [17]

Project data used	EQ, JDT, LC, ML, PDE, Apache, Safe, ZXing, ant-1.3, arc, camel-1.0, poi-1.5, redaktor, skarbonka, tomcat, velocity-1.4, xalan-2.4, xerces-1.2, cm1, mw1, pc1, pc3, pc4, jm1, pc2, pc5, mc1, mc2, kc3, ar1, ar3, ar4, ar5, ar6
Language	Java / Weka
Preprocessing phase	1) Feature Selection (Gain Ratio* , chi-square, relief-F, and significance attribute evaluation) 2) Training/Testing sets' metrics matching (Kolmogorov-Smirnov test based matching*, percentile based matching, and Spearman's correlation based matching)
Classifiers	Simple logistic, Logistic regression*, Random Forest, Bayesian network, Support vector machine, J48 decision tree, and Logistic model tree
Cross-Validation Method	500 times 2-fold stratified CV
Evaluation Method	Area Under the Curve (AUC) of the Received Operating Characteristic (ROC)
Statistical Tests/Methods	Wilcoxon signed-rank test, Cliff's δ, and Kolmogorov-Smirnov test
Research Questions	**RQ1: Is Heterogeneous Defect Prediction Comparable to WPDP, Existing CPDP Approaches for Heterogeneous Metric Sets, and Unsupervised Defect Prediction? RQ2: What Are the Lower Bounds of the Size of Source and Target Datasets for Effective HDP?**

Items marked with an asterisk (*) are the main methods that are used in the research question. The rest are competitive approaches to check the validity of the model.

Table 2: Summary of the project data

Group	Dataset	Abbreviation	buggy (%)	# of instances	buggy (#)	# of metrics	EPV
Eclipse	JDT [5, 6]	JDT	20.7	997	206	19	10.8
	Mylyn [5, 6]	ML	13.2	1862	245		12.9
	PDE [5, 6]	PDE	14.0	1497	209		11.0
	Debug 3.4 [12]	DG	24.7	1065	263	17	14.6
	SWT 3.4 [12]	SWT	44.0	1485	653		38.4
Proprietary	Prop-1 [10]	PR1	14.8	18471	2738	20	130.4
	Prop-2 [10]	PR2	10.6	23014	2431		115.8
	Prop-3 [10]	PR3	11.5	10274	1180		56.2
	Prop-4 [10]	PR4	9.6	8718	840		40.0
	Prop-5 [10]	PR5	15.3	8516	1299		61.9
Apache	Camel 1.2 [10]	CML	35.5	608	216	20	10.8
	Xalan 2.5 [10]	XN2.5	48.2	803	387		19.4
	Xalan 2.6 [10]	XN2.6	46.4	885	411		20.6
Jira	Derby 10.2.1.6 [32]	DY.2	33.7	1963	661	65	10.2
	Derby 10.3.1.4 [32]	DY.3	30.3	2206	669		10.3
NASA	JM1 [22]	JM1	21.5	7782	1672	21	79.6
	PC5 [22]	PC5	27.5	1711	471	38	12.4

All the datasets can be downloaded via github.com/HadiJahanshahi/Replication-HDP.

PDE, JM1, and PC5, are common with the original study, and the remaining 12 projects are new.

From the research questions of this study reported in Section 1, RQ2 is identical to the original study, and the rest are newly defined.

4 COMPARISON OF RESULTS WITH ORIGINAL STUDY

Our partial replication study includes one of the research questions of the original study and three new research questions. Therefore, we extend their work and provide a comparison for RQ2 that is similar to the previous work.

In the first step, we experiment with all cross-project model permutations for the 17 available datasets ($2 \times \binom{17}{2} = 272$ pairs). For each pair, one project is selected to be the testing set and the others as the training set. AUC values of the experiment have been computed and listed in Table 3. Note that many of the cells are empty in the table since we were unable to find enough common metrics to develop the CPDP method. In most cases, sufficient common metrics are found when we choose the training and testing set within the same group. To measure WPDP, we applied 10-fold cross-validation on a single dataset. The 10-fold cross-validation divides the dataset to 10 folds, and in each step, one fold will be used as the testing set and the remaining nine folds as the training set. Since cross-validation is highly affected by the data which is randomly selected in each fold [13], we repeated the process ten times to overcome the randomness issue. Accordingly, each boldface AUC value on the diagonal of Table 3 is the result of aggregating (taking the average of) 10×10-fold cross-validation. Off-diagonal values indicate CPDP's performance using one dataset to predict the defects of another dataset. Therefore, no CV has been done for off-diagonal values, and they are obtained from a single experiment.

In the HDP context, there exists a prediction potential of other mature projects that remains intact. Any missing values in Table 3 offer an unused potential that requires exploration. In Section 4.2.2, we aim to utilise the capability of the other projects in defect prediction.

Using beanplots, Figure 1 demonstrates the cross-project performance of the models scaled (normalised) by the AUC values of the within-project model. The projects are sorted based on their within-project AUC, starting with the SWT 3.4 project (AUC value of 0.9) and ending to Camel 1.2 (AUC value of 0.65). Since they are sorted in descending order based on their WPDP's AUC values, we expect to see a decrease in the AUC of CPDP; however, such a pattern does not emerge. Similar to previous studies [11, 27], it can be concluded that the remarkable performance of WPDP in models is not a reliable indicator of that of CPDP.

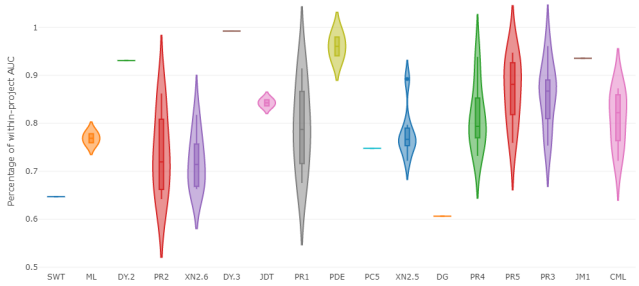


Figure 1: Relation between the performance of CPDP and WPDP. After normalising off-diagonal AUC values in Table 3 by Diagonal of the matrix (columnwise), we sorted projects based on their WP performance along the x-axis.

4.1 RQ1: How do models selected using domain-agnostic similarity perform in a cross-project context?

To address the first Research Question, we used the domain-agnostic similarity of the projects introduced by Kamei et al. [11]. Algorithm 1 lists the pseudocode to compute the domain-agnostic distance between two projects. First, the Spearman correlation of all metrics concerning the dependant variable (buggy or not) is calculated. Then, we choose the top 3 correlated metrics from the project and select the same ones from another project. Next, the pairwise Spearman correlation between selected metrics for each project is computed ($\binom{3}{2}$), and for each project, an array of 3 elements is generated. The Euclidean distance between these arrays informs us about the similarity of the projects.

Using Algorithm 1, we report the pairwise similarity of all the projects in Table 4. We select the most similar project to the testing project as our training set. For instance, in order to predict bugs in JDT project, we select PDE ($dist = 0.11$) which has the shortest distance from the test set.

Figure 2 shows the AUC values of the models selected by the domain-agnostic similarity. The values are normalised by WP performance. We use a t-test to check the hypothesis whether the true mean of domain-agnostic models normalised by WPDP is equal to 1 or not. The *p-value* of 0.83 indicates WPDP outperforms the suggested defect prediction model at the significance level of 0.05 ($\alpha = 5\%$).

Algorithm 1: Domain-agnostic Dissimilarity calculation

```
for metrics ∈ Trainset do
    compute Spearman correlation wrt a dependant variable (label)
end
pick the top 3 metrics based on their Spearman correlation ∈ Trainset ∧ the same metrics ∈ Testset
for i ∈ {1, ..., (3 2)} do
    Q_i ⇐ pairwise Spearman correlation of Trainset's picked metrics
    R_i ⇐ pairwise Spearman correlation of Test set's picked metrics
end
return EuclideanDistance (Q, R)
```

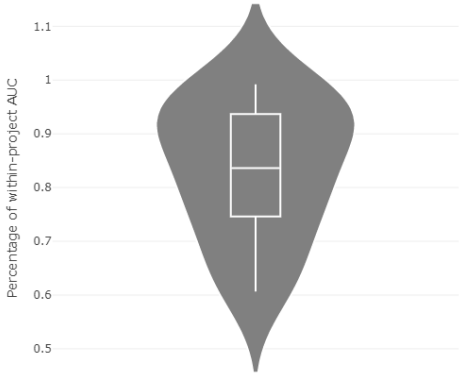


Figure 2: RQ1: the performance of the models selected by the domain-agnostic similarity.

4.2 RQ2: How do HDP methods predict the defect in the system compared to WPDP methods?

One of the main issues of CPDP in traditional defect prediction is its poor performance [27, 33]. However, some methods, such as Peters filter and Transfer Naive Bayes, have been applied to improve the performance of CPDP [20]. The main drawback of cross-project defect prediction that is yet to be addressed is the heterogeneous metrics that different projects may have. In some cases, the overlap of the metrics between the projects is slight or absent, making the CPDP almost infeasible. Therefore, a prediction model cannot be built on either group to predict defects in the other. The heterogeneous metrics in different datasets and metrics collection tools have been reported elsewhere [16, 17, 21].

Nam et al. [17] proposed an HDP approach to address the limitation of CPDP. First, they apply a feature selection technique on the source project (i.e., training set) to remove noise or irrelevant

Table 3: Summary of AUC values for WPDP (boldface) and CPDP. Rows of the table indicate training projects, and the Columns are testing project. Each cluster of AUC performances demonstrates a unique system. (Rows are training, and columns are test sets.)

	JDT	ML	PDE	CML	PR1	PR2	PR3	PR4	PR5	XN2.5	XN2.6	DY.2	DY.3	DG	SWT	JM1	PC5
JDT	**0.81**	0.86	0.74														
ML	0.77	**0.93**	0.71														
PDE	0.81	0.81	**0.77**														
CML				**0.65**	0.55	0.58	0.59	0.51	0.59	0.62	0.66						
PR1				0.54	**0.75**	0.65	0.64	0.71	0.65	0.50	0.54						
PR2				0.56	0.71	**0.72**	0.69	0.70	0.69	0.58	0.69						
PR3				0.58	0.67	0.69	**0.72**	0.65	0.71	0.58	0.66						
PR4				0.54	0.70	0.62	0.62	**0.76**	0.62	0.61	0.68						
PR5				0.59	0.68	0.69	0.70	0.65	**0.71**	0.61	0.69						
XN2.5				0.59	0.52	0.50	0.53	0.59	0.52	**0.69**	0.70						
XN2.6				0.60	0.58	0.63	0.63	0.60	0.64	0.64	**0.82**						
DY.2												**0.86**	0.73				
DY.3												0.84	**0.83**				
DG														**0.73**	0.66		
SWT														0.65	**0.94**		
JM1																**0.69**	0.70
PC5																0.65	**0.73**

Table 4: Domain-agnostic Distance of the Projects. (Rows are training and columns are testing sets.)

	JDT	ML	PDE	CML	PR1	PR2	PR3	PR4	PR5	XN2.5	XN2.6	DY.2	DY.3	DG	SWT	JM1	PC5
JDT	0.00	0.06	0.13														
ML	0.49	0.00	0.54														
PDE	0.11	0.04	0.00														
CML				0.00	0.20	0.14	0.13	0.25	0.26	0.28	0.18						
PR1				0.30	0.00	0.13	0.13	0.15	0.13	0.54	0.81						
PR2				0.11	0.11	0.00	0.06	0.11	0.13	0.44	0.68						
PR3				0.39	0.13	0.08	0.00	0.05	0.06	0.60	0.86						
PR4				0.23	0.12	0.06	0.06	0.00	0.14	0.48	0.69						
PR5				0.42	0.13	0.13	0.06	0.04	0.00	0.65	0.91						
XN2.5				0.34	0.43	0.32	0.35	0.41	0.35	0.00	0.11						
XN2.6				0.32	0.09	0.30	0.30	0.43	0.13	0.13	0.00						
DY.2												0.00	0.04				
DY.3												0.04	0.00				
DG														0.00	0.40		
SWT														0.57	0.00		
JM1																0.00	0.05
PC5																0.02	0.00

metrics. Then, based on the similarity between the distribution of the selected metrics and those of the target project (testing set), a list of the most similar metrics between the two datasets is selected. In this case, metrics' labels may differ. Finally, a classifier is built and trained using the set of the matched metrics and validated on the target project. Figure 3 demonstrates the three main steps of the HDP approach. Note that both the labels and the number of the metrics may differ ($X_1 to X_n$ compared to $Y_1 to Y_m$).

Although there is no best single feature selection technique for all defect prediction models [2, 29], Nam et al. [17] reported that the Gain-Ratio metric selection has the best performance on HDP. In the feature selection part, the top 15% of the metrics have been selected from the source project, as suggested by Gao et al. [7]. For each selected metric in the source project, the HDP approach computes the similarity between this metric and all metrics in the target project. In the original paper, the KS-test based matching approach has been implemented to find the best pair for each selected metric. Since the KS-test is non-parametric, no presumption of the distribution, e.g., normality, is needed. Statistically, the *p-value* of the test indicates whether the distribution of the two metrics is similar. The more similar the metrics are, the closer to 1 the *p-value* would be. Identical to the original experiment, we define the matching score of metric i of the source project with metric j of the target project as:

$$M_{ij} = p\text{-}value_{ij} \quad \text{of KS-test.} \tag{1}$$

After finding all pairwise similarities (*p-values*) between the selected source metrics and target metrics, we define a cutoff threshold to eliminate loosely correlated metrics. In Figure 4, if the cutoff threshold is 0.4, the poorly matched metrics will be removed, and we arrive at the top right graph. However, if our expectation of similarity is higher, the cutoff threshold of 0.8 reaches to infeasible HDP since the number of matched arcs (acceptable scores) is less than the number of source nodes (source metrics). We define Target Prediction Coverage (TPC) as the percentage of the target projects that can be predicted by HDP. The lower TPC is, the less viable HDP will be. In the original paper, the optimistic cutoff threshold of 0.05 is selected, and we use the same value here. Moreover, our HDP design is the same as that of the original study.

After applying the cutoff threshold, if HDP is feasible (i.e. cutoff threshold of 0.4 in Figure 4), the Maximum Weighted Bipartite Matching (MWBM) technique is used to decide on a group of matched metrics whose sum of matching scores is higher. In the top right graph of Figure 4, for metrics Y_m we have only one option: $X_{n'}$. Hence, this pair will be assigned and the arc of $X_{n'}$-Y_m will be removed. Thus, for Y_1 there will be one option to be selected: X_1.

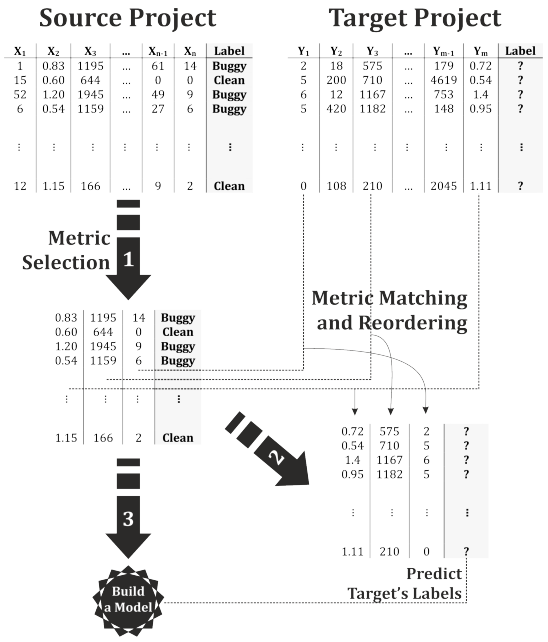


Figure 3: Heterogeneous Defect Prediction (HDP) [17]

Lastly, the pair of X_2-Y_2 will be created and the corresponding set of matches ($\{X_1\text{-}Y_1, X_2\text{-}Y_2, X_{n'}\text{-}Y_m\}$) will be obtained.

After attaining a feasible set of metrics, the prediction model will be trained on the source project. We use both RF and the LR classifier to predict defects in the target set. We compare the result with WPDP as the baseline. Similar to the previous study, in WPDP, no feature selection technique has been applied to the training set.

4.2.1 Experimental Design. From Table 2, we use 17 eligible projects to validate HDP. WPDP can be applied if and only if a dataset is split into training and testing sets. For this purpose, we repeat randomly 2-fold cross-validation (CV) 100 times. For each project, 200 testing sets are obtained. When conducting the 2-fold CV, we used a stratified CV in which the buggy rate of both folds will be the same as that of the original datasets. In WPDP, we use the remaining 200 training sets to train the model; therefore, we obtain 3400 (= 17 × 200) AUC values for the within-project experiment. On the other hand, in HDP, for each 200 test set, we train on all other 16 projects to check the feasibility of HDP and the accuracy of the model. Although in the optimal case, we should obtain 54400 (= 17 × 16 × 200) AUC values for HDP, we face several infeasible cases in which, based on the cutoff threshold, no similar metric can be selected between the source and target project. In Table 5, a typical output of HDP is presented. In the provided result, "Project 3" is unable to have any prediction on "Project 1" since all 200 (= 2 × 100) values in the subsequent columns are "NaN". In addition to infeasible HDP, in the first row, "Project 2" is able to achieve an absolute prediction on "Project 1". Nevertheless, in most pairwise predictions, neither is the case. Out of 200 repetitions, some produce feasible HDP while others fail. Therefore, a threshold on the acceptable number of predictions is required to claim whether HDP between two projects is feasible.

Figure 5 shows different cutoff thresholds for the percentage of acceptable NaNs. When we permit only 1% NaNs out of 200 replications, the feasibility of HDP reduces to 19.9% (= 54 feasible HDPs out 272). When we increase the threshold to 50%, the power of HDP increases to 27.8% (= 75 feasible HDPs out 272), and finally, when we accept 99% of NaNs that seems optimistic, only 114 feasible cases will be obtained (41.9%). Therefore, the KS-test with a cutoff of 0.05 reaches a few feasible HDPs ranging between 19.9% (in the pessimistic case) to 41.9% (in the optimistic case) of the total replications. Notwithstanding the limitation of HDP in prediction, the original paper does not mention the cutoff threshold for the acceptable number of NaNs. The authors of the original study only mentioned the total number of feasible cases being 284 out of 962, which is close to the optimistic result that we reported in our replication experiment.

4.2.2 Prediction Performance. We investigate the representative HDP results based on Gain Ratio feature selection, K-S Test with a cutoff threshold of 0.05, the LR classifier (in addition to RF), and Area Under the ROC Curve (AUC). We also set the percentage of acceptable NaNs in the prediction to 99%. Since each dataset has a different set of metrics, the only acceptable baseline to compare with is the WPDP.

Table 6 compares the performance of HDP (mean AUC) with the baseline. The first column demonstrates the result of WPDP based on the mean of AUC values for 200 replications on the target (test) dataset. The performance of HDP on the target set using other datasets is shown in the third column. The values are obtained through feasible HDPs. For example, JDT project can be predicted by other projects (from the same or different groups) in 34% of the cases. The infeasible HDPs acquired from the remaining projects are ignored. For the target project JDT, there exist 16 other projects as the source project (train set). Furthermore, we split the target to 2 folds 100 times, and, in total, the potential 3200 cases of pairwise prediction can be generated. The fourth column indicates that out of these 3200 cases, only 1087 of them are feasible for JDT project. Therefore, the average rate of predictability is only 28.1%. It indicates that in 71.9% of the replications, no common metrics (similar metrics based on KS-test with a cutoff of 0.05) have been found. The second column reports the magnitude of the effect size between WPDP and HDP in terms of Cliff's δ [3]. Cliff's δ ranges from +1 to -1. The more the Cliff's δ is, the better the HDP performance will be in comparison to WPDP. The estimate of Cliff's δ magnitude (N: Negligible, S: Small, M: Medium, and L: Large) illustrates how significant the statistics are. In all cases, we encounter significant, negative numbers, indicating the baseline outperforms the proposed HDP.

As previously reported in Section 4.2.1, by setting the cutoff threshold to 99% for NaNs acceptance, 114 feasible HDP models would be achieved. To investigate further the results of the mean and Cliff's δ comparison, we applied Wilcoxon signed-rank test [28] to the results. Using an alpha level of 5% ($\alpha = 0.05$), we build a pairwise Wilcoxon test between all feasible HDPs and corresponding WPDPs. Using LR Classifier, in all cases, WPDP outperforms while this ratio is even higher when compared with RF. This observation is different than the original study as we apply different filtering

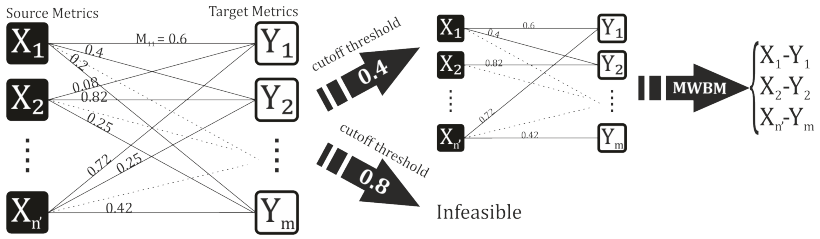


Figure 4: How to match metrics of Source and Target Project. (MWBM: the Maximum Weighted Bipartite Matching)

Table 5: A sample result of the output of HDP. NaNs indicate infeasible HDP. It includes the result of 2-fold cross-validation that is randomly repeated 100 times for each pair of projects. Therefore, the total number of columns is 202 (=2 + 100 × 2), and the total number of rows is 272 (= 17 × 16).

Train	Test	CV1-1	CV1-2	CV2-1	CV2-2	⋯	CV199-2	CV200-1	CV200-2
Project 2	Project 1	0.75	0.92	0.98	0.84		0.57	0.90	0.65
Project 3	Project 1	NaN	NaN	NaN	NaN	⋱	NaN	NaN	NaN
Project 4	Project 1	0.72	NaN	0.65	0.92		0.67	NaN	0.89
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Project 14	Project 17	0.82	0.62	NaN	0.79		0.91	NaN	1.00
Project 15	Project 17	1.00	0.71	0.32	NaN	⋱	NaN	0.65	0.95
Project 16	Project 17	NaN	0.51	0.73	0.48	⋯	0.82	0.62	NaN

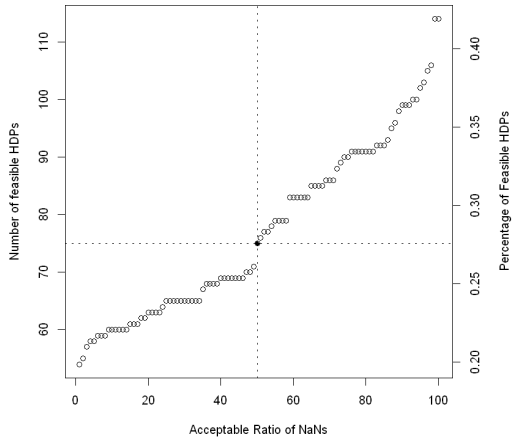


Figure 5: Feasibility of HDP on 17 defect prediction datasets. The number of feasible HDP out of 272 pairwise HDPs, together with its acceptable ratio of NaNs, is reported.

mechanisms and report infeasible cases that are ignored in that paper.

Statistically, it can be concluded that HDP is not a viable alternative for WPDP. For example, in Table 6, PDE was predicted in 8 prediction combinations, and WPDP outperforms in six combinations, and they tie in two cases.

4.3 RQ3: How do the ensembles of models built from several projects perform in an HDP context?

In RQ2, we found that the performance of HDP is considerably worse than the baseline, WPDP. Other papers reported the same issue of poor performance in HDP due to class imbalance problem, complex nonlinear relationship between source and target datasets, and information loss in constructing similar distributions [14, 25]. Therefore, we propose an ensemble of learners in case of a lack of common metrics to have a better HDP.

For this research question, we follow the same steps of HDP, shown in Figure 3. For each project, we first choose only projects that are considered as feasible based on the KS-test with a cutoff threshold of 0.05. Then, we build a model on each of them and predict the label of each instance in the target project. Afterward, each model gives us a probability of being buggy, and we take the average of the vote of all feasible models as the probability of defectiveness. In this case, instead of pairwise prediction, we have an ensemble of models voting for the correct label of the target set. Unlike RQ2, we do not use any cross-validation on the target set; instead, the whole target project is considered as the test set.

Table 7 shows the difference in the performance of ensemble voting compared to normal HDP. We compare the mean of both methods using the Wilcoxon signed-rank test with continuity correction, and the result for both classifiers, LR and RF, demonstrates significant improvement, with *p-value* of 0.0058 and 0.0069 respectively ($\alpha = 0.05$). Hence, we recommend using the potential of all feasible datasets instead of pairwise HDP implementation.

Table 6: Comparing the performance of HDP and WPDP (by mean AUC of all feasible HDPs).

	Logistic classifier (RWeka package in R)								Random Forest		
Target	**WPDP (mean)**	**WPDP (Cliff's δ)**	**HDP (mean)**	**Predictability #**	**Predictability %**	**Win**	**Tie**	**Loss**	**Win**	**Tie**	**Loss**
JDT	0.821	-0.992 (L)	0.589	1087	34.0%	0	1	8	0	1	8
ML	0.920	-1.000 (L)	0.547	1245	38.9%	0	0	8	0	0	8
PDE	0.768	-0.997 (L)	0.556	1107	34.6%	0	2	6	0	2	6
DG	0.722	-0.967 (L)	0.552	1290	40.3%	0	0	7	0	0	7
SWT	0.879	-1.000 (L)	0.529	240	7.5%	0	0	3	0	0	3
PR1	0.745	-1.000 (L)	0.500	401	12.5%	0	1	2	0	1	2
PR2	0.711	-1.000 (L)	0.509	800	25%	0	0	1	0	0	4
PR3	0.692	-1.000 (L)	0.509	1167	36.5%	0	0	4	0	0	7
PR4	0.738	-1.000 (L)	0.499	587	18.3%	0	0	3	0	0	5
PR5	0.703	-1.000 (L)	0.491	939	29.3%	0	0	3	0	0	6
CML	0.629	-0.878 (L)	0.546	1382	43.2%	0	2	10	0	1	11
XN2.5	0.658	-0.969 (L)	0.560	1074	33.6%	0	2	8	0	2	8
XN2.6	0.793	-0.965 (L)	0.596	1116	34.9%	0	2	7	0	2	7
DY.2	0.841	-0.997 (L)	0.594	2246	70.2%	0	1	13	0	1	15
DY.3	0.813	-1.000 (L)	0.571	2071	64.7%	0	0	11	0	0	13
JM1	0.672	-1.000 (L)	0.566	270	8.4%	0	0	2	0	0	2
PC5	0.733	-0.999 (L)	0.495	793	24.8%	0	1	6	0	1	6
Total	0.755	-	0.541	-	32.7%	0 (0%)	12 (9.3%)	117 (90.7%)	0 (0%)	11 (8.6%)	118 (91.4%)

Table 7: Comparison of the performance of ensemble voting approach using Logistic Regression (LR) and Random Forest (RF) in terms of the Area Under the ROC Curve (AUC).

Target Project	**Training Project(s)**	**Average AUC using ensemble voting (LR)**	**Average AUC (LR)**	**Average AUC using ensemble voting (RF)**	**Average AUC (RF)**
JDT	ML, PDE, PR2, PR4	60.3%	53.7%	60.3%	54.4%
ML	JDT, PDE, CML, PR2, PR4	60.6%	54.0%	59.7%	53.8%
PDE	JDT, ML, CML, PR2, PR4	66.1%	54.4%	65.7%	56.1%
CML	XN2.5, XN2.6	59.3%	59.1%	57.6%	56.7%
PR1	ML, CML	49.9%	49.9%	49.3%	50.0%
PR2	JDT, ML, PDE, PR4	51.4%	50.9%	51.5%	51.1%
PR3	JDT, ML, PDE, PR2, PR4, DY.3	51.4%	50.8%	51.0%	50.8%
PR4	CML, PR2	49.4%	49.7%	50.1%	50.0%
PR5	JDT, ML, PR2, PR4	48.9%	49.5%	48.9%	49.4%
XN2.5	CML, XN2.6, DY.3	61.4%	57.8%	64.9%	58.9%
XN2.6	CML, XN2.5	75.1%	64.8%	73.2%	63.2%
DY.2	JDT, ML, PDE, PR2, PR3, PR4, DY.3, JM1, PC5	73.3%	59.0%	77.8%	56.6%
DY.3	JDT, ML, PDE, PR2, PR3, PR4, DY.2, JM1, PC5	71.0%	58.0%	71.2%	55.0%
DG	CML, PR2, PR4, XN2.6, DY.3	68.6%	54.2%	56.5%	51.9%
SWT	CML	54.0%	54.0%	53.9%	53.9%
JM1	CML	56.6%	56.6%	55.0%	55.0%
PC5	ML, CML	47.1%	47.1%	47.0%	47.0%
Mean peformance		59.1%	54.3%	58.4%	53.8%

4.4 RQ4: How Feasible is HDP in terms of target prediction coverage?

HDP applicability is conditional on target prediction coverage — the percentage of target projects that can be predicted by HDP models. If no feasible HDP exists, due to missing matched metrics, it might be impossible to utilise heterogeneous predictors.

Table 8 shows how frequently a source dataset (rows of the table) can predict a target dataset (columns of the table) using the HDP algorithm. For example, Eclipse dataset can predict Apache dataset in 9 combinations, which is equal to 60% of all available combinations between these two sets (= $N_{\text{Eclipse}} \times N_{\text{Apache}} = 5 \times 3$). This pairwise prediction coverage reports many infeasible cases in our experiment.

Another important implication is the bold-faced values on the main diagonal of Table 8. Those values refer to the intragroup prediction feasibility of HDP. Since within each group of projects (e.g., Eclipse, Jira, etc.), almost all the metrics are the same, CPDP on the main diagonal of Table 8 is also feasible. However, HDP is unable to find similar metrics in some projects after applying

Table 8: Target Prediction Coverage of HDP.

Source	Eclipse	Proprietary	Apache	Jira	NASA	WPDP AUC	HDP AUC	HDP Target Coverage
Eclipse	**8 (40%)**	13 (52%)	9 (60%)	5 (50%)	0 (0%)	0.82	0.56	62.5%
Proprietary	1 (4%)	**7 (35%)**	4 (26.7%)	2 (20%)	0 (0%)	0.71	0.50	93.8%
Apache	10 (67%)	9 (60%)	**6 (100%)**	3 (50%)	3 (50%)	0.66	0.56	93.8%
Jira	4 (40%)	9 (90%)	6 (100%)	**2 (100%)**	4 (100%)	0.83	0.63	75.0%
NASA	4 (40%)	1 (10%)	2 (33%)	0 (0%)	**2 (100%)**	0.70	0.48	52.9%

feature selection. For example, Eclipse can predict itself in 8 out of 20 possible combinations (= $N_{\text{Eclipse}} \times (N_{\text{Eclipse}} - 1) = 5 \times 4$). The poor performance of metric selection/matching might lead to a lower self-coverage. We use the diagonal value and their available WPDP to validate the feasibility of HDP.

In Table 8, also, the median of AUC values (using either WPDP or HDP) for each group has been mentioned. WPDP statically performs better than HDP. The last column of Table 8 shows HDP target coverage of each source group. As the definition offers, a source group can cover a dataset of a different group if and only if at least a dataset in the source group can be used to build an HDP model and be tested on the target dataset. For example, NASA has coverage of 52.9% indicating that as the source group, its datasets (JM1 and PC5) can predict 9 out of 17 datasets, namely four projects in Eclipse group, one project in Proprietary, two projects in the Apache group, and two projects from NASA group. Proprietary and Apache by 93.8% coverage are the most reliable dataset groups on which we can build an HDP.

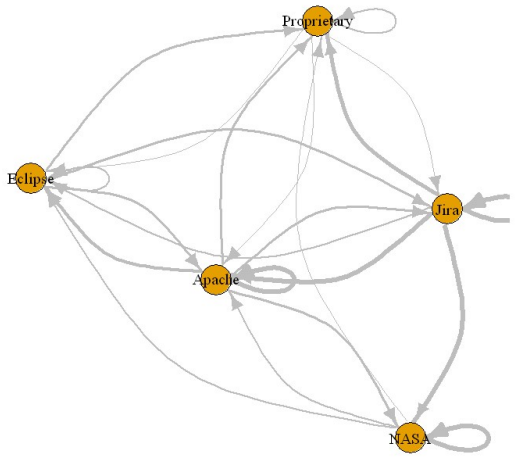


Figure 6: Pairwise target prediction coverage of HDP

Figure 6 illustrates the pairwise target prediction of group datasets. The thicker the arc is, the better the coverage will be. The outgoing arcs indicate the strength of target coverage, whereas the incoming arcs show how coverable a group is. For example, even though Jira can cover most of the datasets inside of the NASA group, NASA does not have such a mutual success rate. Accordingly, the upshot of the experiment indicates that there is a long path to build a real successful HDP.

5 THREATS TO VALIDITY

Both stratified cross-validation and normal cross-validation are not good representatives of the chronology of a dataset. In practice, the training set is always older than the testing set, whereas when we shuffle the dataset to implement CV, this order will be discarded. Therefore, other validation techniques that capture the time order in the dataset may yield different outcomes. Furthermore, we filter out irrelevant and unreliable projects based on predefined assumptions; however, there might exist some different criteria that influence the quality of the defect datasets. Thus, the more comprehensive project selection techniques may lead to better performance of the methods.

Here, we only use publicly available datasets while industry-related datasets may yield to a different conclusion; therefore, our partial replication study can reflect the result of the experiments where there is an overlap in our datasets. Such a difference does not affect the validity of the study, while it may overlook some conclusions in other types of projects.

The original work is implemented in Weka and Java, whereas we applied HDP using R (RWeka). The difference in the results may arise from the difference in their default variable options. Moreover, the correct threshold setting for HDP is a challenging issue that remains as future work. Finally, the granularity of the datasets differs. Considering this phenomenon might result in different outcomes. In future works, the datasets can be clustered based on their granularity, and we need to check whether HDP is sensitive to similarity in/difference between granularity of the source and target projects.

6 CONCLUSIONS ACROSS STUDIES

In this paper, we explored different approaches to build and validate HDP models. Defect prediction plays a crucial role in terms of efficiently allocating limited test resources. Traditional defect models use code metrics (e.g., Halstead and McCabe Metrics[4]) to classify a module as buggy or clean. Since there exist many potential defect datasets with heterogeneous metrics, we applied HDP on publicly available traditional defect datasets to learn whether transfer learning within traditional models is feasible.

We have conducted an empirical validation on HDP application and applied guidelines for reporting an experimental replication [1] to draw a cross-study conclusion between the original study and our replication. The findings of this partial, empirical replication study are as follows.

- The Within-Project performance of models is not a significant factor while applying Cross-Project Models.

- Even though applying a domain-agnostic similarity to predefine the best-performing model brings better performance than the median AUC of CPDP, WPDP still outperforms the models selected by domain-agnostic similarity (RQ1).
- Before using HDP, we need to adjust several hyper-parameters. The feature selection threshold (15%), cutoff threshold of KS-test *p-value* ($p = 0.05$), and the maximum percentage of acceptable infeasible cases (NaNs) (99%) raise the likelihood of unsuccessful HDP (RQ2). Hence, there is a need to optimise these techniques and balance the trade-off between the sensitivity of the algorithm and the selection of the parameters. The original study had not reported this issue.
- We proposed an ensemble voting approach in the HDP context using multiple source projects where there is a lack of common metrics. In terms of AUC, the performance of the revised HDP has increased by 4.8 percent (RQ3) compared to the original study.
- HDP Target Coverage was not promising. Besides, the performance of WPDP is statistically more significant than that of HDP (RQ2 and RQ4). These two issues indicate the need for further analyses to improve and generalise the heterogeneous prediction model. This comparison had not been performed in the original work.
- HDP cannot transfer the knowledge learned from a source dataset due to the substantial difference between the distribution of the metrics in a target and source project. In future research, employing exploratory analyses and conducting preprocessing techniques are highly recommended before applying HDP approaches.

SUPPORTING INFORMATION

To make the study reproducible, the datasets, codes, and outputs are publicly available at github.com/HadiJahanshahi/Replication-HDP.

REFERENCES

[1] J. Carver. 2010. Towards reporting guidelines for experimental replications: A proposal. In *1st international workshop on replication in empirical software engineering*. Citeseer, 2–5.

[2] G. Chandrashekar and F. Sahin. 2014. A survey on feature selection methods. *Computers & Electrical Engineering* 40, 1 (2014), 16 – 28. 40th-year commemorative issue.

[3] N. Cliff. 1993. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological bulletin* 114, 3 (1993), 494.

[4] B. Curtis, S. B. Sheppard, P. Milliman, M. A. Borst, and T. Love. 1979. Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics. *IEEE Transactions on Software Engineering* SE-5, 2 (Mar 1979), 96–104.

[5] M. D'Ambros, M. Lanza, and R. Robbes. 2010. An extensive comparison of bug prediction approaches. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. 31–41.

[6] M. D'Ambros, M. Lanza, and R. Robbes. 2012. Evaluating defect prediction approaches: a benchmark and an extensive comparison. *Empirical Software Engineering* 17, 4 (01 Aug 2012), 531–577.

[7] K. Gao, T. Khoshgoftaar, H. Wang, and N. Seliya. 2011. Choosing software metrics for defect prediction: an investigation on feature selection techniques. *Software: Practice and Experience* 41, 5 (2011), 579–606.

[8] L. Gong, S. Jiang, Q. Yu, and L. Jiang. 2019. Unsupervised deep domain adaptation for heterogeneous defect prediction. *IEICE TRANSACTIONS on Information and Systems* 102, 3 (2019), 537–549.

[9] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson. 2011. The misuse of the NASA metrics data program data sets for automated software defect prediction. In *15th Annual Conference on Evaluation Assessment in Software Engineering (EASE 2011)*. 96–103.

[10] M. Jureczko and L. Madeyski. 2010. Towards Identifying Software Project Clusters with Regard to Defect Prediction. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering (PROMISE '10)*. ACM, New York, NY, USA, Article 9, 10 pages.

[11] Y. Kamei, T. Fukushima, S. McIntosh, K. Yamashita, N. Ubayashi, and A. E. Hassan. 2016. Studying just-in-time defect prediction using cross-project models. *Empirical Software Engineering* 21, 5 (01 Oct 2016), 2072–2106.

[12] S. Kim, H. Zhang, R. Wu, and L. Gong. 2011. Dealing with noise in defect prediction. In *2011 33rd International Conference on Software Engineering (ICSE)*. 481–490.

[13] D. Krstajic, L. J. Buturovic, D. E. Leahy, and S. Thomas. 2014. Cross-validation pitfalls when selecting and assessing regression and classification models. *Journal of Cheminformatics* 6, 1 (29 Mar 2014), 10.

[14] Z. Li, X.-Y. Jing, F. Wu, X. Zhu, B. Xu, and S. Ying. 2018. Cost-sensitive transfer kernel canonical correlation analysis for heterogeneous defect prediction. *Automated Software Engineering* 25, 2 (01 Jun 2018), 201–245.

[15] Z. Li, X.-Y. Jing, X. Zhu, H. Zhang, B. Xu, and S. Ying. 2019. Heterogeneous defect prediction with two-stage ensemble learning. *Automated Software Engineering* 26, 3 (2019), 599–651.

[16] R. Lincke, J. Lundberg, and W. Löwe. 2008. Comparing Software Metrics Tools. In *Proceedings of the 2008 International Symposium on Software Testing and Analysis (ISSTA '08)*. ACM, New York, NY, USA, 131–142.

[17] J. Nam, W. Fu, S. Kim, T. Menzies, and L. Tan. 2018. Heterogeneous Defect Prediction. *IEEE Transactions on Software Engineering* 44, 9 (Sep 2018), 874–896.

[18] S. J. Pan and Q. Yang. 2010. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering* 22, 10 (Oct 2010), 1345–1359.

[19] F. Peters and T. Menzies. 2012. Privacy and utility for defect prediction: Experiments with MORPH. In *2012 34th International Conference on Software Engineering (ICSE)*. 189–199.

[20] F. Peters, T. Menzies, and A. Marcus. 2013. Better Cross Company Defect Prediction. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR '13)*. IEEE Press, Piscataway, NJ, USA, 409–418.

[21] D. Rodriguez, I. Herraiz, and R. Harrison. 2012. On software engineering repositories and their open problems. In *2012 First International Workshop on Realizing AI Synergies in Software Engineering (RAISE)*. 52–56.

[22] M. Shepperd, Q. Song, Z. Sun, and C. Mair. 2013. Data Quality: Some Comments on the NASA Software Defect Datasets. *IEEE Transactions on Software Engineering* 39, 9 (Sep 2013), 1208–1215.

[23] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto. 2017. An Empirical Comparison of Model Validation Techniques for Defect Prediction Models. *IEEE Transactions on Software Engineering* 43, 1 (Jan 2017), 1–18.

[24] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto. 2018. The Impact of Automated Parameter Optimization on Defect Prediction Models. *IEEE Transactions on Software Engineering* (2018), 1–1.

[25] H. Tong, B. Liu, and S. Wang. 2019. Kernel Spectral Embedding Transfer Ensemble for Heterogeneous Defect Prediction. *IEEE Transactions on Software Engineering* (2019), 1–1.

[26] A. Tosun and A. Bener. 2009. Reducing False Alarms in Software Defect Prediction by Decision Threshold Optimization. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM '09)*. IEEE Computer Society, Washington, DC, USA, 477–480.

[27] Burak Turhan, Tim Menzies, Ayşe B. Bener, and Justin Di Stefano. 2009. On the relative value of cross-company and within-company data for defect prediction. *Empirical Software Engineering* 14, 5 (01 Oct 2009), 540–578.

[28] Frank W. 1945. Individual Comparisons by Ranking Methods. *Biometrics Bulletin* 1, 6 (1945), 80–83.

[29] H. Wang, T. M. Khoshgoftaar, and A. Napolitano. 2010. A Comparative Study of Ensemble Feature Selection Techniques for Software Defect Prediction. In *2010 Ninth International Conference on Machine Learning and Applications*. 135–140.

[30] K. Weiss, T. M. Khoshgoftaar, and D. Wang. 2016. A survey of transfer learning. *Journal of Big Data* 3, 1 (28 May 2016), 9.

[31] R. Wu, H. Zhang, S. Kim, and S. Cheung. 2011. ReLink: Recovering Links Between Bugs and Changes. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE '11)*. ACM, New York, NY, USA, 15–25.

[32] S. Yatish, J. Jiarpakdee, P. Thongtanunam, and C. Tantithamthavorn. 2019. Mining Software Defects: Should We Consider Affected Releases?. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. 654–665.

[33] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy. 2009. Cross-project Defect Prediction: A Large Scale Experiment on Data vs. Domain vs. Process. In *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC/FSE '09)*. ACM, New York, NY, USA, 91–100.

Identifying External Cross-References Using Natural Language Processing

Elham Rahmani
Dept. of Computer Science
University of Western Ontario
London, Ontario, Canada
erahman3@uwo.ca

Nazim H. Madhavji
Dept. of Computer Science
University of Western Ontario
London, Ontario, Canada
madhavji@gmail.com

Ibtehal Noorwali
Dept. of Computer Science
University of Western Ontario
London, Ontario, Canada
inoorwal@uwo.ca

ABSTRACT

Software engineers build systems that need to be compliant with relevant regulatory requirements. Project contract contains cross-references to these regulatory requirements that usually exist in numerous and voluminous external documents. Identifying cross-references in such documents is enormously time consuming, costly, and error-prone in software projects. We use Natural Language Processing, Pattern Recognition and Web Scraping techniques to automatically extract external cross-references from a contract and web resources, and summarize them for the stakeholders. The novelty in this work is an algorithm based on: (i) semantic cues for identifying cross-references, (ii) grammatical structures for supporting various combinations of word roles in a sentence, (iii) APA standards for validating cross-references, and (iv) access to web resources. To operationalize this approach, we have created a tool that parses the given project contract and creates a summary of cross-references (currently limited to two levels of indirection) for use by stakeholders for project management, requirements elicitation, and testing. From a case study using an industrial-scale project contract, the performance of the tool suggests a precision of 99%, recall of 87%, and F-measure of 0.92.

CCS CONCEPTS

•Computing methodologies •Artificial intelligence •Natural language processing •Information extraction

KEYWORDS

Regulatory compliance, Regulatory requirements, Cross-references, Natural Language Processing, Pattern Recognition, Web Scraping

ACM Reference format:

CASCON'20, Nov 10-13, 2020, Toronto, Canada

1 Introduction

Software systems are required to comply with applicable regulations and standards [1] such as HIPPA (USA) [7] (health) and Sarbanes-Oxley Act [6] (financial). The applicable regulations and standards permeate the functional and nonfunctional (i.e., quality) requirements of both legacy and new systems.

In order to ensure that a system's requirements are compliant with relevant regulations and standards, analysts often need to identify and follow cross-references in project documents (e.g., a project contract). Such a cross-reference can be from one part of a document to another part of the same document (e.g., Division 4, paragraph 35) or to another part of a third-party document (e.g., IXXX Std 1016 paragraph 3)[3].

To our knowledge, previous work has focused on extracting cross-references internal to the same document and not on cross-references external to the primary document. These two situations are quite different.

For dealing with internal cross-references, all the elements of the document (e.g., headers, footers, titles, chapters, sections, sub-sections, and paragraphs) may refer to one another as necessary [4]. In contrast, for dealing with cross-references to external documents, one needs to, first, access the target document, which can be one of many according to the complexity of the primary document or project. Once the relevant external document is accessed, finding the target text or relevant information can be quite time consuming given that the structure of all relevant documents is not uniform. Moreover, once the target piece of information is identified, it may contain yet more external cross-references. The bigger picture is thus that of external documents forming a network through cross-references. Identifying cross-references in this context is thus arduous, time consuming, and error-prone [3].

For automating the extraction of cross-references internal to a document, researchers have used different techniques such as Part Of Speech Tagging (POST) techniques [19], MaxEnt statistical classification model [2], Support Vector Machine techniques [5], designing UML class diagram, transforming non-markup text to

XML markup text, etc. The type of documents processed are legal texts [4, 8, 9]. Some have classified cross-references based on semantic intent [9]. Section 2 gives more details about such related work.

The key elements of our approach are:

1. Analyzing the format of external cross-references in the primary document.
2. Analyzing phrases (called reporting phrases) that surround cross-references.
3. Creating a list of grammatical structures for supporting word-ambiguities that exist in different sentences.
4. Implementing a tool for identifying internal and external cross-references including from the world wide web.
5. Summarizing the identified cross-references for use by stakeholders.

The novelty of this work lies not only in the collective set of steps described above but also in the use of specific techniques used in individual steps. Section 3.2 describes the foundations for the proposed solution described in Section 4 which also describes a new algorithm at the core of the proposed solution. Using a large sized industry project contract document, the accuracy and precision of the algorithm are 99%, recall is 86%, and the F-measure is 92.

The summary output produced by the solution can save a significant amount of time and effort while avoiding or minimizing errors and frustration in ploughing through regulatory and standards documents during system development (specifically, eliciting regulatory requirements, ensuring design and code compliance, creating test cases, and resource estimation).

We conducted an empirical study on a "contract" for a systems engineering project. Typically, in-house software projects in industry do not have a formal contract as the developing teams are all internal to the same company. However, many projects are outsourced where a formal contract is signed by both the parties. Such projects are called "contractual projects". Our study is in this context.

The rest of the paper is organized as follows: Section 2 describes related work. Section 3 gives analyses of problem and solution-approach. Section 4 describes the proposed solution. Section 5 depicts the underlying algorithm. Section 6 compares our results with related work. Section 7 discusses limitations of the algorithm. Section 8 concludes the paper and describes future work.

2 Related Work

As mentioned earlier for dealing with the internal cross-references all the document's elements, including headers, footers, titles, chapters, sections, sub-sections, and paragraphs, may refer to each other. Therefore, any automated solution approach for detecting internal references needs to determine the essence of such entities. For example, the automated approach should identify the position of the referred "paragraph" or the referred "section" in the current document. In contrast, since external references refer to the documents existing out of the current document, they do not consist of static and certain terms including "paragraph", "section", "subsection", etc. Therefore, for identifying external references we are facing various unpredicted combinations of alphabets, numbers, special characters and words (see Section 3.2.1), which current solutions do not address. In this section, we describe three kinds of related works that focus on internal cross-references: (i) Automated reference resolution in legal texts [8], (ii) An Automated Framework for Detection and Resolution of Cross-References in Legal Texts [4] , and (iii) Automated Classification of Legal Cross-References Based on Semantic Intent [9].

Tran et al. [8] identify internal references in the law corpus. They split references by sequence labelling and Part Of Speech Tagging (POST) techniques [19] in order to determine where each word is located in a given string. Also, they define specific notations for tagging: the beginning of a reference, the inner part of a reference, the end of a reference, and all the elements outside the reference. Further, they define two classes of references, where Class 1 are those references that refer to an entire document; Class 2 for a fragment of a document. They also used two robust classifiers which have demonstrated strong performance in many classification tasks especially in statistical NLP. In the first classifier, the classification is performed with a statistical approach, built around the maximum entropy principle. In the second classifier, they used support vector machines (SVMs); a statistical machine learning technique to determine the class of each reference. Then they attempt to identify the positions of articles, paragraphs and items that are referred to by cross-references.

Adedjouma, et al. [4] implemented an automated approach for identifying internal references in legal documents. They manually built a UML class diagram, where classes represent the structural elements (e.g., articles, clauses, and paragraphs, etc.) of a legal text. These classes are linked via aggregation associations representing the hierarchical containment relationships between the elements. They further include in the schema the multiplicity constraints that need to be satisfied for the legal text to be structurally sound. Algorithmically, they generate markup scripts and execute regular expressions for marking the heads of sections. As for dealing with the internal cross-references all the document's elements, including headers, footers, titles, chapters, sections, sub-sections, and paragraphs, may refer to each other, thus, it is important to determine the essence of such entities and specify where an entity begins and ends. The annotations produced over a non-markup legal text by the regular expressions can be easily turned into a markup format, (e.g., XML). They then attempt to automatically detect explicit and implicit Cross Reference Expressions (CREs) in a given legal text. Based on the CRE categorization they define specific terms and patterns and then interpret the detected CREs into a set of individual cross-

references. For example, "article, articles, art, paragraph" are the predefined patterns used for explicit CREs and "above, below, preceding, following" are the predefined patterns used for implicit CREs.

Some of the phrases have similar meanings and intention but are semantically different than other types of phrases. Based on the semantic intention of phrases, Sannier, et al. [9] categorized phrases into eleven *semantic intent* types which were originally proposed by J. C. Maxwell in [1]. They described an automated approach for classifying cross-references by using a taxonomy of eleven semantic intent types including: Compliance, Constraint, Definition, Delegation, Exception, Refinement, General Amendment, Amendment by Addition, Amendment by Deletion, Amendment by Resignation, and Amendment by Replacement. Their main idea for such classification is identifying cross-references based on the appearance of phrases before or after cross-references.

The described related works [8, 4, 9] exemplify the type of research conducted on cross-references. Their primary focus is identifying cross-references internal to a given document (i.e., legal text). In contrast, in the world of contractual projects, the cross-references from a contract references third party documents (such as regulatory code and standards), which add another level of complexity to the document. Moreover, such third-party documents are structured in undefined or non-uniform ways, making the structure of these "external" cross-references complicated as is their automated identification. To our knowledge, there is no solution that deals with external cross-references, which is the focus of our work, described hereon.

3 Problem and Solution-approach Analysis

To better understand the problem, we provide an overview of a contractual document and its cross-references contained therein from a case study. This is followed by a description of our solution-approach.

3.1 Problem Analysis

Traditionally, the method used in industry for identifying cross-references is to manually identify regulatory requirements scattered in undefined ways in the contract [3]. In turn, a regulatory requirement may refer to an external document such as regulatory code or a standard. An externally referenced document may refer to yet other documents which, in turn, may repeatedly do so [3]. Thus, without automated support, identifying cross-references is arduous and error-prone and can easily result in systems that are non-compliant.

In this paper, we describe an investigation involving an industrial-scale project-contract of the size of approximately 700 pages containing approximately 10,300 paragraphs of text that have large number of external cross-references. Such contracts are quite typical in large systems engineering projects where system compliance to regulatory requirements is critical. Failing to satisfy compliance can lead to penalties or other forms of punishment to supplier organizations. The proposed approach (Section 4) should be generalizable to similar projects in industry.

3.2 Foundational Concepts

In this section, we describe the foundational concepts used in the design of the solution. In particular, we discuss four key concepts: Cross-References, Reporting Phrases, Cross-Reference Validation Conditions, and a high-level analysis of the NLP techniques applicable for phrase processing.

3.2.1 Cross-References

A Cross Reference Expression (CRE) (e.g.,"Submit shop drawings, diagrams, plans and details associated with demolition and removals work in accordance with Section 01340 SHOP DRAWINGS, and IXXX Std 1016 - Software Quality Assurance Plan") is a natural language phrase [4] which embodies one internal cross-reference: "Section 01340 SHOP DRAWINGS" by virtue of the fact that a document section number is cited and no external document is cited. The cross-reference "IXXX Std 1016 - Software Quality Assurance Plan" is an external reference.

An external cross-reference is a combination of letters (e.g., IXXX Std 1016 - Software Quality Assurance Plan) containing one or more sequence of words, acronyms, alphanumeric and characters such as: "/, -, _, ., (,)". In order to algorithmically recognize such cross-references, we decomposed its structure into meta-categories: "proper singular noun" (e.g., IXXX or Std) and number (e.g., 1016). Our algorithm uses such meta-categories (when parsing the contract document) since we are facing a problem that cross-references can be composed of recognizable (e.g., "Software") or unmeaningful (e.g., "IXXX") tokens. Also, each reference component (e.g., "Software", "Quality", "Assurance", "Plan") has a unique identity in the structure of an English sentence (e.g., the role of "Software" is a noun in the CRE). Thus, it is important to identify each component of a CRE in the identification process.

3.2.2 Reporting Phrases

Upon analysis of the case study contractual document, we noted that cross-references are predominantly associated with specific kinds of key-phrases in the contract (e.g., in accordance with, indicated in, specified in). We observed from published English literature [11, 13, 14, 15, 16] that the majority of such key-phrases used in our case study lies in the group of "Reporting Phrases". A *Reporting Phrase* is a phrase that is used for referring to the key details that uniquely identify a source of information [10]. Table 1 shows some of the reporting phrases from diverse literature (and their frequency count) prior to an external cross-reference in our case study contract. We, thus, use reporting phrases to identify cross-references in the contract. A key advantage of using Reporting Phrases from the established literature, as opposed to of creating our own scheme, is the generalizability of this idea across other contractual documents.

Table 1: Example reporting phrases identified from established sources [11, 13, 14, 15, 16]

Reporting phrases	Frequency	Reporting phrases	Frequency
in accordance with	241	based on	12
conforms to	65	proposed by	16
specified in	32	determined by	10
approved by	16	suggested by	14
indicated in	26	required by	12
recommended by	13	according to	10

Based on the position/status of reporting phrases in a CRE we categorized CREs into three groups for ease of processing: (i) Direct Cue (DC) – the reporting phrase appears prior to the reference (83%) (e.g., "Designed and certified for 85 dBA maximum noise level when measured in accordance with IXXX No.85"); (ii) In-Direct Cue (IDC) – a part of the reference appears before the reporting phrase and the rest of the reference appears after the reference (1.8%) (e.g., "Cooper E90 loading in accordance with AREAA"); and (iii) No Cue (NC) – there is no key-phrase associated with the reference (13.96%) (e.g., "DSPO 2110.050"). In this paper, we focus on DCs due to their high frequency in documents.

3.2.3 Cross-Reference Validation Conditions

Our cross-reference detection algorithm consists of conditions that validate the components of a cross-reference in the contract document. We defined validating conditions based on the American Psychological Association (APA) properties of in-text citations. APA in-text citations standards [12] are used for referring to, summarizing, paraphrasing or quoting from another source [18]. Thus, in our solution strategy, once a reference is identified, its components should be presented in one or more of the following APA in-text citation formats to be accepted as valid components of a cross-reference:

1. Acronym with all capital letters (e.g., *In accordance with **IXXX** No. 85*)
2. Sequence of words where each word starts with a capital letter (e.g., *in accordance with **National Building Code***)
3. Digits (e.g., *in accordance with IXXX Std No. **85***)
4. Characters such as *"/, -, _, ., (,)"* (e.g., *given in NAC/ASC A23.1/A23.2-M*)
5. A CRE may contain more than one reference separated by:
   i. And (e.g., *in accordance with the Manufacturer's Instructions **and** NAC/ASC A23.1/A23.2-M*)
   ii. Or (e.g., *shown in the Contract Documents **or** as determined by the GO Wheel*)
   iii. And Or (e.g., *in accordance with the NAC / ASC S16.1-M **and or** National Building Code*)
   iv. Comma (e.g., *in accordance with IXXX Std 1058, Software Project Management Plans*)

We observed that 8 out of a total number of 802 external references (i.e., 1%) in our case study contract do not meet APA properties. An example of this is:

*Prior to placing fresh concrete apply epoxy bonding agent **in accordance with** manufacturer's instructions or a neat cement wash consisting of one (1) part latex bonding agent mixed with two (2) parts cement and in accordance with manufacturer's instructions.*

In this example, "manufacturer's instructions" is an external reference that does not start in capital letters and, hence, fail to meet APA properties #1 or #2. Currently, such cases are not in the scope of our design solution.

3.2.4 Analysis of NLP techniques applicable for phrase processing

We observe from the analyses provided in the previous subsections that in order to identify cross-references, we need to deal with various types of phrases, words, alphabets, numbers and special characters see Section 3.2.1). Part of Speech Tagging (POST) [19] and Named Entity Recognition (NER) [19] are two different types of sequence labeling Techniques in NLP that can be applied on different components of a sentence. Part of speech tagging aims to identify which grammatical group a word belongs to, e.g., whether it is a noun, adjective, verb, or adverb and so forth, based on its context. The technique identifies relationships within the sentence and gives a corresponding tag to each word in the sentence.

Furthermore, regular expressions provide the possibility to process symbols and characters in a string or pattern to be searched for within a longer piece of text. Thus, they can be used for identifying sequence of grammar roles in a string.

On the other hand, Named Entity Recognition attempts to discover whether or not a word is a named entity (e.g., persons, locations, organizations, and time expressions etc.). This problem can be broken down into detection of names followed by classification of name into the corresponding categories. Most often once a word is recognized by NER, it may be recognized as a noun by POST. So, POST is more global, since it can determine the relationships between the first and the last word of a sentence.

For dealing with words and phrases in natural language, Word Lexical Disambiguation (Semantic or Syntactic) [19] is a commonly used approach. Semantic ambiguity occurs when a word, phrase or sentence of a context, has more than one interpretation. Word Sense Disambiguation (WSD) is defined as the ability to determine which meaning of word is activated by the use of word in a particular context. On the other hand, syntactic ambiguity is a situation where a sentence may be interpreted in more than one way due to ambiguous sentence structure.

Therefore, considering the above, in our problem, POST can be the choice NLP technique for figuring out the essence of each

token in a CRE in our specific context. This is because, as discussed in the problem analysis, we are not only faced with recognizing the names in a sentence; we are also facing various formats and identities of words. For dealing with these complexities, NER cannot provide us sufficient power to identify the identity of all the tokens. On the other hand, individual words that may be interpreted differently and causes ambiguity do not constitute a problem in our case. Thus, the techniques for word lexical disambiguation are not deemed an appropriate solution for our problem.

4 Proposed Solution

In this section, we describe the elements of the proposed cross-reference detection method (including an algorithm) consisting of the following key steps: (i) Create a whitelist and a pattern list; (ii) Check for reference patterns; (iii) Identify References, (iv) Find External Resources, and (v) Visualize References.

4.1 Create a Whitelist and Patterns List

4.1.1 Creating a whitelist of Reporting Phrases

With reference to Table 1, the purpose of "Reporting Phrases" is to help identify references [11, 13, 14, 15, 16]. Thus, we created a "whitelist" of these terms. Our aim is to recognize that a cross-reference (e.g., CAN/CSA A23.1/A23.2-M) exists in a given regulatory requirement (e.g., "Treat and finish exposed formed surfaces in accordance with CAN/CSA A23.1/A23.2-M").

4.1.2 Creating HasLeaf_Pattern list

From the analysis of numerous cross-reference expressions (CRE) in the contract document, we noted the existence of specific patterns in stating the references such as "*in accordance with CAN/CSA A23.1/A23.2-M*." We, thus, identified these patterns to create a list of patterns. By using a pattern recognition technique (RegExp Parser), it would then be possible to identify the matching patterns in the contract document.

To define patterns, we needed to understand the linguistic structure in the English grammar and specify the role of each word of a sentence. By using Part of Speech Tagging (POST) [19], we can tag the word types (e.g., **VB** for verbs, **NN** for nouns, etc.). With this, we created a list of grammatical patterns that we called "HasLeaf_Pattern". For example, the pattern "**{<VB><IN><DT><NNP>+}**" created by parsing some text denotes a sequence of words that are: verb, proposition, determiner and pronoun noun. Such patterns are also helpful in specifying the start and end points of a cross-reference pattern. For example: "in accordance with CAN/CSA A23.1/A23.2-M" is the start and end point of a cross-refence pattern which should be automatically specify from "Treat and finish exposed formed surfaces in accordance with CAN/CSA A23.1/A23.2-M." The main features of the "HasLeaf_Pattern" list consist of the following items:

i. One pattern can be matched with numerous references
ii. Different tenses/forms of reporting phrases are considered
iii. Supporting different propositions for different key-phrases
iv. Supporting key-phrases that do not have propositions
v. Supporting different propositions for one reporting phrase
vi. Supporting different determiners which might/might not be present after the reporting phrases and before the reference part

4.2 Check for Reference Pattern

The contract document is decomposed into individual paragraphs so that each paragraph is checked against the whitelist items (see Section 3.2.2 and 4.1.1 for the creation of the whitelist). If the paragraph contains whitelist items, it means that the paragraph might contain references. Following this, tokenization and POST are applied on that paragraph to specify the grammatical role of each paragraph tokens. Figure 1 depicts these steps which, together with Figure 2 and Section 4.3, illustrate how our proposed algorithm detects cross-references.

Paragraph before tokenization: *Conductor shall be soft or annealed copper,* ***in accordance with CBC B3-95****, stranded* ***in accordance with CBC B8-95****.*

After tokenization: [*'Conductor', 'shall', 'be', 'soft', 'or', 'annealed', 'copper', ',', 'in', 'accordance', 'with', 'CBC', 'B3-95', ',', 'stranded', 'in', 'accordance', 'with', 'CBC', 'B8-95', '.'*]

POST result over tokens: [('Conductor', 'NNP'), ('shall', 'MD'), ('be', 'VB'), ('soft', 'JJ'), ('or', 'CC'), ('annealed', 'VBN'), ('copper', 'NN'), (',', ','), ('in', 'IN'), ('accordance', 'NN'), ('with', 'IN'), ('CBC', 'NNP'), ('B3-95', 'NNP'), (',', ','), ('stranded', 'VBD'), ('in', 'IN'), ('accordance', 'NN'), ('with', 'IN'), ('CBC', 'NNP'), ('B8-95', 'NNP'), ('.', '.')]

Figure 1: Tagging paragraph tokens

Following the application of POST, the tagged tokens of each paragraph is processed by a regular expression parser, RegexpParser from NLTK [17] in search for patterns from the "HasLeaf_Pattern" list. If one or more patterns are identified in the paragraph, the "HasLeaf_Pattern" label is assigned to each pattern. Figure 2 shows the two identified cross-references from the sample paragraph of Figure 1. Each reference pattern ("Reporting phrase" and associated cross-reference) is shown as a leaf.

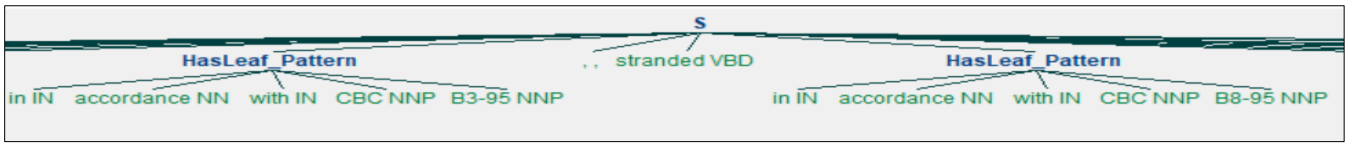


Figure 2: Tagging paragraph tokens

4.3 Identify Reference

Having obtained "HasLeafPatterns" as described in section 4.1.2, the whitelist item (i.e., reporting phrase) should be removed from each leaf because it is no longer needed (or rather all we need from the leaf is the cross-reference portion). For ensuring that the residue portion is a reference, it is checked against the standard properties of APA in-text citations (see Section 3.2.3). "Has_APA_Properties" is a module that we designed for checking the value of each identified leaf with the standard properties of APA in-text citations that are proposed by American Psychological Association.

The reason for considering this step as the step of reference validation is because, in some cases, the identified reference contains extra tokens that might be identified as one of the main components of the reference. For instance, the word "*the*" has a <DT> (Determiner) grammatical role in "*comply with **the** CP-100 SCM-S-0930-01*" and we have defined this grammatical role in the patterns of the proposed "*HasLeaf_Patterns*" list ({<VB><IN><**DT**><NNP>+}). Once the identified reference has been determined to contain the token "the", the reference is then transmitted to the "*Has_APA-Properties*" module, which then removes the token "the" from the rest of the reference components because "the" does not satisfy any of the APA properties.

Another example that demonstrates the functionality of the "*Has_APA-Properties*" module is that if a CRE contains the "*CBC-100 CND-S-0930-01*" reference, the pattern {<NNP>+} is matched with this reference. "NNP" refers to any word which has the role of "noun" or "proper singular noun". Thus, in this example, both the "*CBC-100*" and "*CND-S-0930-01*" components are labeled as "NNP" by POST. Therefore, <NNP>+ is matched with the above-mentioned reference. The quantifier (+) after the <NNP> means supporting unlimited number of sequential words which have an "NNP" grammatical role in the sentence. An important point to make is that the pattern {<NNP>+} can also be matched with the "*Software Quality Assurance Plan*" reference because all the components of this reference are also tagged as NNP by POST. That means that one {<NNP>+} pattern can support all references that have a sequence of NNP tokens. This is a strength of our "HasLeaf_Pattern" list, which allows us to detect numerous references that follow the same pattern by using only one pattern.

However, the "HasLeaf_Pattern" list may cause a problem in some cases. Particularly, when the parser considers the next <NNP> tokens as the reference component when, in fact, they are not reference components. For example, suppose that there is a regular word written in small letters (e.g., "conductor") and have the NNP role after this reference "*CBC-100 CND-S-0930-01*". In such a situation, the pattern identifier (RegExpress) will not detect that such a word ("conductor") is not a reference component because this word is another NNP and can be matched with the <NNP>+ pattern. Thus, the "*Has_APA-Properties*" module is helpful in dealing with such situations because all the components of the identified reference must satisfy at least one of the properties of APA standards. Therefore, words that are not written with capital letters identified as extra tokens and would be removed from the identified reference. Therefore, the "*Has_APA-Properties*" module operates as a filter for dismissing extra reference components and helps to choose the standard identified components.

4.4 Find External Resources

Having identified the cross-references in the given set of paragraphs from the project contract, the proposed algorithm considers each identified reference as a keyword. If no local sources are found for a given keyword (denoted by the node "Is there any local resource for the identified reference?" in Figure 3), a web scraping technique [15] invokes the Google search engine API (denoted by the node "Receiving reference by Google search API" in Figure 3), using the *Request* Python library, to find and open an external resource for the intended reference.

We choose the first recommended URL which may be directed to a pdf or an HTML file. If it is directed to a pdf file, we add it to our local database and convert it to unstructured data. Otherwise, by using the *BeautifulSoup* Python library the HTML page context is parsed from which we extract all the HTML open-close tags (e.g., <a>, <p>, <title>, etc.) including the intervening text. This is denoted by the node "Extracting HTML context" in the algorithm depicted in Figure 3. Thus, in this step, we have again structured textual data that may contain cross-references.

For example, with reference to Table 2, (see the column "Reference Level 2") our algorithm identified the following two references (among others) from third party documents: "ASTM A653/A653M SS Grade 80 (550)" and "ICA T1.1/T1.1R" – by first identifying reference "CSA S136-M" in the contract itself. We can then repeat: steps in Section 4.2 (denoted by the node "Splitting the context into paragraphs" in Figure 3); and Section 4.3, for finding any further references in the text contained in the two "Reference Level 2" documents mentioned above. For example, once the algorithm opens the resource "Reference Level 1" "CSA S136-M", it then looks for "reporting phrases" (see

Table 1) -- in this document -- that are followed by any matching pattern with the patterns defined in "HasLeaf_Patterns". For instance, in the following paragraph (in document CSA S136-M) "conforming to" is tagged as a reporting phrase and it then looks for a matching pattern for "ASTM A653/A653M SS Grade 80 (550)" in "HasLeaf_Patterns" list:

"Steels **conforming to ASTM A653/A653M SS Grade 80 (550)**, that do not meet the minimum 10 percent elongation requirement in Section A2.3.1."

4.5 Visualize References

This step is concerned with visually representing all the identified references which have been gathered in the previous steps. Once the references of each contractual requirement are identified, we generate a table using *Textable* Python library to depict the identified references of the contract. Once the tool finds a local or external resource related to the intended reference, the tool reapplies the reference identification step to that resource. The newly identified references are listed in the second column of the table.

In Table 2, the first column consists of the contractual requirement ID. The second column shows the first level references identified for the requirement. The third column represents all the indirect references identified through the documents at Level 1. In principle, the cross-reference identification can be extended to more levels of indirect references. However, we restrict the scope of our work in this study to two levels of references. It is important to note that all the listed references in the "Reference level 2" column are identified in the resources of the first reference from the "Reference Level 1" column. Pinpointing specific parts (e.g., paragraph, table, figure, etc.) embedded within the identified cross-references at "Reference Level 2" (or in further indirect documents) is not in the scope of this paper.

Table 2: Two levels of indirection of identified references starting from the contractual requirement.

Contractual Requirement Num	Reference Level 1	Reference Level 2
CR 2.4.1	1- CSA S136-M	1- ASTM A653/A653M SS Grade 80 (550)
	2- CRL/AMQ G164	2- A1008/A1008M SS Grade 80 (550)
	3- FBC 41-GP-34M Type G	3- A792/A792M Grade 80 (550)
	4- McFfoy Foundry Co. Ltd. MH332	
CR 4.1.1	1-Document 00500 SPECIAL CONDITONS	1- ICA T1.1/T1.1R
	2-CAN/ACA-A6/A362 Portland Type	2- Contract Code 7104

4.6 Flowchart Algorithm

Figure 3 depicts the flowchart of the proposed algorithm. It consists of five main steps that are described in Sections 4.1 to 4.5.

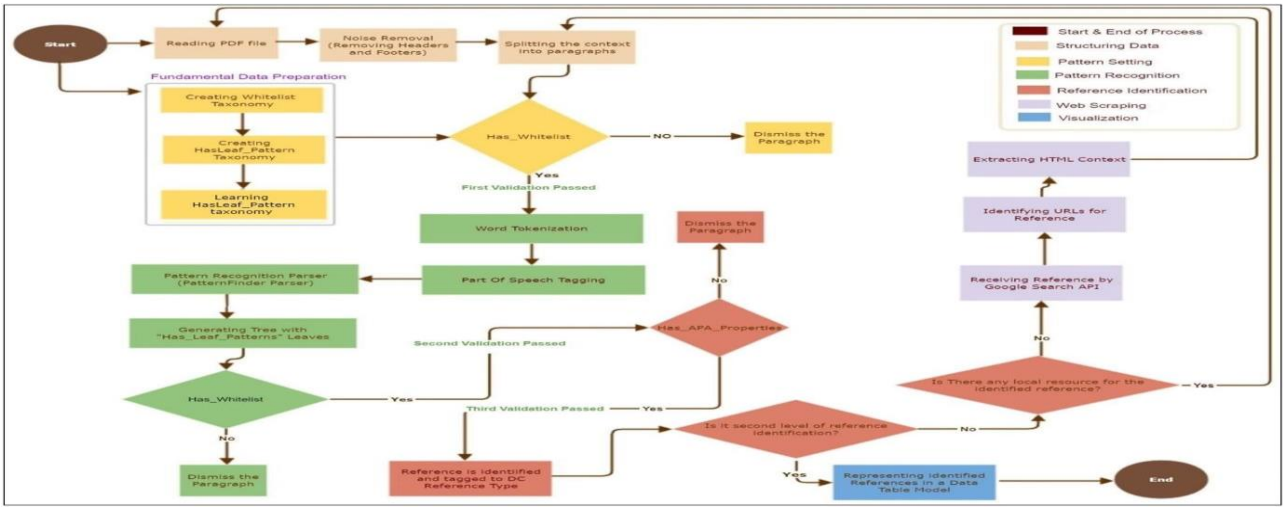


Figure 3: Flowchart of algorithm for Extracting External Cross-references (DC Reference)

4.7 Performance of the Algorithm

To test our algorithm's performance, we ran the algorithm on the case study contract document as described in Section 3.1. In this work, a word or sequence of words with all capital letters or a word or sequence of words starting with capital letters are considered an observation. For example: "Products", "Contract Documents", "RSS", "USRC" are some examples of observations. Since our dataset (contractual document) consists of raw textual data, and positive and negative observations are not automatically labelled, we manually labelled the sample data and tested the algorithm on 3,405 observations (approximately 20% of the total number of observations in the contractual document). The results of the algorithm's performance are as follows: True Positive: 258; True Negative: 3,104; False Positive: 2; False Negative: 41; Accuracy: 99%; Recall: 86%; Precision: 99%; and F-measure: 0.92.

5 Comparison with Related Work

Our approach (Section 4.6) for identifying external references differs from the discussed related works (Section 2) in a variety of ways. Table 3 provides a summary of the comparison.

Table 3: Summary of comparison with related work

No.	Comparison criterion	Tran, et al., 2014] [8]	(Adedjouma *et al.*, 2014) [4]	(Sannier *et al.*, 2016) [9]	Proposed approach
1	Text-Type	Law document	Law document	Law documents	Project contract
2	Type of cross-references	Internal Cross-References	Internal Cross-References	Internal Cross-References	External Cross-references
3	Phrase-Types	----	----	Classifying phrases into the 11 groups	Using a huge diversity of reporting phrases
4	Source of Phrases	----	----	Two legal documents	Established English literature (see Table 1)
5	Determining references boundaries	Tagging different parts of the references	Creating patterns based on explicit and implicit CREs	----	Creating "Has_Leaf_Pattern" list (see Section 4.1)
6	Validation of Cross-references	Non-Existent	Non-Existent	Non-Existent	The identified references must satisfy APA properties (see Section 3.2.3)
7	Access to third party resources	Non-Existent	Non-Existent	Non-Existent	Using web scraping techniques (see Section 4.4)

With Reference to Table 3:

1. **Text-Type**: All related studies dealt with law documents; whereas, our approach deals with a systems project contract.
2. **Type of Cross-References:** This criterion is a key differentiator. Tran et al. [8] and Adedjouma et al. [4] focused solely on extracting cross-references *internal* to a given document and not *external* ones referring to external documents.
3. **Phrase-Types:** As described in Section 2, Sannier et al. [9] proposed a taxonomy of 11 phrase-categorizations for identifying cross-references. In contrast, the proposed algorithm creates a huge variety of "Reporting Phrases" (see Table 1) from diverse sources.
4. **Source of Phrases:** In Sannier et al. [9], the authors identified their phrases from two legislative documents. In contrast, the reporting phrases are identified from established English literature resources [11, 13, 14, 15, 16]. This gives a wide latitude to the proposed algorithm to fit in different problem contexts
5. **Determining Reference Boundaries:** In Tran et al. [8], the starting and ending points of a reference is determined by defining different notations for different parts of the

reference. Adedjouma et al. [4] used predefined terms and patterns based on their CRE categorization (explicit and implicit). In this paper, by creating the *HasLeaf_Pattern* list (see Section 4.1.2), we use a new approach for defining the boundaries of references. Specifically, by using POS tagging, we generate a list containing various grammatical structures. Thus, if a CRE contains the reference "Electrical Code", the pattern {<JJ><NNP>} is matched with the reference. However, "JJ" also refers to any "adjective" role and "NNP" refers to any "noun" or "proper singular noun" role. Thus, by using this approach, the starting and ending points of a reference can be identified without predefined values.

6. **Validation of Cross-References:** As described in Section 4.3, sometimes the patterns in the "*Has_Leaf_Pattern*" list causes that RegExpress parser to detect extra tokens as the boundary of the reference. We define APA standards and design a cross-reference validation step for verifying the validity of identified tokens which is, to our knowledge, a novel approach in detecting reference components.
7. **Access to Third-Party Resources:** Our approach involves creating a chain of references, across a set of target documents for each contractual requirement, which we have implemented up to two levels of indirection thus far. In addition, we use web scrapers to access third party resources that are not locally available. This is a new approach as the surveyed related approaches were restricted to identifying *internal* references in *one* legal document and, thus, did not need to identify such indirections. Therefore, our approach shifts the focus to the network of *external* project documents, which current methods do not address.

In summary, the above comparison demonstrates the significantly different approach we have taken in identifying cross-references in contractual documents in software projects. More specifically, our approach contributes to, and complements, existing work by adding to the set of available tools for identifying cross-references. In other words, stakeholders can identify external references in addition to identifying internal references, which is anticipated to further facilitate the compliance process.

6 Limitations

With reference to Section 3.2.2, our analysis focused on identifying the Direct Cue (DC) category of references. One limitation of the current result is that it does not deal with the type of key-phrases that appear prior to the reference but are not in the group of reporting phrases (listed in Table 1). Examples of such key-phrases include "length to", "joints to", and "element to". This calls for a mechanism to identify cross-references which are not followed by reporting phrases.

Also, recall that in Section 3.2.2 we identified three types of references: DC (83%), IDC (1.8%) and NC (13.96%). The latter two are out of the scope of the current work.

In addition, as we discussed in Section 4.4, in our current work when the identified references are posted to google search API, it recommends 10 URLs from which, for simplicity, the current algorithm chooses the first one as the related resource. However, this may not be the best choice resource. Further analysis is needed to identify the most relevant URL based on the content from the recommended URLs. This can have a huge impact on the quality of the requirements elicited for the project.

With reference to Table 2, the proposed algorithm can identify cross-references in the case-study contract document (e.g., those listed in the column "Reference Level 1"). The algorithm also identified a number of cross-references from within the resources listed under "Reference Level 1" (e.g., those listed under "Reference Level 2"). In the current work, however, we do not explore this, indirect access in more detail but – from the preliminary experience -- are optimistic about the prospects with the proposed algorithm, perhaps with some adjustments, to make it generalizable across other contracts and regulatory documents.

Identifying references that are not APA-like, and are also not absolute cross-references (i.e., those that do not refer to a regulatory document) – e.g., "manufacturer's instructions" (see Section 3.2.3) -- are not in the scope of this paper.

7 Conclusion and Future Work

Project contracts or official documents in an organization often contain intra- and inter-document cross-references. Other researchers have attempted to automate the identification of intra-document references, e.g., within: legislative documents Tran et al. [8], Adedjouma, et al. [4], and Sannier, et al. [9]. Inter-document references can be significantly more challenging due to non-standard structure of the external documents (e.g., regulatory codes and standards), formats of the cross-references, and accessibility issues with third-party documents. There is also the issue of a chain of cross-references through multiple third-party documents. In large projects, manually identifying the cross-references is both time consuming and error-prone [3] and thus calls for automation.

This paper describes a new approach (see Sections 4 and 5) that identifies external references from a project contract and indirectly from external documents including the world wide web. This approach involves: (i) a list of semantic cues for identifying cross-references, (ii) a list of grammatical structures for supporting various combinations of word roles in a sentence, (iii) APA standards for validating cross-references, and (iv) third party access for unavailable resources. An overall algorithm (see Section 4.6) binds these aspects into a cohesive flow of steps implemented as a tool.

The tool produces a summary of cross-references (currently limited to two levels of indirection – see Table 2) that can be used for such tasks as: (i) time and resource estimation in project management, (ii) requirements elicitation, and (iii) creating test cases. In a case study (see Section 3.1) evaluation of this tool, using an industrial-scale project contract, the output suggests a precision of 99%, recall of 87%, and F-measure of 0.92.

From this advancement, we can conclude that extracting external cross-references from a network of documents is strongly feasible. Further work remains, of course as described in Section 6 (Limitations) as fodder for future work.,.

ACKNOWLEDGMENTS

We are most grateful to the reviewers for their constructive comments that helped us to improve this paper. Also, we are very grateful to NSERC, Canada, for supporting this research.

REFERENCES

[1] J. C. Maxwell, A. I. Antón, and P. Swire, "A legal cross-references taxonomy for identifying conflicting software requirements," Proc. 2011 IEEE 19th Int. Requir. Eng. Conf. RE 2011, pp. 197–206, 2011.

[2] B. Mack and B. Waske, "In-depth comparisons of MaxEnt, biased SVM and one- class SVM for one-class classification of remote sensing data," Remote Sens. Lett., vol. 8, no. 3, pp. 290–299, 2017.

[3] M. R. I. Nekvi and N. H. Madhavji, "Impediments to regulatory compliance of requirements in contractual systems engineering projects: A case study," ACM Trans. Manag. Inf. Syst., vol. 5, no. 3, 2014.

[4] M. Adedjouma, M. Sabetzadeh, and L. C. Briand, "Automated detection and resolution of legal cross references: Approach and a study of Luxembourg's legislation," in 2014 IEEE 22nd International Requirements Engineering Conference, RE 2014 - Proceedings, 2014, pp. 63–72.

[5] Scikit-learn developers (BSD License), "Support Vector Machines (SVM), 2020. Available: https://scikit-learn.org/stable/modules/svm.html.

[6] Sarbanes-Oxley Act (2017),"Sarbanes-Oxley Act of 2002,". Available at: https://pcaobus.org/About/History/Documents/PDFs/Sarbanes_Oxley_Act_of_2002.pdf.

[7] O. for C. R. (OCR) (2017), "Health Insurance Portability and Accountability Act of 1996 (HIPAA),". Available at: https://www.hhs.gov/hipaa/for-professionals/index.html.

[8] O. T. Tran, B. X. Ngo, M. Le Nguyen, and A. Shimazu, "Automated reference resolution in legal texts," Artif. Intell. Law, vol. 22, no. 1, pp. 29–60, 2014.

[9] L. B. Nicolas Sannier, Morayo Adedjouma, Mehrdad Sabetzadeh, "Automated Classification of Legal Cross References Based on Semantic Intent," in IEEE Software, 2016, vol. 28, no. 4, pp. 86–91.

[10] U. Sydney, "Introducing Quotations and Paraphrases," 2019. Available at: https://student.unsw.edu.au/introducing-quotations-and-paraphrases.

[11] EAP Foundaton (2019), "Reporting verbs,". Available at: https://www.eapfoundation.com/writing/references/reporting/

[12] PennState Library University (2020), "APA Quick Citation Guide", Available at: https://guides.libraries.psu.edu/apaquickguide/intext.

[13] AUSB, "APA Signal Phrases for Quotes/Paraphrases," 2017. Available at: https://www.antioch.edu/santa-barbara/wp-content/uploads/sites/4/2017/02/APA-Signal-Phrases-for-Quotes-and-Paraphrases.pdf.

[14] Toronto, U. of (2005), "Verbs For Citing Sources,". Available at: https://www.utsc.utoronto.ca/ccds/sites/utsc.utoronto.ca.ccds/files/5.pdf.

[15] Mitchell, R. (2018), Web Scraping with Python: Collecting More Data from the Modern Web. Available at: https://yanfei.site/docs/dpsa/references/PyWebScrapingBook.pdf.

[16] T. R. University, "Reporting Words / Phrases," p. 2007, 2007.

[17] Hardeniya, N. (2015) NLTK Essentials. Available at: https://books.google.ca/books?hl=en&lr=&id=NDlECgAAQBAJ&oi=fnd&pg=PP1&dq=NLTK+Library&ots=dCgq1d61TC&sig=MNqZmCjgLkYjkSH1_DfxzdWcoM4&redir_esc=y#v=onepage&q=NLTK&f=false.

[18] Association, A. P. (2019b), "APA Style". Available at: https://apastyle.apa.org/style-grammar-guidelines/citations/index.

[19] Steven Bird, Ewan Klein, and E. L. (2009) Natural Language Processing with Python. Available at: http://www.nltk.org/book/.

Time Series Sampling for Probabilistic Forecasting

Nicholas Prayogo
Ryerson University
Toronto, Ontario, Canada
nprayogo@ryerson.ca

Mucahit Cevik
Ryerson University
Toronto, Ontario, Canada
mcevik@ryerson.ca

Merve Bodur
University of Toronto
Toronto, Ontario, Canada
bodur@mie.utoronto.ca

ABSTRACT

Deep learning-based models for multiple time series probabilistic forecasting have gained significant attention in the recent literature. Given the abundance of data, many successful global and hybrid models that can learn complex patterns from multiple related time series have been developed. The main focus being on novel architecture development, little attention has been given to the investigation of input data for those models, making the impression that using more data is always better. In this paper, we strive to answer the following question: Is using more time series always better? Specifically, we investigate the usefulness of time series sampling in achieving better performance within lower run times on time series probabilistic forecasting. We evaluate the performance of two state-of-the-art models, namely DeepAR and DeepState, when using different numbers of time series that are selected based on a variety of distance-based similarity criteria for forecasting a single target time series. Through empirical evaluation on various common real-life datasets from the literature, we show that strategically selecting time series to train could help state-of-the-art models achieve improved forecasting accuracy while requiring a significantly less model training time.

KEYWORDS

Probabilistic forecasting, time series, DeepAR, DeepState

ACM Reference Format:
Nicholas Prayogo, Mucahit Cevik, and Merve Bodur. 2020. Time Series Sampling for Probabilistic Forecasting. In *Proceedings of 30th Annual International Conference on Computer Science and Software Engineering (CASCON'20).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Time series forecasting has applications across various industries, such as finance [15], energy [11, 13], weather [2], and business [28]. Accurate forecasts not only give businesses immediate insights on a fair assessment of their past actions, currently implemented policies and overall trends in the domain, but also play a crucial role in future policy making, being inputs to downstream decision-making problems. In regards to the latter, probabilistic forecasting became increasingly more popular since it allows sampling of future scenarios of uncertain parameters, which in turn helps making better decisions (via stochastic optimization, for instance).

Methods for time series forecasting have evolved from traditional ARIMA and exponential smoothing methods [6], which in general work well with structured and limited data, to neural network based methods such as recurrent neural networks (RNNs) and convolutional neural networks (CNNs), and attention-based models [16], which are capable of capturing more intrinsic and complex patterns and likely more useful in data-rich settings. Moreover, merge of such models, such as a hybrid ARIMA and neural network forecasting model proposed in [27] has also been studied and can result in well-performing models. For more details on neural network based forecasting methods and in particular on deep learning approaches, we refer the reader to the recent surveys [4] and [16], respectively.

For the purpose of our study, we look at some recent deep neural network-based probabilistic forecasting models, namely DeepAR [20] and DeepState [19]. In particular, these are global models capable of providing probabilistic forecasts of multiple time series *simultaneously*, which is not to be confused with multivariate forecasting where one model uses multiple time series to forecast a single time series. Leveraging information from multiple related time series, especially for the underlying global nonlinear patterns, these models have been shown to work particularly well across a variety of real-life datasets, such as electricity [25], traffic [10], and even most notably the recent M4 competition dataset [17]. However, the use of RNNs for these models limits training parallelism and thus often requires long run times to obtain a well-performing model, especially if trained with large amounts of data, where the size of the data is not only a function of the length of time series but also the number of time series itself. Perhaps in the interest of shorter training times, can these models be trained with a smaller subset of the data and still perform sufficiently well? This question has been theoretically and empirically investigated in terms of time series length. Differently, in this work, we propose to conduct a sensitivity analysis on the number of time series.

We study whether carefully selecting a subset of the time series as input can result in a similar or even better predictive performance for modern global deep neural models, especially when the objective is to forecast an individual target time series. We empirically analyze the impact of multiple distance-based time series sampling strategies, such as training with the most similar or dissimilar time series based on dynamic time warping, pointwise Euclidean norm in the frequency domain and Pearson correlation coefficient. We also consider clustering-based and data normalization-based approaches. The main contribution of this empirical study is to illustrate that a modern global probabilistic forecasting model can be trained quicker and achieve comparable or better performance with a carefully selected and significantly smaller subset of time series in the dataset.

CASCON'20, November 10-13 2020, Toronto, Canada

The rest of the paper is organized as follows. Section 2 reviews the most closely related literature, while Section 3 sets up the stage by formally defining the time series sampling problem. Section 4 walks through the components used to form this empirical study, both in terms of global models and sampling methods used, and outlines how we execute our experiments and which datasets were used. Lastly, Section 5 presents our results, followed by a conclusion and future research direction discussion in Section 6.

2 LITERATURE REVIEW

Some studies have indeed explored methods to forecast time series with limited data, mostly done by fusing capabilities of forecasting models. For instance, Choi et al. [8] combined the extreme learning machine and the grey model, Christodoulos et al. [9] combined ARIMA and diffusion models, and Tiwari and Adamowski [24] used an ensemble wavelet–bootstrap–artificial neural network, all to form a more robust time series forecasting model that performs well with limited data. However, this requires a significant amount of trial and error in terms of model combinations. Moreover, little study has been done on whether more advanced, neural probabilistic forecasting models [19, 20, 26], which are mostly known to require large amounts of data, can perform well or even better with less data.

In a recent study, Rabanser et al. [18] provided an empirical study on the impact of data input and output transformations, specifically data scaling and conversion of real-valued time series into categorical ones, on the predictive performance of multiple global neural probabilistic forecasting models. They showed that improved accuracy can be achieved via such data transformations.

As also mentioned in [18], the importance of careful data preprocessing for learning across multiple time series has been pointed out in the literature, despite not being explored in detail. The focus of the studies empirically analyzing this question, including the original papers proposing the model architectures, has been on the careful selection of features, covariates, time steps as well as data scaling. We instead focus on the selection of time series for training to obtain improved forecasts for a target time series, which to the best of our knowledge has not been studied before.

Hewamalage et al. [12] conducted a thorough empirical study on popular RNN-based forecasting models with the main motivation of showing their competitiveness against traditional methods. In the data preprocessing phase of their study, the authors mostly applied the ideas by Bandara et al. [3]. Although their study was limited to point forecasting in a univariate context, it shed some light into some of the aforementioned data pre-processing aspects that have been pointed out in the literature. Moreover, they observed that global models may not be suitable for the individual requirements of certain time series, and suggested that training single models on different subsets of time series as done by Bandara et al. [3] especially combined with some hybrid models as also performed by Sen et al. [21] can help. In that regard, our proposal of time series sampling can provide a simple remedy for such certain time series suffering from low accuracy in predictions of a global model trained with (a large number of) all available time series.

One of the most closely related work to ours is by Bandara et al. [3], who proposed to cluster a given set of time series and build a global model per cluster, in order to benefit more from local structures. More specifically, they obtained clusters using the unusual time series detection idea by Hyndman et al. [14], and trained an long short term memory (LSTM) network for each cluster. Their approach yielded consistent forecast accuracy improvements against learning a single global model on multiple commonly used datasets from the literature. Different than their feature-based clustering approach, we consider distance-based time series sampling as well as clustering in our study. Also, while their main motivation arose from the observation that training a global model using *disparate* time series may be detrimental to the *overall* forecasting accuracy, we propose sampling even for similar time series to significantly reduce training time for global model training without losing accuracy in obtained predictions.

The other closest work to ours is by Tadayon and Iwashita [23] who combined anomaly detection with clustering to improve predictive performance of an LSTM model. They proposed two feature extraction methods for time series data, which in turn outperform distance-based clustering methods in terms of speed, yielding similar forecast accuracy on their synthetically created dataset. They also analyzed the impact of adding more measurements to the training set, and observed that it does not necessarily improve the predictive performance. In our study, we analyze the same question in terms of addition of time series, rather than time steps.

3 PROBLEM DESCRIPTION

We define the necessary notation for our analysis as follows. For $a, b \in \mathbb{Z}_+$ with $b \geq a \geq 1$, let $[a, b] := \{a, a+1, \ldots, b\}$ denote the set of integers between a and b. Also, let $[a] := [1, a]$.

3.1 Probabilistic forecasting problem

Given a forecast horizon $H \in \mathbb{Z}_+$, and I length-T time series, described by tuples

$$\{z^i\}_{i\in[I]} := \left\{\left((y_t^i)_{t\in[T]}, (x_t^i)_{t\in[T+H]}\right)\right\}_{i\in[I]}$$

of observations (denoted by y^i) and input covariates (denoted by x^i), the probabilistic forecasting problem aims to model the joint conditional probability ($\mathbb{P}$) distribution of future observations represented by random variables $\{\boldsymbol{y}_t^i\}_{i\in[I],t\in[T+1,T+H]}$:

$$\mathbb{P}\left(\{\boldsymbol{y}_t^i\}_{i\in[I],t\in[T+1,T+H]} \mid \{z^i\}_{i\in[I]}\right). \tag{1}$$

That is, given the values observed at T consecutive time steps for each series together with their characteristics in the form of covariates until the end of the prediction horizon, it tries to give an idea about the likelihood of observations to be made in the subsequent H time steps.

In this work we focus on global neural models, which define some global parameters Θ for (1) that are learned using all the available time series simultaneously,

$$\Theta = g\left(\{z^i\}_{i\in[I]}\right),$$

which is then used to estimate the prescriptive probability distribution of individual time series as

$$\mathbb{P}_i := \mathbb{P}\left(\{\boldsymbol{y}_t^i\}_{t\in[T+1,T+H]} \mid z^i\right) \sim f_i(z^i, \Theta), \text{ for all } i \in [I]. \tag{2}$$

In other words, they construct separate (local) models of conditional probability distributions for individual time series, that are parametrized by a set of common (global) factors, which requires learning g and $f_1, \ldots, f_I$ functions whose forms are usually fixed based on the considered neural architecture with parameters to be learned.

The goal is to minimize the error in the approximations in (2) as

$$\min_{g,\{f_i\}_{i\in[I]},\Theta} \sum_{i\in[I]} \rho\left(\mathbb{P}_i, f_i\left(z^i, \Theta\right)\right) \tag{3a}$$

$$\text{subject to } \Theta = g\left(\{z^i\}_{i\in[I]}\right) \tag{3b}$$

where ρ denotes an error measure, which is usually considered to be a log-likelihood type of loss function.

3.2 Time series sampling problem

Suppose that the global model is fixed, i.e., the forms of g and f functions are fixed. Given a subset of target time series, indexed by $\mathcal{I}^{\text{target}} \subseteq [I]$, for which we care about obtaining accurate forecasts, or rather improved accuracy from the global model under certain computational effort considerations, we define the time series sampling problem as the following bilevel problem:

$$\min_{\mathcal{I}^{\text{sample}}\subseteq[I]} \quad \min_{g,\{f_i\}_{i\in\mathcal{I}^{\text{sample}}},\Theta^{\text{sample}}} \sum_{i\in\mathcal{I}^{\text{target}}} \rho\left(\mathbb{P}_i, f_i\left(z^i, \Theta^{\text{sample}}\right)\right) \tag{4a}$$

$$\text{subject to} \qquad \Theta^{\text{sample}} = g\left(\{z^i\}_{i\in\mathcal{I}^{\text{sample}}}\right) \tag{4b}$$

$$\texttt{TrainEffort}(\mathcal{I}^{\text{sample}}) \leq E \tag{4c}$$

where we aim to find a subset of time series, indexed by $\mathcal{I}^{\text{sample}} \subseteq [I]$, using which the global model can be trained respecting our capacity constraints (such as time and memory limits) given in (4c), and yield the lowest possible prediction error for our target time series. Note that the global model is now constructed using the time series included in the selected sample, as indicated in (4b).

In this work, as the first step to analyze the time series sampling problem, we limit our focus to $|\mathcal{I}^{\text{target}}| = 1$, and investigate alternative methods to construct $\mathcal{I}^{\text{sample}}$ to obtain good quality solutions to this bilevel problem.

4 METHODOLOGY

In this section, we first provide a short description of the forecasting models and then elaborate on time series sampling methods employed in our analysis. Then, we provide the data and experimental setting information.

4.1 Models

We experiment with two state-of-the-art multiple time series forecasting models, namely DeepAR and DeepState, whose details are briefly explained below.

4.1.1 DeepAR [20]. DeepAR is an RNN-based global time-series forecasting method; it uses an autoregressive recurrent network architecture of LSTM cells to model the joint conditional probability distribution, given in (1). It can learn complex patterns, such as seasonality, but as observed in the literature since it does so purely from the data, its performance usually improves when the training data size is increased. We question whether the same phenomenon occurs when the available number of time series increases. Another advantage of DeepAR is that it can handle large variations in time series scales thanks to its internal data normalization and velocity-based sampling features. In fact, in the next subsection, we propose a time series sampling idea based on this underlying normalization process of DeepAR.

4.1.2 DeepState [19]. Deep State Space Models (SSMs), DeepState in short, combines the capabilities of understanding structural time series of traditional SSMs like ARIMA, with deep recurrent neural networks' ability to capture inherent and complex temporal patterns. This results in a model that can be trained with little data and still perform well due to the structure imposed by the SSM and its ability to still extract latent features. It has been noted to be more suitable for datasets of related time series [19], where the *relation* is interpreted as the belonging to the same domain. This leads to the question of whether this global model can perform better with a carefully chosen subset of time series versus when using the entire dataset. In other words, we question whether there could be more to the relation for an improved predictive performance.

4.2 Sampling methods

In this study, we test a variety of sampling ideas, based on standard distance-based measures for time series [1] except a new one based on our observations on how DeepAR architecture works. (For the literature on time series clustering, we refer the readers to the survey paper by Aghabozorgi et al. [1].)

4.2.1 DTW. While L2-norm can be used to measure point-wise distance between time series, this measure can be misleading when lag exists between two time series. To overcome this issue, an algorithm such as dynamic time warping (DTW) [5] can be useful. DTW is indeed the most commonly used distance measure for time series. The idea is to find a nonlinear mapping of two series that minimizes the point-wise distance between the mapped series. More specifically, DTW uses a dynamic programming algorithm to determine an optimal warping path traversing a set of points, satisfying certain conditions (e.g. boundary, monotonicity and step size), where the objective is to minimize the L2-norm of the point vector describing the path. Having a quadratic complexity, DTW can be slow for long time series, although the run times can be possibly improved Silva and Batista [22].

4.2.2 Freq-L2. Given that any time series is essentially a sum of sinusoids of various frequencies, Fourier transform converts one from time domain to frequency domain. The frequency domain thus represents an amplitude-frequency graph that shows the magnitude of each frequency captured by the time series. This is useful in terms of capturing the seasonality of the time series data, and thus to compare the seasonality between two time series, where the standard L2-norm measure can be used. An example for this approach is provided in Figure 1. While the magnitudes of two time series in the time domain differ significantly, which would result in a high pointwise L2 distance, they follow a similar seasonality and thus are relatively close in the frequency domain.

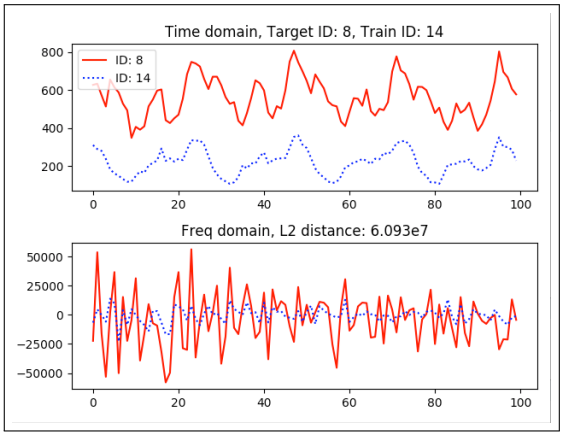


Figure 1: Calculating the distance between two time series in the frequency domain

4.2.3 Pearson Correlation Coefficient. Given two time series, Pearson Correlation Coefficient divides the product of the covariances of the time series over the product of their respective standard deviations, giving a value between -1 and 1, where these extreme values respectively mean that the series are perfectly negatively and positively correlated. For sampling, we choose to start with time series that have higher absolute correlations to the target time series.

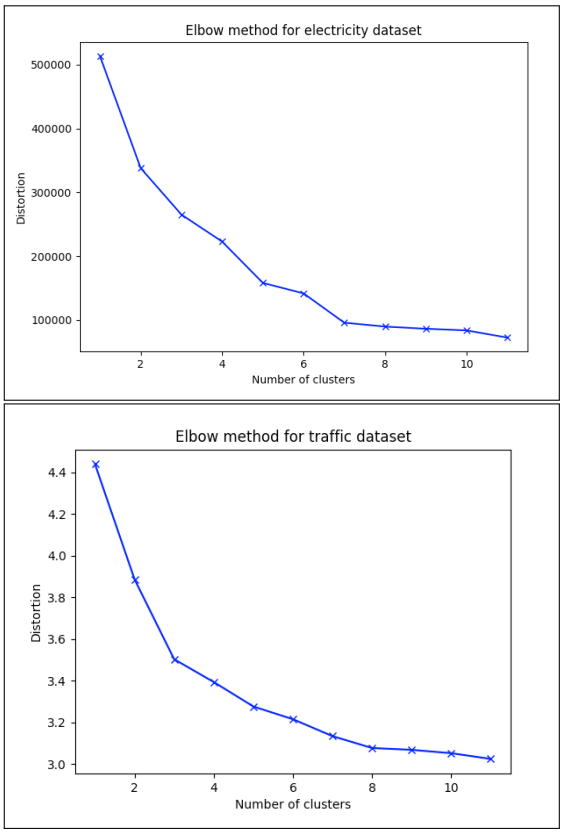


Figure 2: Elbow method to determine the optimal number of clusters

4.2.4 K-Means Clustering. K-means clustering is an unsupervised learning algorithm that identifies K centroids and assigns data points to the nearest centroid. By clustering time series, we train the model using only time series belonging to the same cluster as the target time series. To determine the optimal number of clusters, we use the elbow method as shown in Figure 2. As mentioned in Section 2, [3] employed time series clustering to build multiple global models, one per cluster, to improved predictive performance. While the authors in [3] obtained clusters based on some extracted feature vectors of the given time series, we experiment with distance-based clustering.

4.2.5 Normalizing Constant. For the case of time series, we define normalizing constant as the sum of all values of a time series. This definition is particularly useful for the training of DeepAR, since it is trained via weighted Monte Carlo simulation where the sorted normalizing constants are used to decide on sampling weights [20]. As such, time series with larger normalizing constants have a higher impact in the loss minimization process.

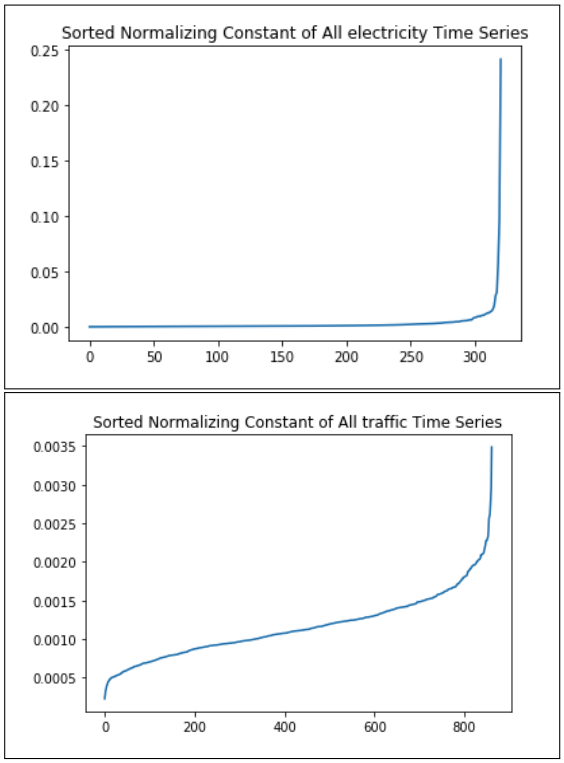


Figure 3: Area Coverage of Normalizing Constant

As Figure 3 illustrates, for our two datasets, we find that roughly 15% of the time series represents 75% of the normalizing constant area under the curve. This means that only this 15% is mostly sampled by the DeepAR model and affects the loss minimization the most during training. As such, we propose to use only this

percentile of the dataset to train the model, with the expectation that the model would perform similarly to the case where we train using the entire dataset.

4.3 Data

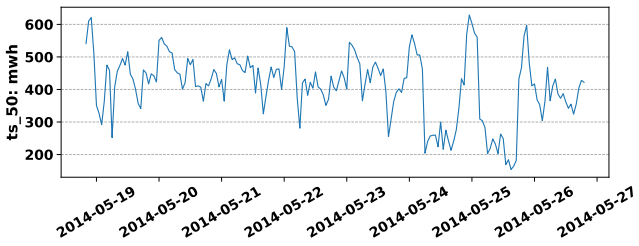
Our empirical study was conducted on electricity [25], traffic [10], and m4_hourly [17] datasets, whose properties are summarized in Table 1.

The electricity dataset consists of electricity consumption of 370 households for more than 1400 days during 4 years. The records for some clients were created after 2011, for which the consumption before their creation date is assumed to be zero. For the sake of a more qualified data, we eliminated the time series with high volume of missing values and/or zeros. After this elimination, we decided to use only 321 households' electricity consumption data for two years.

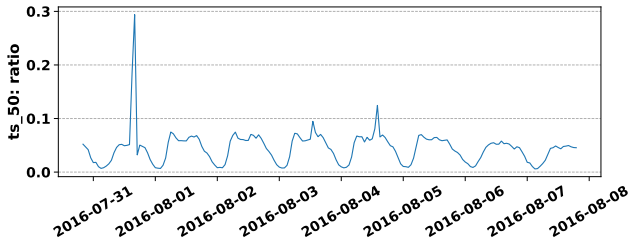
The traffic data is collected from over 39,000 detectors at 10 minute intervals. These sensors span the freeway system across all major metropolitan areas of the State of California, US. This dataset consists of 15 months worth of daily data, describing the occupancy rate, between 0 and 1, of different car lanes of San Francisco Bay Area freeways. Every single lane in this database is considered as one time series.

The m4_hourly dataset consists of 414 hourly time series, which is a subset of the M4 competition dataset. These time series were selected from a database called ForeDeCk compiled at the National Technical University of Athens (NTUA) which highlights a wide range of business forecasting applications including domain from natural resources and tourism to stocks and bonds [17]. This dataset is also scaled such that observations are neither negative nor less than 10, as well as ignores low-volume and intermittent time series, to prevent methodology, architecture, or error measure-related problems.

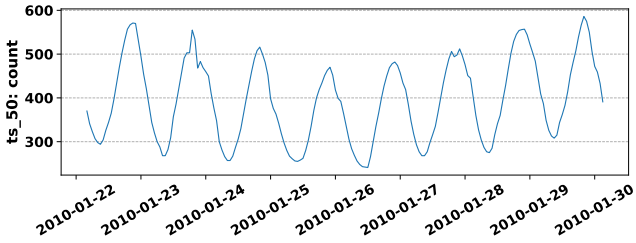
Figure 4 demonstrates a sample from the target time series of the electricity, traffic, and m4_hourly datasets, all of which shows a certain degree of seasonality.



(a) electricity target time series 50



(b) traffic target time series 50



(c) m4_hourly target time series 50 (data start dates are not known and are arbitrarily set to Jan 1, 2010)

Figure 4: Electricity, traffic, and m4_hourly data samples

4.4 Experimental setup

Our experiments were conducted with GluonTS, a Python library for probabilistic time series forecasting. Many hyperparamaters in GluonTS are set to certain default values, which are not necessarily ideal for the model. For that, we listed the parameters we used for training the models on Table 2. Also note that, to ensure that the model goes through the entire training set every epoch, given that T is the number of training time steps, C is the context length, N is the number of time series to be trained with, and B is the batch size, the number of batches per epoch is scaled as $(T/C) \times (N/B)$. We also observe that when trained with covariates, the model did not show significant improvements and in some occasions even performed worse than the version without covariates. Thus, since training with covariates also significantly increases runtime, for this empirical study, covariates were not used.

To compare model performance over the various distance and correlation measures, we define six different time series sampling methods: 1) "random", which chooses n time series at random for each run, 2) "dtw_min", which chooses n time series closest in DTW measure to the target time series, 3) "freq_min", which chooses n time series closest in Freq-L2 measure, 4) "dtw_max", n farthest in DTW measure, 5) "freq_max", n farthest in Freq-L2 measure, and 6) "pearson", n most correlated in Pearson's coefficient. Using these selection methods, we start off with evaluating the model's performance when trained with the target time series itself, then increment by n where $n = 10$ for electricity, $n = 25$ for traffic, and $n = 15$ for m4_hourly until the model is trained with the entire dataset. We used Numpy for all of these selection methods, with the addition of fastdtw Python library, specifically for DTW.

For the remaining methods, different experimental setups were used. For the normalizing constant method in particular, we find the number of time series that represents 75% of the area under the curve (AUC) of the sorted normalizing constant plot, and train the model using that number of time series. Then, when using K-means clustering, we cluster the time series into the optimal number of clusters obtained from the elbow method, then select the top 25% time series closest to the centroid of the cluster that the target time series belongs to.

Table 1: Dataset Properties

Dataset	Number of time series	Length	Domain	Granularity	Any missing data	Min	Max
electricity	321	21068	$\mathbb{R}_+$	Hourly	No	0	764000
traffic	862	14036	$\mathbb{R}_+$	Hourly	No	0	1
m4_hourly	414	854 (mean)	$\mathbb{R}_+$	Hourly	No	10	703008

Table 2: DeepAR and DeepState model parameters

Data	Epochs	Learning rate	Batch size	Context length	Prediction length	# of batches per epoch
electricity	40	0.001	64	168	24	Scaled
traffic	40	0.001	64	168	24	Scaled
m4_hourly	40	0.001	16	48	48	Scaled

4.5 Performance Evaluation

We evaluate the performances of the forecasting models with the following metrics:

- Normalized Deviation (ND)

$$ND = \frac{\sum_{i,t} |z_{i,t} - \hat{z}_{i,t}|}{\sum_{i,t} |z_{i,t}|}$$

- Normalized Root Mean Squared Error (NRMSE)

$$NRMSE = \frac{\sqrt{\frac{1}{N}\sum_{i,t}(z_{i,t} - \hat{z}_{i,t})^2}}{\frac{1}{N}\sum_{i,t} |z_{i,t}|}$$

- 90% Quantile Loss or p90 (where $\rho = 0.9$)

$$p90 = 2\frac{\sum_{i,t} P_\rho(z_{i,t}, \hat{z}_{i,t})}{\sum_{i,t} |z_{i,t}|}$$

$$P_\rho(z, \hat{z}) = \begin{cases} \rho(z - \hat{z}) & \text{if } z > \hat{z} \\ (1-\rho)(\hat{z} - z) & \text{otherwise} \end{cases}$$

where $z_{i,t}$ and $\hat{z}_{i,t}$ are the actual value and predicted median value respectively for item $i \in \mathcal{I}$ at time $t \in \mathcal{T}$, and $N = |\mathcal{I}| \times |\mathcal{T}|$. For each metric, the lower values point to a better model performance.

5 RESULTS

In this section, we present the results of our detailed numerical study along with relevant observations and conclusions. We first provide an overview of the models performance on the entire data set as well as a couple arbitrarily selected target time series. Next, we assess the impact of distance and correlation-based time series sampling strategies on predictive performance for individual target time series. Lastly, we focus on sampling strategies based on clustering and data normalization.

5.1 Performance of the forecasting models

Before evaluating our time series sampling approaches, we first compare the performance of DeepAR and DeepState when trained with the entire dataset, and evaluated on both the entire and single time series scenarios. Table 3 shows lower ND, NRMSE, and p90 values for DeepAR, indicating that DeepAR consistently outperforms DeepState across all three datasets and evaluation scenarios. Also, as expected, the methods can yield significantly different errors for different individual time series. We next analyze whether better predictions can be obtained for the two individual target time series (indexed by 8 and 50) when these global models are trained with smaller samples of time series.

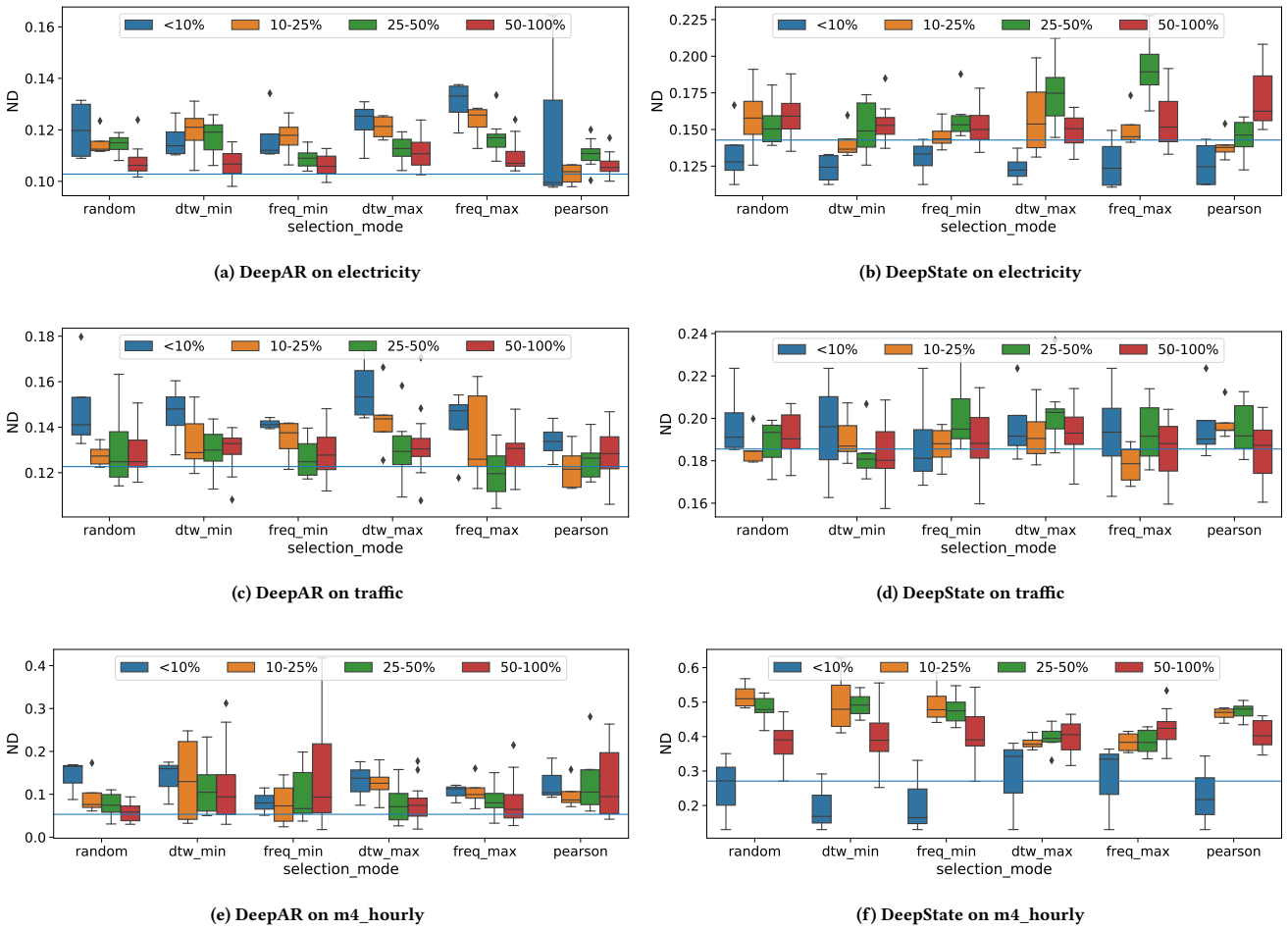
Table 3: Performance comparison with the models trained over the entire dataset

	Evaluation set	ND	NRMSE	p90
DeepAR	electricity	0.075	0.680	0.042
	electricity - 8	0.103	0.171	0.068
	electricity - 50	0.132	0.183	0.070
	traffic	0.134	0.412	0.095
	traffic - 8	0.123	0.241	0.070
	traffic - 50	0.076	0.188	0.061
	m4_hourly	0.073	0.427	0.054
	m4_hourly - 8	0.072	0.082	0.033
	m4_hourly - 50	0.071	0.071	0.032
DeepState	electricity	0.095	0.750	0.057
	electricity - 8	0.143	0.207	0.102
	electricity - 50	0.205	0.288	0.085
	traffic	0.241	0.526	0.230
	traffic - 8	0.186	0.323	0.195
	traffic - 50	0.173	0.262	0.203
	m4_hourly	0.377	2.917	0.038
	m4_hourly - 8	0.337	0.469	0.051
	m4_hourly - 50	0.339	0.475	0.039

5.2 Distance and correlation-based sampling

We provide box plots for the aforementioned six distance- and correlation-based sampling strategies in Figure 5 and Figure 6. For each strategy, four boxes are drawn corresponding to the error ranges obtained by using the labeled percentage of the total number available time series in the dataset. The (blue) straight line in these plots represent the ND value obtained by using all the time series, that are provided to the global models in the order of the "random" strategy, thus constitutes a baseline.

In these plots, we observe that, for DeepState, selecting relevant time series might enable the model to perform better than when



(a) DeepAR on electricity

(b) DeepState on electricity

(c) DeepAR on traffic

(d) DeepState on traffic

(e) DeepAR on m4_hourly

(f) DeepState on m4_hourly

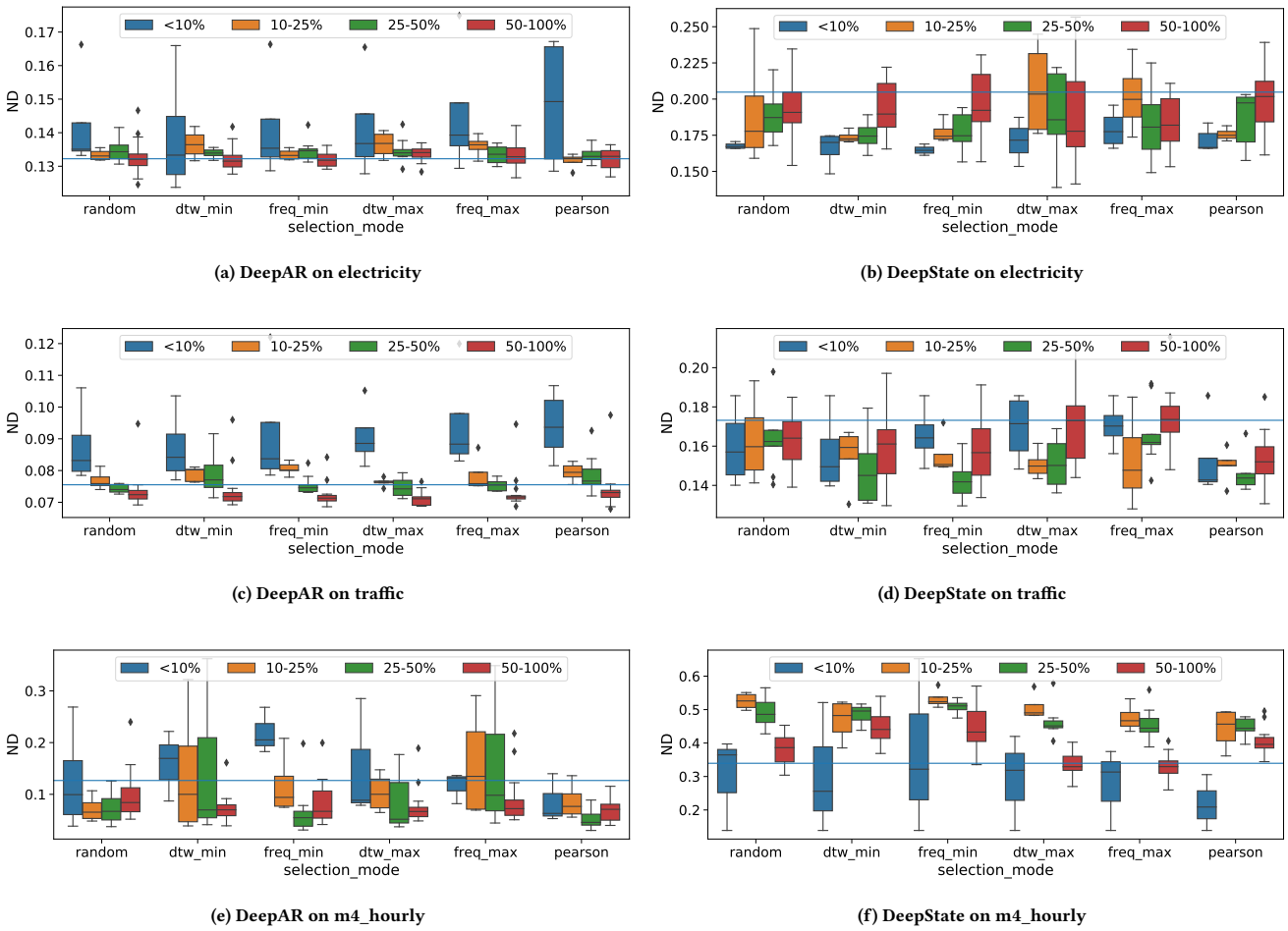
Figure 5: Distance- and correlation-based sampling results for target time series 8

using the entire dataset in certain cases e.g. for the electricity dataset. Also, the minimum distance-based strategies yield a much smaller variance for small sample cases, thus can be seen as better sampling options. On the other hand, for DeepAR, it seems like the inclusion of more time series leads to improved performance, while it is again possible to obtain smaller errors with a much smaller sample of time series. Although these plots show there is not necessarily a clear winner regarding which selection method works best for time series sampling, we can see that in most cases, training with less yet strategically-sampled data can give the model a similar, and even better, forecasting performance. These general findings actually align well with our expectations based on the observations from the literature that (i) DeepAR learns complex patterns purely from data, thus its performance usually improves when the training data size is increased, and (ii) DeepState is more suitable for datasets of related time series. Accordingly, the differences in our results can be attributed to the architecture differences in these global models. Lastly, we note that our conclusions for this experiment are quite similar for both target time series 8 and 50.

5.3 Other sampling strategies

Since we have seen that distance- and correlation-based sampling might allow DeepState to perform better with little and carefully selected data, we provide in Table 4 our results of using other sampling methods for DeepAR specifically, to see whether this architecture can benefit from other sampling methods. For comparison, we also show results when using dtw_min to sample 25% of the dataset (dtw_min_25). As shown here again, carefully selecting time series allows the model to almost emulate its performance when trained with the entire dataset while taking significantly less time to train.

Looking at the Normalizing Constant (NC) selection method for instance, when trained to forecast electricity data, even when using only 15% of the dataset representing those with highest normalizing constant, the DeepAR model is able to forecast time series 8 and 50 almost as well as when it uses the entire dataset (9.7% and 2% difference for ND). A similar observation can be made for traffic dataset. While it requires about 64% of the traffic dataset to represent 75% of the AUC, we can see that especially for target 50, the model even performs better with only 64% of the dataset. This method can



(a) DeepAR on electricity

(b) DeepState on electricity

(c) DeepAR on traffic

(d) DeepState on traffic

(e) DeepAR on m4_hourly

(f) DeepState on m4_hourly

Figure 6: Distance- and correlation-based sampling results for target time series 50

thus be used to significantly reduce training data especially when a small percentage of the dataset represents a large percentage of the AUC curve. A possible extension to this approach can be to find the optimal percentile given accuracy and run time constraints.

Similarly, by using only the top 25% closest to the target cluster centroid (kmeans_25), the model is able to perform almost as well while taking only roughly 6%, 1%, and 3% of the time to train for electricity, traffic, m4_hourly respectively. However, as we similarly observe in the previous section, there is no clear distinction on which sampling method is the best.

6 CONCLUSION AND FUTURE DIRECTIONS

In this paper, we present an empirical study to quantify the impact of adding more time series to the training set of a global deep neural probabilistic model on improving the predictive performance. We find that a careful selection of time series can leverage cross-series information better and in turn yield improved forecasts for a target time series, and save significant computational effort. Considering the fact that the performance of many modern global methods working with a large number of time series highly depend on data pre-processing, e.g., scalarization, and careful hyperparameter tuning, time series sampling can also help in reducing such efforts and help us obtain more robust methods.

In that regard, we see our study as the first step to investigate the impact of strategically sampling time series in global forecasting models. We conclude the paper by outlining some intriguing directions for future research as follows:

- Experiment with other methods such as Deep Factors [26] and Deep TCN [7]
- Compare with feature-based sampling
- Extend from a single target time series to a subset of target time series
- Combine time series sampling with input transformation
- Develop a dynamic/adaptive time series sampling method
- Extend to an interpretability setting to determine the importance of individual time series in global models
- Incorporate these ideas into architecture design

Table 4: Impact of time series sampling on DeepAR model performance

	target_id	mode	n_series	training_time (sec)	ND	NRMSE	p90
electricity	8	full	321	24,859.2	0.103	0.171	0.068
	8	dtw_min_25	80	5,436.8	0.123	0.187	0.067
	8	kmeans_25	63	1,491.1	0.117	0.182	0.070
	8	NC	51	1,867.5	0.113	0.180	0.066
	50	full	321	22,269.9	0.132	0.183	0.070
	50	dtw_min_25	80	5,732.4	0.135	0.181	0.071
	50	kmeans_25	63	1,529.5	0.131	0.180	0.073
	50	NC	51	1,837.2	0.134	0.183	0.076
traffic	8	full	862	38,887.6	0.123	0.241	0.070
	8	dtw_min_25	225	9,053.7	0.137	0.267	0.088
	8	kmeans_25	37	473.1	0.146	0.283	0.085
	8	NC	551	11,185.4	0.140	0.268	0.088
	50	full	862	34,896.6	0.076	0.188	0.061
	50	dtw_min_25	225	10,082.4	0.076	0.174	0.066
	50	kmeans_25	45	587.6	0.078	0.175	0.066
	50	NC	551	11,132.4	0.073	0.170	0.061
m4_hourly	8	full	414	2191.2	0.072	0.082	0.033
	8	dtw_min_25	105	229.6	0.052	0.066	0.019
	8	kmeans_25	3	6.4	0.065	0.081	0.039
	8	NC	31	67.9	0.092	0.101	0.036
	50	full	414	2190.0	0.071	0.089	0.032
	50	dtw_min_25	105	230.4	0.049	0.065	0.023
	50	kmeans_25	84	182.9	0.092	0.106	0.035
	50	NC	31	67.6	0.054	0.067	0.030

Acknowledgment. The authors would like to thank Juyoung Wang for valuable discussions throughout this work. The authors also would like to thank LG Sciencepark for providing support for this project.

REFERENCES

[1] Saeed Aghabozorgi, Ali Seyed Shirkhorshidi, and Teh Ying Wah. 2015. Time-series clustering–a decade review. *Information Systems* 53 (2015), 16–38.
[2] S Santhosh Baboo and I Kadar Shereef. 2010. An efficient weather forecasting system using artificial neural network. *International journal of environmental science and development* 1, 4 (2010), 321.
[3] Kasun Bandara, Christoph Bergmeir, and Slawek Smyl. 2020. Forecasting across time series databases using recurrent neural networks on groups of similar series: A clustering approach. *Expert Systems with Applications* 140 (2020), 112896.
[4] Konstantinos Benidis, Syama Sundar Rangapuram, Valentin Flunkert, Bernie Wang, Danielle Maddix, Caner Turkmen, Jan Gasthaus, Michael Bohlke-Schneider, David Salinas, Lorenzo Stella, et al. 2020. Neural forecasting: Introduction and literature overview. *arXiv preprint arXiv:2004.10240* (2020).
[5] Donald J Berndt and James Clifford. 1994. Using dynamic time warping to find patterns in time series.. In *KDD workshop*, Vol. 10. Seattle, WA, 359–370.
[6] George EP Box and Gwilym M Jenkins. 1968. Some recent advances in forecasting and control. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 17, 2 (1968), 91–109.
[7] Yitian Chen, Yanfei Kang, Yixiong Chen, and Zizhuo Wang. 2020. Probabilistic forecasting with temporal convolutional neural network. *Neurocomputing* (2020).
[8] Tsan-Ming Choi, Chi-Leung Hui, Na Liu, Sau-Fun Ng, and Yong Yu. 2014. Fast fashion sales forecasting with limited data and time. *Decision Support Systems* 59 (2014), 84–92.
[9] Charisios Christodoulos, Christos Michalakelis, and Dimitris Varoutas. 2010. Forecasting with limited data: Combining ARIMA and diffusion models. *Technological forecasting and social change* 77, 4 (2010), 558–565.
[10] M. Cuturi. 2015. UCI Machine Learning Repository. https://archive.ics.uci.edu/ml/datasets/PEMS-SF.
[11] Chirag Deb, Fan Zhang, Junjing Yang, Siew Eang Lee, and Kwok Wei Shah. 2017. A review on time series forecasting techniques for building energy consumption. *Renewable and Sustainable Energy Reviews* 74 (2017), 902–924.
[12] Hansika Hewamalage, Christoph Bergmeir, and Kasun Bandara. 2019. Recurrent neural networks for time series forecasting: Current status and future directions. *arXiv preprint arXiv:1909.00590* (2019).
[13] Che-Chiang Hsu and Chia-Yon Chen. 2003. Regional load forecasting in Taiwan—-applications of artificial neural networks. *Energy conversion and Management* 44, 12 (2003), 1941–1949.
[14] Rob J Hyndman, Earo Wang, and Nikolay Laptev. 2015. Large-scale unusual time series detection. In *2015 IEEE international conference on data mining workshop (ICDMW)*. IEEE, 1616–1619.
[15] Bjoern Krollner, Bruce J Vanstone, and Gavin R Finnie. 2010. Financial time series forecasting with machine learning techniques: a survey.. In *ESANN*.
[16] Bryan Lim and Stefan Zohren. 2020. Time Series Forecasting With Deep Learning: A Survey. *arXiv preprint arXiv:2004.13408* (2020).
[17] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. 2020. The M4 Competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting* 36, 1 (2020), 54–74.
[18] Stephan Rabanser, Tim Januschowski, Valentin Flunkert, David Salinas, and Jan Gasthaus. 2020. The Effectiveness of Discretization in Forecasting: An Empirical Study on Neural Time Series Models. *arXiv preprint arXiv:2005.10111* (2020).
[19] Syama Sundar Rangapuram, Matthias W Seeger, Jan Gasthaus, Lorenzo Stella, Yuyang Wang, and Tim Januschowski. 2018. Deep state space models for time series forecasting. In *Advances in neural information processing systems*. 7785–7794.
[20] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. 2019. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting* (2019).

[21] Rajat Sen, Hsiang-Fu Yu, and Inderjit S Dhillon. 2019. Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting. In *Advances in Neural Information Processing Systems*. 4837–4846.

[22] Diego F Silva and Gustavo EAPA Batista. 2016. Speeding up all-pairwise dynamic time warping matrix calculation. In *Proceedings of the 2016 SIAM International Conference on Data Mining*. SIAM, 837–845.

[23] Manie Tadayon and Yumi Iwashita. 2020. Comprehensive Analysis of Time Series Forecasting Using Neural Networks. *arXiv preprint arXiv:2001.09547* (2020).

[24] Mukesh K Tiwari and Jan F Adamowski. 2015. Medium-term urban water demand forecasting with limited data using an ensemble wavelet–bootstrap machine-learning approach. *Journal of Water Resources Planning and Management* 141, 2 (2015), 04014053.

[25] A. Trindade. 2015. UCI Machine Learning Repository. https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014.

[26] Yuyang Wang, Alex Smola, Danielle C Maddix, Jan Gasthaus, Dean Foster, and Tim Januschowski. 2019. Deep factors for forecasting. *arXiv preprint arXiv:1905.12417* (2019).

[27] G Peter Zhang. 2003. Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing* 50 (2003), 159–175.

[28] G Peter Zhang. 2004. *Neural networks in business forecasting*. IGI global.

Insights into WebAssembly: Compilation Performance and Shared Code Caching in Node.js

Tobias Nießen
Faculty of Computer Science
University of New Brunswick
tniessen@unb.ca

Michael Dawson
IBM Runtime Technologies
IBM Canada
Michael_Dawson@ca.ibm.com

Panos Patros
Software Engineering
University of Waikato
panos.patros@waikato.ac.nz

Kenneth B. Kent
Faculty of Computer Science
University of New Brunswick
ken@unb.ca

ABSTRACT

Alongside JavaScript, V8 and Node.js have become essential components of contemporary web and cloud applications. With the addition of WebAssembly to the web, developers finally have a fast platform for performance-critical code. However, this addition also introduces new challenges to client and server applications. New application architectures, such as serverless computing, require instantaneous performance without long startup times. In this paper, we investigate the performance of WebAssembly compilation in V8 and Node.js, and present the design and implementation of a multi-process shared code cache for Node.js applications. We demonstrate how such a cache can significantly increase application performance, and reduce application startup time, CPU usage, and memory footprint.

CCS CONCEPTS

• **Software and its engineering** → **Compilers**; **Software performance**; • **Computer systems organization** → *Cloud computing*.

KEYWORDS

WebAssembly, compiler, code cache, Node.js, V8, JavaScript

ACM Reference Format:
Tobias Nießen, Michael Dawson, Panos Patros, and Kenneth B. Kent. 2020. Insights into WebAssembly: Compilation Performance and Shared Code Caching in Node.js. In *Proceedings of 30th Annual International Conference on Computer Science and Software Engineering (CASCON'20).* IBM Corp., RIverton, NJ, USA, 10 pages.

1 INTRODUCTION

WebAssembly is a new hardware abstraction that aims to be faster than interpreted languages without sacrificing portability or security. Conceptually, WebAssembly is a virtual machine and binary-code format specification for a stack machine with separate, linearly addressable memory. However, unlike many virtual machines for high-level languages, the WebAssembly instruction set is closely related to actual instruction sets of modern processors, since the initial WebAssembly specification does not contain high-level concepts such as objects or garbage collection [8]. Because of the similarity of the WebAssembly instruction set to physical processor instruction sets, many existing "low-level" languages can already be compiled to WebAssembly, including C, C++, and Rust.

WebAssembly also features an interesting combination of security properties. By design, WebAssembly can only interact with its host environment through an application-specific interface. There is no built-in concept of system calls, but they can be implemented through explicitly imported functions. This allows the host to monitor and restrict all interaction between WebAssembly code and the host environment. Another important aspect is the concept of *linear memory*: Each WebAssembly instance can access memory through *linear memory*, a consecutive virtual address range that always begins at address zero. The host environment needs to translate virtual memory addresses into physical addresses on the host system, and ensure that virtual addresses do not exceed the allowed address range. On modern hardware, this can be implemented using the Memory Management Unit (MMU) and hardware memory protection features, leading to minimal overhead while allowing direct memory access to *linear memory* and preventing access to other memory segments of the host system [8]. Combined, these properties allow running the WebAssembly code both with full access to the real system, and in a completely isolated sandbox, without any changes to the code itself.

These properties make WebAssembly an attractive platform for performance-critical portable code, especially in web applications. However, WebAssembly makes no inherent assumptions about its host environment, and can be embedded in other contexts such as Node.js, a framework built on top of the JavaScript engine V8. Not only does Node.js share many technological aspects, such as the programming language JavaScript, with web applications, but its performance and portability goals align well with those of WebAssembly. Sandboxing, platform-independence, and high performance are especially relevant in cloud-based application backends, the primary domain of Node.js.

However, one hindrance remains: Because WebAssembly code is tailored towards a conceptual machine, it can either be interpreted on the host machine, or first be compiled into code for the actual host architecture. As we will see below, interpretation leads to

CASCON'20, November 10–13, 2020, Toronto, Canada

inadequate performance. At the same time, compilation of large WebAssembly modules can lead to considerable delays during the application's startup phase.

Node.js is often used in "serverless computing:" Instead of keeping a number of servers running at all times, the provider allocates resources dynamically with a minimal scope. For example, Function-as-a-Service (FAAS), sometimes referred to as "serverless functions," is a deployment model in which the provider only allocates enough resources for a single function to be executed for each request, and no internal state is preserved between requests [2]. In this scenario, it is crucial for the function to be executed quickly in order to produce the desired response to an incoming request with minimal latency, making it unfeasible to compile large WebAssembly modules on each request.

In this paper, we analyze the performance of V8's WebAssembly compilers. Sections 2 and 3 provide an overview of background and related work. We analyze the performance of code generated by V8 in Section 4. In Section 5, we investigate the performance of V8's WebAssembly compilers themselves, and the performance benefits of caching compiled code. Finally, in Section 6, we present and evaluate the design and implementation of a multi-process shared code cache for WebAssembly code in Node.js applications.

2 BACKGROUND

With the addition of WebAssembly to Node.js, there are three kinds of code that are supported: JavaScript, native addons, and WebAssembly. While JavaScript code and WebAssembly are interpreted and/or compiled by V8 (see Section 4), native addons behave like shared libraries and allow embedding "native" code, e.g., compiled C or C++ code. Unlike WebAssembly, native addons have direct access to the underlying system and its resources (file systems, network interfaces, etc.). While this might be a desired or even required feature for some use cases, it might be a security risk in others [7]. Additionally, native addons usually need to be compiled on the target platform, while WebAssembly is portable and agnostic of the system architecture. Most existing research around WebAssembly focuses on performance comparisons between JavaScript, WebAssembly, and native code.

Haas et al. described the motivation behind WebAssembly, its design goals, code execution and validation [8]. The authors used the PolyBench/C benchmark [23] to compare the performance of WebAssembly to that of native code and asm.js [12], a subset of JavaScript designed to be used as a compilation target for C code, which could then be executed by a JavaScript runtime. They found that WebAssembly was 33.7% faster on average than asm.js, and that the execution time of WebAssembly was less than 150% of the native execution time for 20 out of 24 benchmarks. It is important to note that V8 only implemented the TurboFan compiler [25] at that time, and did not use the Liftoff compiler [10].

Herrera et al. used the Ostrich benchmark suite [16] to compare the performance of native code, WebAssembly, and JavaScript. The benchmark performs numerical computations that are deemed relevant for scientific computations, such as machine learning. While they also found WebAssembly in Node.js to be slower than native code, WebAssembly consistently outperformed JavaScript in all tested web browsers and Node.js [13, 14].

Malle et al. conducted experiments comparing WebAssembly to JavaScript, asm.js, and native code in the context of artificial intelligence algorithms [18]. The results are in line with the results reported by Haas et al. [8] and Herrera et al. [13, 14], and again show that WebAssembly is faster than JavaScript, but slower than native code. The authors suggest that future additions to WebAssembly such as SIMD instructions will likely reduce the difference between WebAssembly and native code.

Hall et al. investigated WebAssembly as a platform for serverless applications. They came to the conclusion that the security properties of WebAssembly allow isolation similar to virtualization via containers, and that, while WebAssembly generally did not outperform native code, containers often took longer to start than the respective WebAssembly implementations [9].

Matsuo et al. suggested using WebAssembly in browser-based volunteer computing. They found WebAssembly outperformed JavaScript for long-running tasks, but the overhead of compiling and optimizing WebAssembly before being able to run it caused performance gains to disappear for tasks with short durations [19].

Jangda et al. exposed performance flaws of WebAssembly implementations in web browsers [15]. They found that, on average, WebAssembly code in the V8-based web browser Chrome is 55% slower than native code. While comparing code generated by V8 to native code generated by a C compiler, the authors observed that V8 produces more instructions. This leads to more CPU cycles required to execute the code and more cache misses in the processor's L1 instruction cache. Code generated by V8 also suffers from increased register pressure due to sub-optimal register allocations and the fact that V8 reserves some registers for memory management. Finally, the WebAssembly specification mandates certain safety checks at runtime, which also incur a performance cost. However, despite these problems, the authors also showed that WebAssembly was 54% faster than asm.js in the same browser.

The multitude of publications highlighting the performance benefits of WebAssembly over JavaScript has inspired efforts to simplify the integration of WebAssembly into existing JavaScript applications. For example, Reiser et al. proposed a cross-compilation method from JavaScript to WebAssembly that resulted in significant speedups of computationally intensive algorithms [24].

3 RELATED WORK

The idea of caching compiled code beyond single processes is not new, and has been implemented for other languages.

Bhattacharya et al. discussed improvements for the shared class cache (SCC) used by the J9 Java virtual machine. While its primary purpose is to reduce memory usage by sharing required class files, the SCC also contains compiled code, reducing application startup times significantly [3, 11].

Patros et al. invented a mechanism to reuse compilation results for Platform as a Service (PaaS) clouds via Dynamically Compiled Artifact Sharing (DCAS), with a focus on the Java SCC [6, 21].

Park et al. proposed a method to reuse code generated by an optimizing JavaScript just-in-time (JIT) compiler, allowing ahead-of-time (AOT) compilation based on previous compilations of the same code. Their benchmarks demonstrated significant speedups

in JavaScript application performance [20]. The authors also highlighted the need for such technologies due to the increasing code size of web applications.

Haas et al. discuss two ways to improve compilation and startup times for WebAssembly in web browsers [8]. According to their research, parallel compilation using multiple threads leads to compilation times that are 80% lower than those of single-threaded compilation. V8 already implements parallel compilation by assigning individual WebAssembly functions to separate compilation threads. The authors also suggest that developers cache compiled WebAssembly modules in browsers using client-side IndexedDB databases [1]. However, IndexedDB is not available in Node.js, and as of 2020, support for WebAssembly modules in IndexedDB databases has been removed from V8, making it impossible for developers to explicitly cache compiled WebAssembly modules. Instead, web browsers are encouraged to implement implicit code caching as part of streaming WebAssembly compilation [4], which is not available in Node.js.

4 COMPILATION AT RUNTIME

The compiler infrastructure within V8 has changed significantly in the last few years. Even for the relatively new WebAssembly language, V8 implements a complex combination of compilation procedures for WebAssembly. The basic components are a WebAssembly interpreter, the baseline compiler *Liftoff* [10], and the optimizing compiler *TurboFan*, that V8 also uses to compile JavaScript [25, 26].

Since JavaScript code itself does not contain static type information, it is difficult to compile it directly [20]. Due to this difficulty, V8 begins JavaScript execution using the *Ignition* interpreter, and only when the interpreter has identified "hot" code sections, the TurboFan compiler is used to optimize and compile these JavaScript functions using type information gathered by Ignition. WebAssembly, on the other hand, is not a high-level language, and not dynamically typed, and it is, therefore, not necessary to collect dynamic type information before compiling WebAssembly code [10].

The TurboFan compiler optimizes and compiles WebAssembly through a complicated pipeline that first decodes WebAssembly function bodies and constructs graph representations. These graph representations are in *Static Single Assignment form (SSA)* and use the "Sea of Nodes" concept introduced by Click in his dissertation [5]. TurboFan then applies optimizations to the SSA, selects appropriate instructions for the target architecture, allocates CPU registers, and finally generates code.

The Liftoff compiler, on the other hand, was designed to be fast at the cost of generating less optimized code. Even though it is newer than the TurboFan compiler, it is not meant as a replacement, but as the initial compilation stage to quickly produce a usable module. Like TurboFan, Liftoff begins by decoding WebAssembly function bodies, but then immediately begins code generation in a single pass, without constructing an SSA graph representation or optimizing the code [10].

4.1 WebAssembly JavaScript Interface

From an application developer's perspective, JavaScript applications can compile WebAssembly modules in two ways. The first is to call the constructor of the `WebAssembly.Module` class, which will synchronously compile the code, meaning that it will block the calling thread for the duration of the compilation. The second is the asynchronous function `WebAssembly.compile`, which will not block the calling thread.

By default, `WebAssembly.Module` uses Liftoff to compile the code, which is the faster compiler, and thus causes the smallest delay in the calling thread. V8 compiles the same module again, in a set of background threads, using the optimizing TurboFan compiler. When the optimized compilation result for a WebAssembly function is ready, the next invocation of the function uses the code produced by the TurboFan compiler instead of the output of Liftoff. This process is called "tiering up", and is a tradeoff between startup time and code generation quality [10].

`WebAssembly.compile`, on the other hand, is an asynchronous function and, therefore, not as concerned with blocking the calling thread. Its default behavior is to use the TurboFan compiler, skipping the baseline compilation step. This causes the compilation to generally take longer than synchronous compilation would, but produces the optimized result directly.

4.2 Performance of generated code

In order to compare the performance of code generated by Liftoff to code generated by TurboFan, we compiled the *PolyBench/C 4.2* benchmarks [23] to WebAssembly with compiler optimization and Link Time Optimization (LTO) enabled. These benchmarks are scientific computing kernels and were already used by Haas et al. [8] and Jangda et al. [15] to compare the performance of WebAssembly to the performance of native code execution. Instead, we use the benchmarks to compare the performance of code generated by Liftoff to the performance of code generated by TurboFan.

We conducted all experiments on Ubuntu 19.04 running on an Intel® Core™ i7-8700 processor (base frequency 3.20GHz, turbo frequency 4.60GHz, 6 cores, 12 threads) with 32GB of memory (2666 MHz). We used Node.js v14.2.0, the most recent Node.js version at the time of writing, which is based on V8 version 8.1.307.31-node.33.

We compiled and ran each of the 30 PolyBench/C benchmarks one hundred times with only Liftoff enabled, and another one hundred times with only TurboFan enabled. We measured the time it took for the benchmarks to complete, which does not include their respective compilation times. Figure 1 shows the average speedup of the code generated by TurboFan with respect to the code generated by Liftoff for each benchmark, with error bars indicating the standard deviation. All benchmarks were faster when compiled with TurboFan, the average speedup across all benchmarks is 2.0, and the maximum speedup is 3.2.

We also ran all benchmarks using V8's WebAssembly interpreter. On average, the PolyBench/C benchmarks were 247 times slower when interpreted than when compiled using TurboFan, and 115 times slower than when compiled using Liftoff. Sixteen of the 30 benchmarks were at least 200 times faster when compiled with TurboFan than when interpreted. While interpretation allows running WebAssembly code without prior compilation, its code execution is too slow for use in real applications.

We can conclude that the optimized code generated by TurboFan is indeed significantly faster than code generated by the baseline compiler Liftoff, and that the code produced by both compilers is

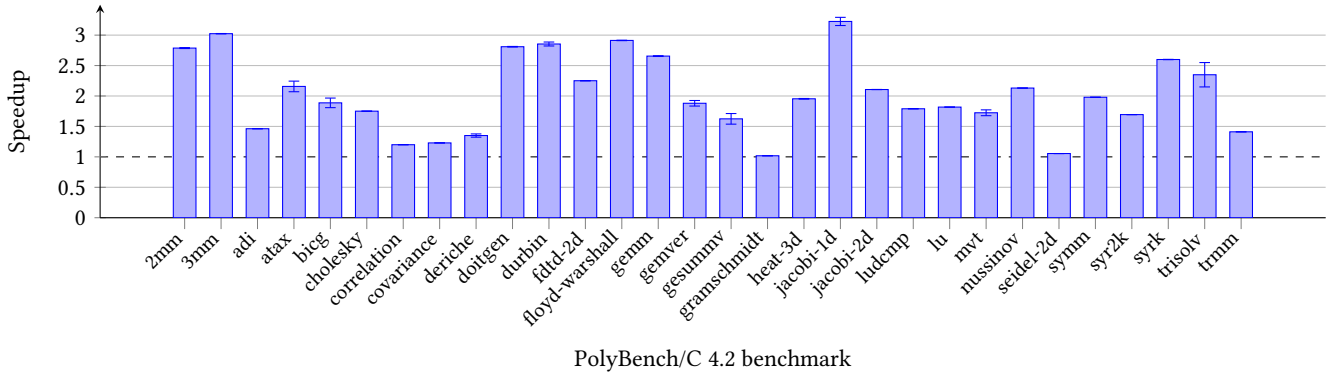


Figure 1: Speedup of code generated by TurboFan with respect to Liftoff

much faster than V8's WebAssembly interpreter. However, PolyBench/C is not a good basis for testing the performance of the compilers themselves, since the benchmarks are small and the WebAssembly modules are structurally very similar. In the following, we compare compilation times as well as CPU and memory footprint of both compilers.

5 CODE CACHING

Due to portability and security concerns, WebAssembly was designed to be compiled to the target architecture's instruction set at runtime. However, when running code from a trusted source on a single architecture, or untrusted code within a container or sandbox, these concerns become less relevant. Especially in scenarios where a Node.js application is expected to be initialized quickly, for example, when used as a command-line tool, as a desktop application, or in serverless computing, performance might be a more crucial factor. Here, using WebAssembly modules by first compiling them can cause visible delays.

5.1 Code extraction and insertion

Prior to designing a shared code cache, we need to find a way to efficiently retrieve compiled code from V8, and later inject the same code in a different V8 process.

While current versions of V8 provide such features for streaming WebAssembly compilation, no usable interface exists for Node.js, which only supports non-streaming WebAssembly compilation. However, V8 has internal functions that allow serializing compiled WebAssembly modules into byte sequences, and deserializing byte sequences into compiled WebAssembly modules. We developed an add-on for Node.js that exposes these internal V8 features to Node.js applications: `serialize` returns a JavaScript `ArrayBuffer` based on a given `WebAssembly.Module`, and `deserialize` creates a `WebAssembly.Module` based on the WebAssembly module bytes (referred to as "wire bytes" within V8) and the byte sequence generated by the `serialize` function.

This pair of functions is sufficient to extract code from a compiled module, store it in a cache entry, and later use the cache entry to obtain a usable module. This data flow is depicted in Figure 2.

V8 allows selectively disabling Liftoff and TurboFan. If a process is started with only Liftoff enabled, V8 prevents inserting code

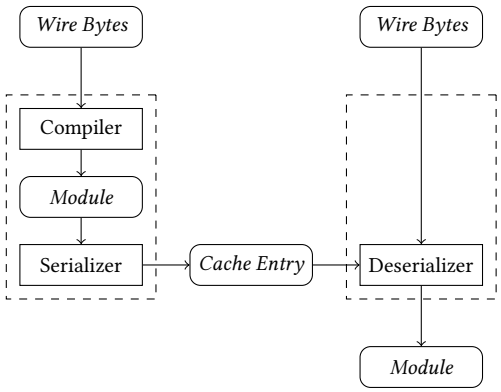


Figure 2: WebAssembly cache data flow: Cache entry creation (left) and cache entry retrieval (right)

generated by TurboFan (and vice versa). A proper cache lookup therefore requires knowledge about the current process's V8 configuration. To achieve this, our Node.js add-on allows applications to check relevant V8 flags.

In order to create realistic benchmarks, we extracted 115 WebAssembly modules from existing JavaScript applications, with module sizes ranging from as little as 1068 bytes to 37.3 MiB. It would be difficult to run the code represented by the WebAssembly modules in the way intended by their creators, since each module performs application-specific tasks and has certain requirements towards its host environment. However, our experiment is focused on compiling WebAssembly modules, which does not require running the compiled modules.

The approach Park et al. [20] used to cache compiled JavaScript code used cache entries that were much larger than the original JavaScript files. Similarly, we observe that serialized compiled WebAssembly modules are often considerably larger than the original WebAssembly files. Figure 3 shows the ratio of the serialized size to the original size based on the WebAssembly modules we extracted from existing applications, depending on which compiler was used by V8. In the case of JavaScript, better performance was achieved by caching optimized code in addition to intermediate bytecode, effectively increasing the size of cache entries [20]. For WebAssembly,

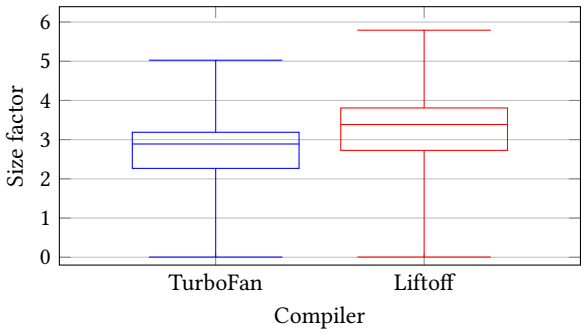


Figure 3: Ratio of serialized compilation result size to WebAssembly module size

it is sufficient to store the optimized compilation result (left side), which is significantly smaller than the result of the non-optimizing compiler Liftoff (right side), but still up to five times larger than the original WebAssembly module.

5.2 Compiler performance

Most importantly, we need to compare the performance of both compilers to the previously mentioned deserialization method. The hypothesis is that deserializing cached code uses less resources than compiling a WebAssembly module.

In order to test the hypothesis, we measured the real elapsed time it takes to obtain a compiled, usable module from the WebAssembly module bytes ("wire bytes"). To achieve comparable results, we disabled tiering up (see Section 4.1), only enabled one compiler at a time, and used the synchronous `WebAssembly.Module` constructor for Liftoff and TurboFan. Additionally, we took the same measurements for the previously described deserialization method. In this case, the module had already been compiled and optimized by the TurboFan compiler, and the serialized compiled module is available in memory (in addition to the wire bytes).

Each measurement is taken in a separate process. For each such process, we also record the CPU time of the process, that is, the total duration that threads of the process were scheduled on any of the CPU cores in user or system mode, and the peak physical memory usage through the `VmHWM` statistic provided by the Linux kernel. These metrics are equally if not more important than the elapsed real time required to compile a module. Even under the simplified assumption that CPU time and memory are the only resource constraints of a process, both of these resources are finite, and must be shared among all processes on the same system. While a high CPU time to real elapsed time ratio is an indication of well-designed parallelism, it also means that few concurrent instances of the same process might already use all available CPU time, and any additional instances could cause the performance of all processes to degrade. For cloud applications, it is realistic to assume that more than one process will be active on the same hardware at a time.

We recorded each measurement for each of the 115 WebAssembly modules 100 times. The mean values of elapsed real time, CPU time, and memory usage for each module are depicted in Figures 4, 5, and 6, respectively.

As shown in Figure 4, all three methods generally take longer for larger modules than for smaller ones. For legibility, Figures 4, 5, and 6 do not include errors bars. Instead, Figures 7, 8, and 9 show the significance of improvements. For two variables with mean values μ_1, μ_2 and standard deviations σ_1, σ_2, we define the *significance* of the change as $(\mu_1 - \mu_2)/(\sigma_1 + \sigma_2)$. By convention, we call the difference *statistically significant* if the significance is at least one.

By this definition, Liftoff was significantly faster than TurboFan for 111 modules (96.5%), used significantly less CPU time for 98 modules (85.2%), and had a significantly smaller memory footprint for 110 modules (95.7%).

When comparing deserialization to compilation using TurboFan, we measured statistically significant compilation time improvements for 112 modules (97.4%), CPU time improvements for 102 modules (88.7%), and memory usage improvements for 101 modules (87.8%). For 111 modules (96.5%), the speedup was at least 2, and for 77 modules (67.0%), the speedup was at least 20. Similarly, for 98 modules (85.2%), the CPU time was reduced by at least 50%, and for 54 modules (47.0%), it was reduced by at least 90%.

Compared to compilation using Liftoff, we observed significant compilation time improvements for 99 modules (86.1%), significant CPU time improvements for 77 modules (67.0%), and significant memory usage improvements for 41 modules (35.7%). For 69 modules (60.0%), the speedup was at least 2. The CPU time was reduced by at least 50% for 64 modules (55.7%).

The only statistically significant regression is an increase in memory usage for 39 modules (33.9%) when compared to Liftoff, and for 6 modules (5.2%) when compared to TurboFan. In these cases, however, the difference is small (less than 20%, see Figure 6).

Since the deserializer is synchronous, it is consistent to compare it to synchronous WebAssembly compilation. However, we also repeated this experiment with asynchronous WebAssembly compilation, and found that asynchronous compilation was significantly slower than synchronous compilation for 85 modules (73.9%) in the case of TurboFan, and for 91 modules (79.1%) in the case of Liftoff. For both compilers, this results in even larger differences when compared to deserialization, and we therefore decided not to present these results in detail.

We also repeated the experiment with deserialization of code generated by Liftoff instead of optimized code produced TurboFan, which leads to a larger serialized format (see Section 5.1). We found that it also causes longer deserialization times, and no significant improvements of real elapsed time, CPU time, or memory usage.

5.3 Module identification

The client needs to be able to identify each WebAssembly module in order to look it up in a cache. Since the JavaScript WebAssembly API is agnostic to the source of the WebAssembly module code, a module can only be identified by its code, and no file path or URL is available. Traditional information-theoretic algorithms, such as CRC32C, and cryptographic hash functions, such as SHA-1, provide reliable and, in the case of hash functions, collision-resistant identification methods. In scenarios where a hash collision could result in a security problem, cryptographic hash functions must be used for identification. However, these functions generally run in

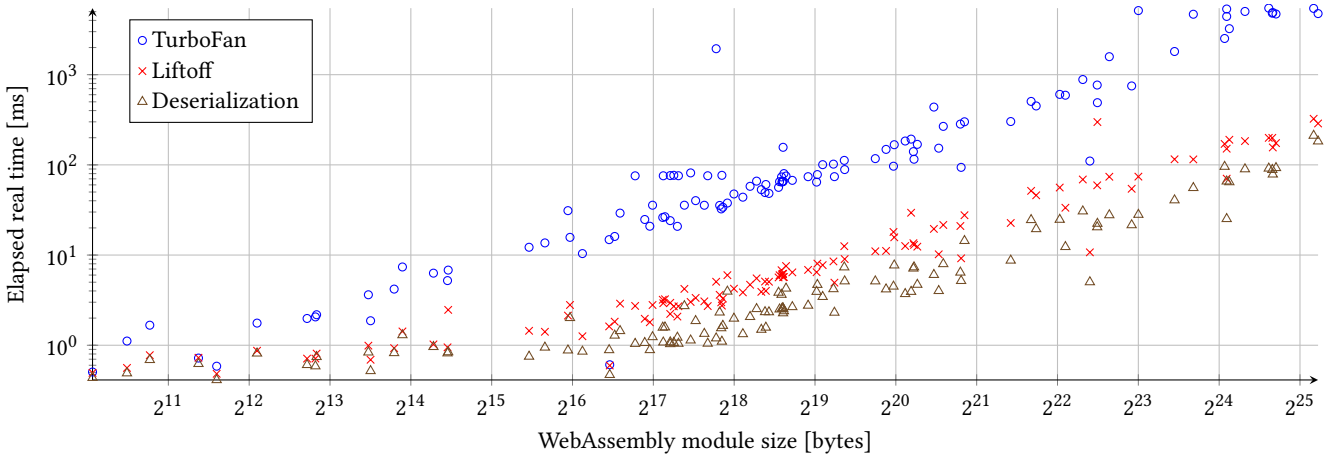


Figure 4: Compilation times by approach

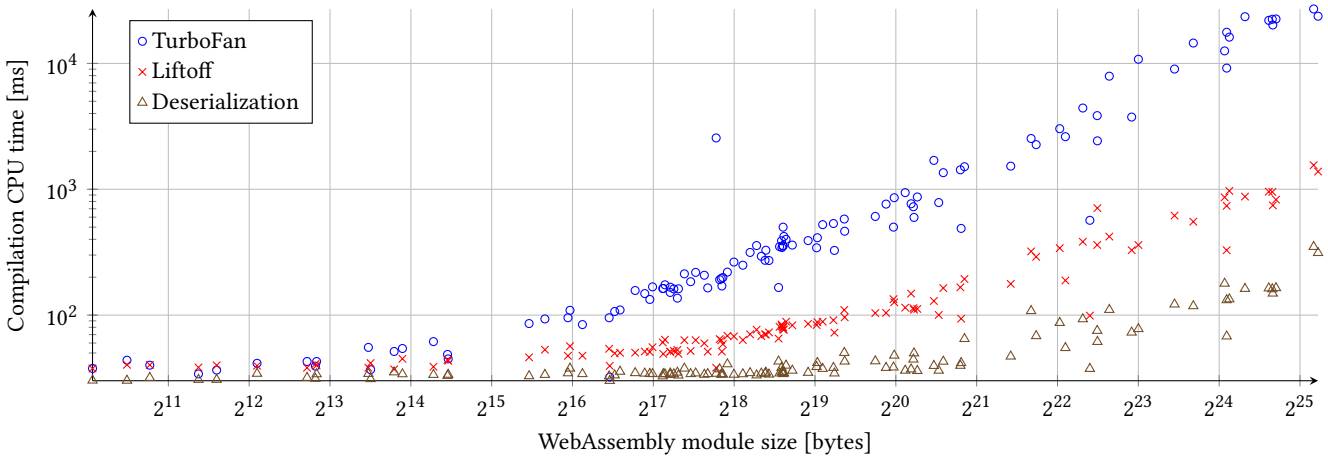


Figure 5: Compilation CPU times by approach

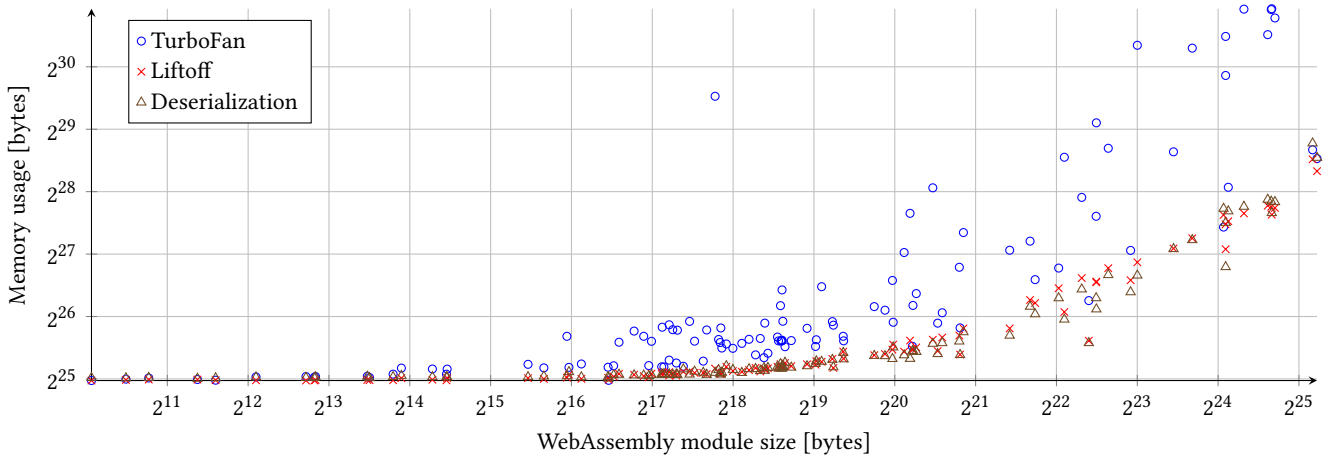


Figure 6: Compilation memory usage by approach

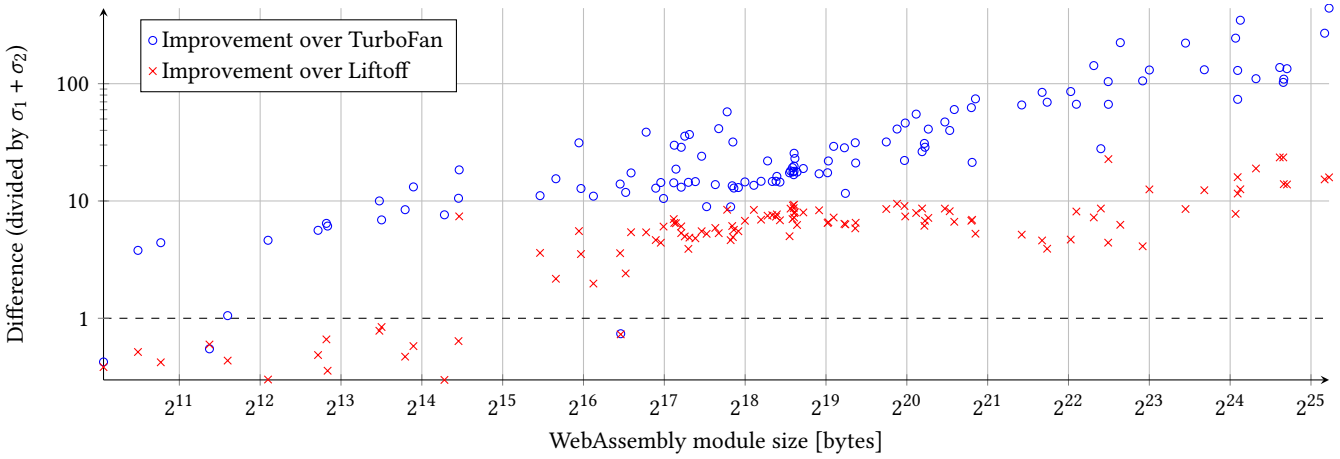


Figure 7: Significance of compilation time improvements

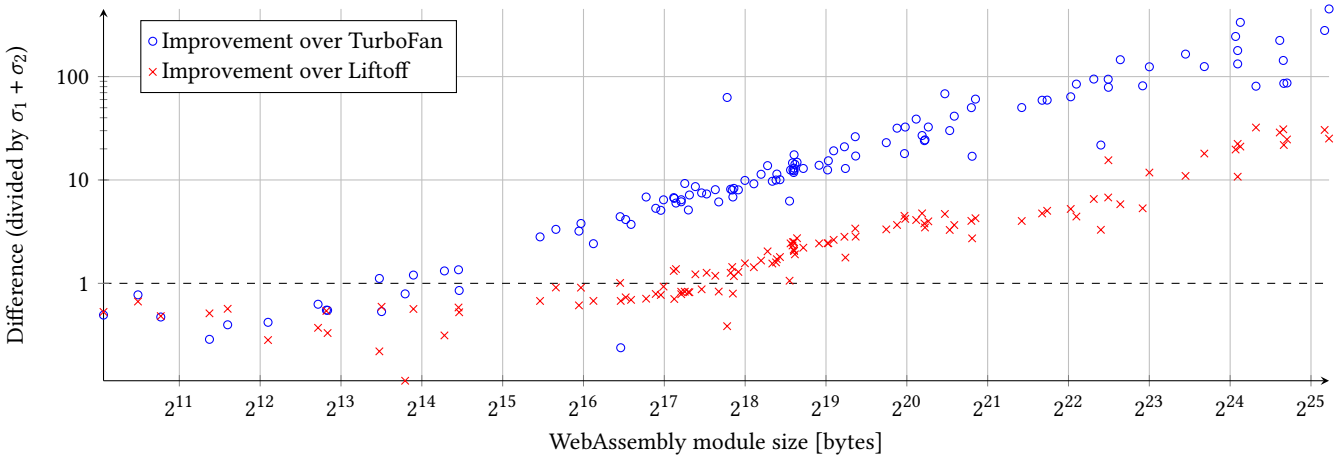


Figure 8: Significance of compilation CPU time improvements

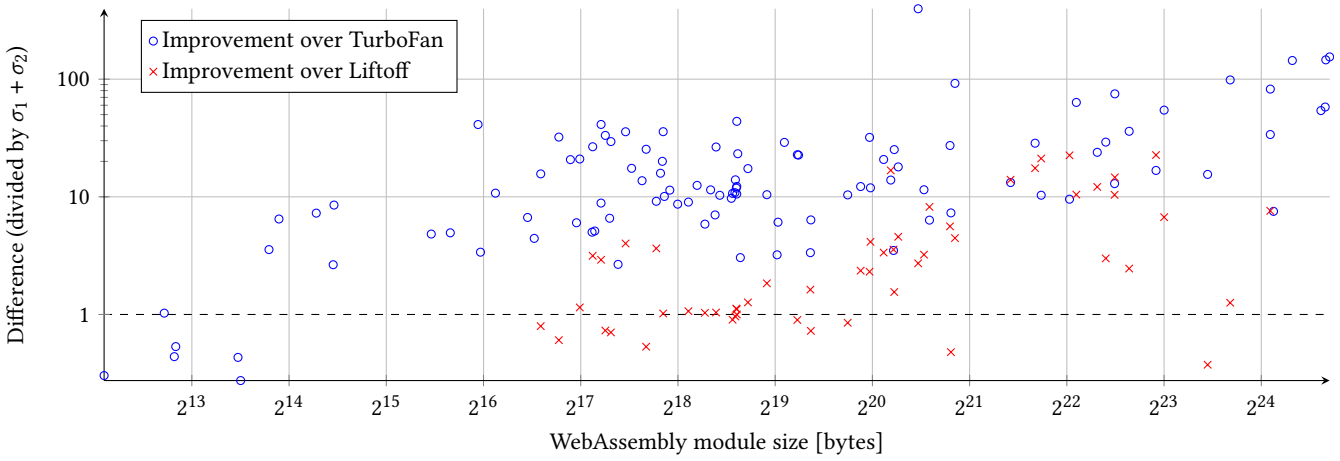


Figure 9: Significance of memory usage improvements

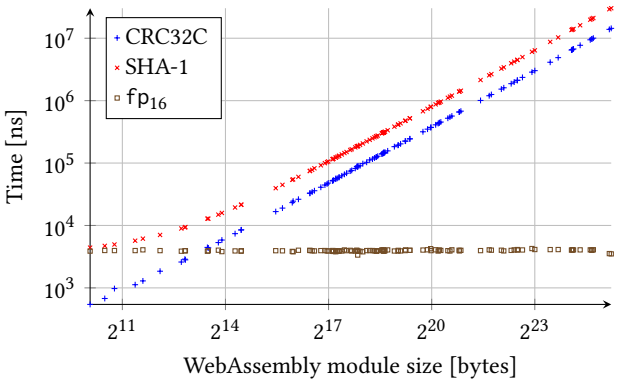


Figure 10: Performance comparison between `fp16`, CRC32C, and SHA-1 in Node.js

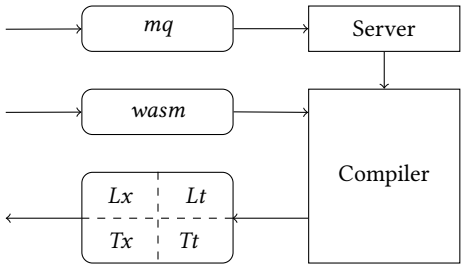


Figure 11: Shared cache server architecture

$\Theta(n)$, which is undesirable for large WebAssembly modules in situations where a collision would not result in a security problem, e.g., because access to the cache is restricted to a single client. For such cases, we define the function family $\texttt{fp}_r$ for $r \geq 2$ as follows: Let $b_0, \ldots, b_{n-1}$ be the input byte sequence, with $b_i \in \{0, \ldots, 2^8 - 1\}$ for $0 \leq i < n$. Let p be a linear congruential generator with $p(0) = b_{\lfloor n/2 \rfloor}$ and $p(i+1) = (a * p(i) + c) \bmod 2^{32}$. We chose $a = 1664525$ and $c = 1013904223$ as suggested by Knuth [17]. Let the result be the r byte vector $f_0, \ldots, f_{r-1}$ with $f_0 = \lfloor n/2^8 \rfloor \bmod 2^8$, $f_1 = n \bmod 2^8$, and $f_{2+i} = b_{p(i+1) \bmod n}$. In other words, the result consists of the length n modulo 2^{16} and the module bytes at $r-2$ pseudo-random locations. Each such function $\texttt{fp}_r$ only requires 32-bit integer arithmetic, can be implemented efficiently in JavaScript, and runs in $O(1)$. While the resulting "fingerprint" is neither unique nor collision-resistant, it is sufficiently unlikely to collide with another module's fingerprint. Figure 10 shows a performance comparison between $\texttt{fp}_{16}$, CRC32C, and SHA-1 in Node.js, based on running each function 100,000 times on each of the 115 WebAssembly modules. With a constant runtime of 4.0 μs, $\texttt{fp}_{16}$ is significantly faster than other identification methods.

6 SHARED CODE CACHE

With the previously discussed cache creation and retrieval method from Section 5.1 and the performance benefits of code caching presented in Section 5.2, it is viable to construct a disk-based cache that can be populated and used by individual processes. However, this approach is troublesome for large application clusters: First, write access to the code cache should be controlled strictly to prevent malicious code injections, and it might be undesirable for all processes that use WebAssembly to have permission to write compiled code to the cache. Second, if a process uses Liftoff or tiering up to improve its own startup time (see Sections 4 and 5.2), it might not insert optimized code into the cache, but instead the output of the baseline compiler. Third, a disk-based cache might reduce expected speedups due to disk access times associated with potentially large cache entries (see Section 5.1), which each new process might copy from disk into memory.

6.1 Design and Implementation

To circumvent these problems, we designed and implemented a novel approach to share compiled WebAssembly code between Node.js processes. In the following, we will refer to the processes of one or more Node.js applications as *client processes*. In this context, a *V8 configuration* is a set of flags that affect V8's internal behavior. The cache implementation prevents loading incompatible cache entries, and potentially maintains multiple cache entries for the same WebAssembly module, but for different V8 configurations. For example, Figure 11 includes a matrix of four configurations *Lx*, *Lt*, *Tx*, and *Tt*, where the first letter indicates which is the fastest enabled compiler (**L**iftoff or **T**urboFan), and the second indicates whether the compiler is used e**x**clusively, or if **t**iering up is enabled.

Client processes compile WebAssembly modules through a modified WebAssembly JavaScript Interface, which is compatible with the one described in Section 4.1. The compilation procedure, given a module represented by bytes $b_0, \ldots, b_{n-1}$, computes the module identifier $\texttt{fp}_{16}(b_0, \ldots, b_{n-1})$, and attempts to locate a cache entry in a shared memory segment based on the computed module identifier and the current V8 configuration. If such an entry exists, the compilaton procedure deserializes the cache entry to obtain a WebAssembly module instance without having to compile the module bytes. If no such entry exists, the client process copies $b_0, \ldots, b_{n-1}$ into a new shared memory segment (*wasm* in Figure 11), and sends a pointer to the new shared memory segment and the current V8 configuration to an existing message queue (*mq* in Figure 11). Finally, the client process falls back to V8's original compilation procedure, which, depending on the current configuration, uses Liftoff and/or TurboFan to compile the code.

A separate server process is responsible for creating the message queue *mq*. Upon receiving a pointer to a shared memory segment *wasm* along with a valid V8 configuration, the server process starts a new compiler process with parameters matching the received V8 configuration. The compiler process loads the module bytes $b_0, \ldots, b_{n-1}$ from the shared memory segment, unlinks the segment, and computes the module's fingerprint $\texttt{fp}_{16}(b_0, \ldots, b_{n-1})$. After ensuring that no other compiler process is already compiling the same module with the same V8 configuration, the process compiles the module. If TurboFan has not been disabled in the given V8 configuration, the compiler process uses it to produce optimized code. Only if TurboFan has been disabled, the compiler process uses Liftoff, and therefore generates unoptimized code. Once compilation finishes, the compiler process serializes the compiled WebAssembly module, and writes the result to shared memory.

The modified WebAssembly JavaScript Interface was implemented in JavaScript, except for the deserialization logic, which was implemented in C++ due to the necessity to access internal V8 features, and communication with the message queue, which was implemented in C++ and uses POSIX functions.

The server process code is written in C using POSIX functions to access the message queue. The compiler processes execute JavaScript code, and serialization logic written in C++. The actual compilation procedures are an existing part of V8.

6.2 Evaluation

To evaluate our design and implementation, we consider two cases:

Cache miss: When a client process fails to locate a compiled WebAssembly module in the shared cache, it not only needs to compile the module itself, but also suffers from two additional performance impairments. First, the client process needs to copy the WebAssembly module bytes to a shared memory segment, and notify the server process about the cache miss. Depending on the module size, this can cause a short delay before compilation begins. Second, while the client process compiles the module itself, the server process will spawn a compiler process, which also compiles the module, effectively increasing the system load, and potentially increasing compilation times.

Cache hit: Upon successfully locating a compiled WebAssembly module in the shared cache, the client process benefits from two performance aspects. First, it does not need to compile the module, which, on average, improves the time until the module is available, and likely reduces CPU load and memory footprint (see Section 5.2). According to the model proposed by Patros et al. [22], this also reduces performance interference on co-located cloud tenants. Second, if not forbidden by the process's V8 configuration, the obtained compiled code is already optimized, which would not be the case with V8's default *tiering up* behavior, or when using Liftoff. As we have seen in Section 4.2, this can lead to improved execution times.

While we already know the impact of deserialization as compared to compilation based on Section 5.2, we used PolyBench/C (see Section 4.2) to create a set of artificial Node.js applications to evaluate the performance impact of the shared cache. We measured the real elapsed time it takes for each application to compile and then execute its associated PolyBench/C benchmark. For this experiment, we use the default V8 configuration, which enables both Liftoff and TurboFan, and uses tiering up (see Section 4), and ran each application 45 times.

Figure 12 shows the mean execution times for cache misses and cache hits, with error bars corresponding to the standard deviation. Since execution times between benchmarks vary tremendously, all execution times were divided by the same measurement taken without a cache in place. Similarly, the ratio between execution time and compilation time varies greatly, therefore, we do not display compilation and execution times in a stacked manner.

As expected, we see a performance regression for cache misses. It is worth noting that the benchmarks with the largest (by percentage) performance regressions such as jacobi-1d are particularly short-running, which means that the delay caused by copying the module bytes to shared memory has a larger (by percentage) impact on the total elapsed time.

We also observe performance improvements for almost all benchmarks when a cache entry is found. The average speedup is 1.8, and the maximum speedup is 3.0. We expect that different Node.js applications would see vastly different performance benefits, depending on the WebAssembly modules in use.

Finally, most operating systems allow protecting shared memory segments from unintended write access. It appears that such measures allow controlling read and write access to the shared code cache sufficiently to prevent malicious code injections, for example, by only giving write access to the compiler processes, and not to client processes.

7 FUTURE WORK

A future direction for a shared code cache could be an extension to a disk-based cache. While the system kernel might keep frequently accessed cache entries in memory up to a certain size, large cache entries might still have to be loaded from the disk, and could negatively affect the cache performance. A balanced strategy might be to only move modules from shared memory to disk when most of the available memory is in use, and to prioritize frequently accessed modules in memory.

While our shared cache implementation prevents duplicate compilation on the server side, it does not prevent duplicate work among client processes. The primary reason is that client processes benefit from the shorter compilation times of tiering up, whereas the server process is focused on producing optimized code at the cost of longer compilation times. A future implementation could reduce the amount of duplicate work between processes further.

It might also be worth considering data compression for cache entries. Park et al. compressed cache entries in their JavaScript code cache implementation, and successfully reduced the size of the code cache with only minimal performance sacrifices [20]. However, as long as cache entries are stored in shared memory, decompression would require copying the decompressed data to a new memory area on each invocation, which makes it unlikely to result in large performance improvements. A disk-based cache solution could potentially benefit from compression to reduce cache entry sizes and therefore disk access times.

8 CONCLUSION

As we have seen in Section 5.2, compiling WebAssembly modules at runtime can lead to a delay of multiple seconds during an application's startup phase, and can require vast amounts of CPU time and physical memory. While *Liftoff* is much faster than the optimizing compiler *TurboFan*, its generated code is significantly slower than the code produced by *TurboFan*, but still much faster than interpreting WebAssembly code without compiling it first.

We reduced module load times by caching compiled and optimized code for the target architecture, and observed large performance benefits for many WebAssembly modules. Finally, in Section 6, we extended the idea to a scalable multi-process shared code cache, which provides an efficient way to load WebAssembly modules in Node.js applications, without having to compile and optimize each module in each process. The smaller CPU and memory footprint can reduce interference on co-located cloud tenants, and, therefore, improve scalability [22].

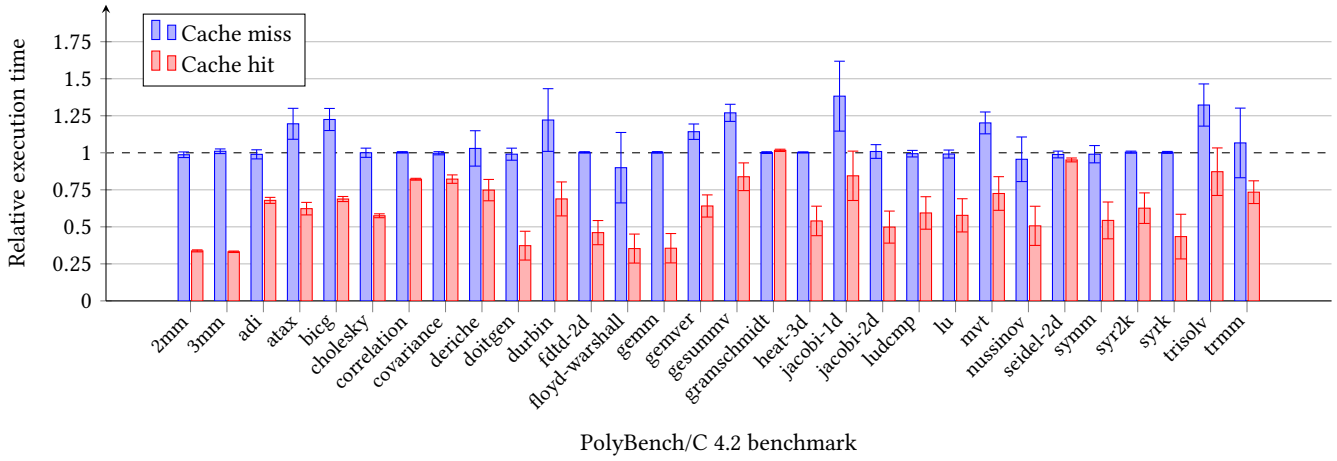


Figure 12: Shared cache performance impact on PolyBench/C benchmarks

While WebAssembly is still an emerging technology, we expect growing adoption over the next few years. These performance improvements and reduced startup times presented in this paper might allow widespread use of WebAssembly in serverless computing and other cloud configurations, without sacrificing the speed, portability, and security of WebAssembly.

9 ACKNOWLEDGMENTS

This research was conducted within the Centre for Advanced Studies — Atlantic, Faculty of Computer Science, University of New Brunswick. The authors are grateful for the colleagues and facilities of CAS Atlantic in supporting our research. The authors would like to acknowledge the funding support of the Natural Sciences and Engineering Research Council of Canada (NSERC), 501197-16. Furthermore, we would also like to thank the New Brunswick Innovation Foundation for contributing to this project.

REFERENCES

[1] 2018. Indexed Database API 2.0. https://www.w3.org/TR/IndexedDB/
[2] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, and Philippe Suter. 2017. Serverless Computing: Current Trends and Open Problems. In *Research Advances in Cloud Computing*, Sanjay Chaudhary, Gaurav Somani, and Rajkumar Buyya (Eds.). Springer Singapore, Singapore, 1–20.
[3] Devarghya Bhattacharya, Kenneth B. Kent, Eric Aubanel, Daniel Heidinga, Peter Shipton, and Aleksandar Micic. 2017. Improving the performance of JVM startup using the shared class cache. In *2017 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*. IEEE, Victoria, BC.
[4] Bill Budge. 2019. Code caching for WebAssembly developers. https://v8.dev/blog/wasm-code-caching
[5] Cliff Click and Keith D. Cooper. 1995. Combining analyses, combining optimizations. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 17, 2 (March 1995), 181–196.
[6] Michael H. Dawson, Dayal D. Dilli, Kenneth B. Kent, Panagiotis Patros, and Peter D. Shipton. 2019. Dynamically compiled artifact sharing on PaaS clouds. Patent US 10,338,899 B2.
[7] Node.js Foundation. [n.d.]. C++ Addons. https://nodejs.org/api/addons.html
[8] Andreas Haas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and Jf Bastien. 2017. Bringing the web up to speed with WebAssembly. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation - PLDI 2017*. ACM Press, Barcelona, Spain, 185–200.
[9] Adam Hall and Umakishore Ramachandran. 2019. An execution model for serverless functions at the edge. In *Proceedings of the International Conference on Internet of Things Design and Implementation - IoTDI '19*. ACM Press, Montreal, Quebec, Canada, 225–236.
[10] Clemens Hammacher. 2018. Liftoff: a new baseline compiler for WebAssembly in V8. https://v8.dev/blog/liftoff
[11] Daniel Heidinga, Peter D. Shipton, Aleksandar Micic, Devarghya Bhattacharya, and Kenneth B. Kent. 2020. Enhancing Virtual Machine Performance Using Autonomics. Patent US 10,606,629 B2.
[12] David Herman, Luke Wagner, and Alon Zakai. 2014. asm.js. http://asmjs.org/
[13] David Herrera, Hanfeng Chen, Erick Lavoie, and Laurie Hendren. 2018. Numerical computing on the web: benchmarking for the future. In *Proceedings of the 14th ACM SIGPLAN International Symposium on Dynamic Languages - DLS 2018*. ACM Press, Boston, MA, USA, 88–100.
[14] David Herrera, Laurie Hendren, Hangfen Chen, and Erick Lavoie. 2018. *WebAssembly and JavaScript Challenge: Numerical program performance using modern browser technologies and devices.* Technical Report SABLE-TR-2018-2. Sable Research Group, School of Computer Science, McGill University, Montréal, Canada.
[15] Abhinav Jangda, Bobby Powers, Emery D. Berger, and Arjun Guha. 2019. Not So Fast: Analyzing the Performance of WebAssembly vs. Native Code. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. USENIX Association, Renton, WA, 107–120.
[16] Faiz Khan, Vincent Foley-Bourgon, Sujay Kathrotia, and Erick Lavoie. 2014. Ostrich Benchmark Suite. https://github.com/Sable/Ostrich
[17] Donald E. Knuth. 1981. *The Art of Computer Programming, Volume II: Seminumerical Algorithms, 2nd Edition.* Addison-Wesley.
[18] Bernd Malle, Nicola Giuliani, Peter Kieseberg, and Andreas Holzinger. 2018. The Need for Speed of AI Applications: Performance Comparison of Native vs. Browser-based Algorithm Implementations. *arXiv:1802.03707* (Feb. 2018).
[19] Hiroyuki Matsuo, Shinsuke Matsumoto, Yoshiki Higo, and Shinji Kusumoto. 2019. Madoop: Improving Browser-Based Volunteer Computing Based on Modern Web Technologies. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, Hangzhou, China, 634–638.
[20] Hyukwoo Park, Sungkook Kim, Jung-Geun Park, and Soo-Mook Moon. 2018. Reusing the Optimized Code for JavaScript Ahead-of-Time Compilation. *ACM Transactions on Architecture and Code Optimization* 15, 4 (Dec. 2018), 1–20.
[21] Panagiotis Patros, Dayal Dilli, Kenneth B. Kent, and Michael Dawson. 2017. Dynamically Compiled Artifact Sharing for Clouds. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, Honolulu, HI, USA, 290–300.
[22] Panagiotis Patros, Stephen A. MacKay, Kenneth B. Kent, and Michael Dawson. 2016. Investigating Resource Interference and Scaling on Multitenant PaaS Clouds. In *Proceedings of the 26th Annual International Conference on Computer Science and Software Engineering (CASCON '16)*. IBM Corp., USA, 166–177.
[23] Louis-Noel Pouchet and Tomofumi Yuki. 2016. PolyBench/C. https://web.cse.ohio-state.edu/~pouchet.2/software/polybench/
[24] Micha Reiser and Luc Bläser. 2017. Accelerate JavaScript applications by cross-compiling to WebAssembly. In *Proceedings of the 9th ACM SIGPLAN International Workshop on Virtual Machines and Intermediate Languages - VMIL 2017*. ACM Press, Vancouver, BC, Canada, 10–17.
[25] V8 Team. 2017. Launching Ignition and TurboFan. https://v8.dev/blog/launching-ignition-and-turbofan
[26] Seth Thompson. 2016. Experimental support for WebAssembly in V8. https://v8.dev/blog/webassembly-experimental

An ELF-based Storage Option for the Eclipse OMR Ahead-of-Time Compiler

Damian Diago D'monte
ddmonte@unb.ca
University of New Brunswick
Fredericton, New Brunswick, Canada

Georgiy Krylov
georgiy.krylov@unb.ca
University of New Brunswick
Fredericton, New Brunswick, Canada

Daryl Maier
maier@ca.ibm.com
IBM Canada Ltd.
Markham, Ontario, Canada

Gerhard W. Dueck
gdueck@unb.ca
University of New Brunswick
Fredericton, New Brunswick, Canada

Kenneth B. Kent
ken@unb.ca
University of New Brunswick
Fredericton, New Brunswick, Canada

ABSTRACT

Ahead-of-time (AOT) compilation involves converting program code into native code prior to dynamically linking and loading into memory at runtime. Apart from code generation, code storage and persistence are the major aspects of AOT compilation. Eclipse OMR, a toolkit for construction of language runtimes, could benefit from enhanced storage container support for AOT compiled code. Our focus is on the code storage, sharing and reuse options available for the AOT compiler module in Eclipse OMR. We propose to increase the utilization of Executable and Linkable Format (ELF) shared objects for storing the AOT compiled code and data in Eclipse OMR. The paper continues by surveying state-of-the-art runtimes to explore the opportunities for persisting AOT compiled code in active projects. Moreover, the research uncovers the benefits of the proposed approach and possible challenges such as symbol resolution, patching the shared ELF object, dynamic linking and loading.

CCS CONCEPTS

• **Software and its engineering → Compilers**; **Runtime environments**; **Code storage**; **Dynamic linking and loading**.

KEYWORDS

Ahead-of-time compilation, runtime systems, shared libraries

ACM Reference Format:
Damian Diago D'monte, Georgiy Krylov, Daryl Maier, Gerhard W. Dueck, and Kenneth B. Kent. 2020. An ELF-based Storage Option for the Eclipse OMR Ahead-of-Time Compiler. In *Proceedings of 30th Annual International Conference on Computer Science and Software Engineering (CASCON'20).* ACM, New York, NY, USA, 6 pages.

1 INTRODUCTION

Modern programming languages such as Java, Python, and C# require a managed runtime environment for program execution. Runtime environments provide automatic memory management, code compilation, execution and storage mechanisms. Language runtime environments use interpreters to execute instructions one by one directly, or a compiler is used to convert a source program into native code.

Originally, Java runtimes only interpreted the bytecode line-by-line and executed it at runtime [1]. The problem with using only interpreters was the overhead of fetch-decode operations from the interpreting process. For instance, a piece of code in a loop needs to be retranslated every time, causing performance issues. To avoid this, a Just-in-Time (JIT) compiler feature was introduced [1]. The JIT compiler incurs the one-time cost of compiling the block of interpreted code into native code and allowing for later reuse within the same instance of a runtime environment, thereby saving the execution overhead caused by an interpreter, since decoding is no longer required. Moreover, the JIT compilation allows compile-time optimizations [1]. JIT compilation has enabled Java virtual machines to dynamically convert bytecodes into native code. Another traditional compilation method is Ahead-of-Time (AOT) compilation, also known as static compilation. In AOT compilation, the source code or an intermediate representation (IR) is converted into native code before the execution, i.e., before the runtime is invoked [20]. During execution, reusing AOT compiled code rather than recompiling the same code again can enhance the startup time of the VM. This AOT generated code needs to be stored in a container format such as an object format or an equivalent. For example, C generates an object file with a *.o* extension, whereas Java generates a *.class* file (bytecode) post compilation.

Modern runtime environments allow for concurrent execution of multiple instances as separate processes. Compiled code sharing techniques and complex dynamic solutions help in enabling such concurrency. One example of such a technique, is to store the compiled code in a shared library. A shared library is a collection of pre-compiled pieces of code that can be reused by other programs. These libraries store methods, routines, classes, data structures and other information that can be shared. Our research highlights Eclipse OMR, a set of open source modules that can be used to build robust language runtimes [13].

The AOT compiler for Eclipse OMR is defined by a set of modules performing related functions and their interaction. The four main functions of an AOT compiler in Eclipse OMR are code generation, code loading, code persisting and compilation control. Code generation is performed by the existing Eclipse OMR compiler module,

CASCON'20, November 10–13, 2020, Toronto, Canada

also known as the Testarossa compiler. The code loading part is currently a work in progress for OMR [20], whereas a module for dynamic linking and loading is an ongoing project. The compilation control module is non-existent and the logic is left to the runtime constructors for implementation. This research is focused on the storage components that exist in OMR for storing the AOT compiled code. Currently, the Eclipse OMR supports the generation of executable and relocatable Executable and Linkable Format (ELF) object types. However, these object types do not allow code sharing, modification or operations like dynamic linking and loading at runtime in Eclipse OMR. A key component of this research is to enhance ELF capabilities in Eclipse OMR by introducing the ELF shared libraries in order to elevate shareability of AOT code and support dynamic operations.

The paper provides motivation for the research, as well as outlines design ideas and is structured as follows: Section 2 introduces ELF and outlines its characteristics, focusing on ELF shared libraries; Section 3 reviews current state-of-the-art runtimes and their AOT code storage containers; Section 4 describes the ELF infrastructure in Eclipse OMR and states its limitations; Section 5 proposes a technique for storing the AOT generated code and discusses the potential challenges in its implementation; and Section 6 concludes the paper.

2 ELF OBJECT OVERVIEW

Executable and Linking Format, also known as ELF, is a portable object file format developed by UNIX System Laboratories. ELF describes the structure of the object file in Unix and Linux systems [6]. Executable files, shared libraries (.so), object files (.o) and core dumps are examples of files that comply with the ELF standard [21]. The executable object file contains an execution-ready program code. The relocatable object file contains code that is used for linking with other object files. Such object files need to be processed by the linker before running them. Generation of an ELF object file is either the consequence of compilation or the linking activity. The resulting ELF object may contain program code, information related to methods, metadata, variables, relocations and other data. Compilers, assemblers and linkers treat ELF file as a set of sections described by a section header table, while loaders treat the file as a set of segments described by a program header table.

2.1 Shared Libraries

Shared object files (.so), also referred to as shared libraries, are the most significant kind of ELF files for this research. The linker can either link the shared object to a relocatable object to create a new object or can link it with an executable object file to generate a process image i.e., an executable. Shared libraries can be modified, extended and recompiled independently [21]. Shared libraries can be loaded into the memory and linked dynamically at load time or runtime. Unlike the relocatable (.o) ELF object files, they are not embedded into the final executable, but are needed to be accessible wherever the executable refers to them. The process of creating a shared library directly from the C program using the GNU Compiler Collection (GCC) is described below.

```
gcc -Wall -Werror -fPIC test.c
gcc -shared -o libtest.so test.o
```

The GCC provides built-in support to produce an ELF object file as an outcome of the compilation activity. In the first command, GCC compiles the program `test.c` and stores the compiled code in the `test.o` file i.e., a relocatable object. The `fPIC` flag indicates the position independent code (PIC), whereas the `-shared` flag indicates that the library can be shared. The second command will create a shared object file of the relocatable object file. This generated `libtest.so` shared library can be linked to other executables or loaded at runtime.

2.2 Sections and Metadata

A shared library is viewed as a collection of sections, where each section carries code and data concerning the object file. The four canonical sections are *.text, .data, .rodata* and *.bss*. These sections hold binary code, data, initialized and uninitialized variables associated with the program. Other sections, *.rel.text, .rel.data* and *.rel.rodata* hold tabular information associated to relocation patches for *.text, .data, .rodata* respectively [21].

The ELF metadata comprises the ELF header, program header table, section header table and other metadata sections. The ELF header resides at the beginning of the ELF object file and holds information related to the ELF identifier, file type, architecture, version, section and program headers. The program header table holds the details related to segments, whereas, the section header table contains an entry for each section. Apart from these sections, the shared libraries contain *.symtab, .strtab* and *.hash* sections. Although the symbol table (*.symtab* section) stores all the global variables and functions used by the program, all names corresponding to the symbol table entries are stored in the string table (*.strtab* section) [24]. The *.hash* section holds the hash table, which provides fast access to symbol table entries without using a linear search.

Specialized sections like *.dynsym, .dynstr, .got* and *.plt* are used to promote the dynamic linking process. The *.dynsym* section contains the dynamic linker symbol table that stores all the imported and exported symbols of the file. All string names corresponding to dynamic symbol linker tables are stored in the dynamic string table, which is located in the *.dynstr* section. The *.got* section holds the global offset table (GOT), which stores pointers for each global variable (static data) defined or referenced by the shared library. The *.plt* section holds a procedure linkage table (PLT), which contains an entry for each non-local routine called from the shared library [21]. The *.dynamic* section holds the address and size of the string table, symbol table, hash table and relocation tables.

2.3 ELF Characteristics

The ELF specification encompasses several attributes that distinguishes it from other existing object file formats like COFF, a.out, etc. Crucial traits of the ELF object include:

2.3.1 Broad Platform Support. The ELF standard provides a set of binary interface definitions that support several operating systems. These interfaces reduce the need for recording and recompiling to smoothen the software development process [6]. Starting with GCC compiler version 2.7, the ELF binary was chosen as the default object file format for Linux [2]. Compilers that run on Unix platforms support ELF as the standard file format [3]. Furthermore, the Tool Interface Standards committee (TIS) chose the ELF standard as a

portable object file format. The latest version of Microsoft Windows i.e., Windows 10, includes a Windows subsystem for Linux that enables ELF support [25].

2.3.2 Extensible and Flexible. ELF files are defined to be extensible to larger architectures and are not bound to a specific processor or instruction set [21]. Additionally, they do not rely on a specific word length or data alignment, like the big endian or little endian, and support both [24]. Except for the ELF header, every other section and segment has no predetermined position or size and is extremely adaptable. The ELF design provides sufficient flexibility to modify the existing data and add arbitrary optional sections.

2.3.3 Code Sharing and Reuse. Unlike the relocatable and executable ELF objects, ELF shared libraries are capable of sharing code between multiple virtual machine instances at runtime. This enhances code sharing and decreases the redundancy of code in memory. For instance, identical methods in multiple programs will have only one copy stored in the shared object. This copy can be consumed by multiple processes at once. This saves memory and does not require loading the same method twice.

2.3.4 Position Independent Code (PIC). PIC is the basis of ELF shared libraries that enable them to be loaded at any address in memory [31]. To achieve PIC, the GOT and PLT are introduced to ELF. Each ELF executable and the shared object associated with it contains a PLT. The PLT is a kind of jump table, which contains entries that instructs an indirect jump to a symbol global offset table entry. The GOT adds a level of indirection for static data (global variables), whereas PLT adds a level of indirection for function calls [21]. The PLT permits lazy evaluation, that is, it does not resolve the procedure address until they are called for the first time.

2.3.5 Dynamic. ELF promotes dynamic linking and loading in shared libraries. The three responsibilities of the dynamic linking and loading are to perform relocations, resolve symbols and load the code into main memory. The dynamic linker resolves and relocates all the pointers of the global offset table [21]. To perform automated linking and loading, linux provides two programs, *ld.so* and *ld-linux.so* [23]. They find and load the shared libraries required by the program to execute. The dynamic linker is also known as the program interpreter and can be accessed from the *.interp* section [23]. Dynamically linked shared libraries allow programs to load and unload routines at runtime, which cannot be achieved using the static ones. To gain access to a dynamic shared library, the *dlopen*(3) function is used. Along with *dlopen*(), *dlsym*(), *dlclose*(), *dlerror*() implement the interface to the dynamic linking loader [22]. The *dlopen*() function returns a handle that can be employed with other functions in the *dlopen* API, such as *dlsym*() and *dlclose*() [22]. One of the arguments to *dlopen*() is the mode, which controls the visibility of symbols and relocations. For instance, *RTLD_LAZY* mode performs lazy binding and resolves symbols only when the code referencing it is executed.

2.3.6 Tooling Support. The ELF specification is supported by numerous tools, libraries and utilities for manipulating objects and libraries. The GNU binary utilities (*Binutils*) [5] introduce the *readelf* and *objdump* utilities, which can access sections data and display the information about object files in a human readable format. *Binutils* provides the *objcopy* utility, which can rename sections, update sections, add symbols, copy contents of one object file to another and perform various other operations. The *libelf* library, from the *elfutils* package [4], is capable of reading, modifying and creating the ELF object files. For managing portable dynamic shared libraries, a tool called *GNU Libtool* can be employed.

3 RELATED WORK

The concept of storing and loading compiled code in the state-of-the-art ELF shared library is an open-ended research topic. For instance, members of the WebAssembly community advocate for the usage of ELF objects as a container format [15]. The discussion mainly focuses on the characteristics of the ELF format, the advantages and the downsides of using it for the WebAssembly language runtime. Likewise, a similar discussion was held about the possibility of using the ELF object for AOT compilation in Eclipse OMR [17]. Such an ongoing exchange of views indicates continued interest in ELF object files. AOT compilation is adopted by several well-known language runtimes such as the Java Virtual Machines (JVM), Mono, the Android Runtime (ART), V8 and certain Python implementations like PyPy. In this section, we review a handful of runtimes and their AOT compilation techniques. The primary emphasis will be on the storage of AOT compiled code and data.

3.1 Eclipse OpenJ9

Eclipse OpenJ9 is an open-sourced JVM implementation based on the IBM J9 JVM. It supports JIT compilation as well as AOT compilation and stores the compiled code in a common container format termed the Shared Class Cache (SCC) [8]. In addition to compiled code, the SCC also stores metadata required for execution and linking. Typically, the SCC is located in shared memory and exists beyond the lifetime of the JVM. Also, it is a fixed size cache that allows storing AOT code until there is no free space available.

In Eclipse OpenJ9, the AOT compiler is automatically activated upon enabling the SCC [9]. The JVM splits and stores a class in immutable (read-only) and mutable (writable) portions [8]. Caching AOT method data reduces the effect of JIT compilation as all subsequent runs can utilize the cached data rather than JIT compiling it again. The JVM decides the extent of AOT methods to be stored in the shared cache. Typically, the AOT method data is around 10% of the total class shared data. There is no limitation on the number of SCCs and no particular JVM owns any SCC. However, a JVM can connect and read from only one SCC at a time. The SCC locates a fixed-sized AOT relocation header in addition to the AOT code. The relocation records along with validation records are packed contiguously into fixed-size blocks in the AOT relocation header [29]. Another benefit is that the persistent SCCs can be moved between machines having the same operating systems and hardware specification. The SCC improves the startup time of the VM in its subsequent runs as the native code stored in the SCC is loaded.

3.2 Mono Runtime

Mono Runtime is an open-sourced implementation of Microsoft's .NET framework [28]. The AOT compiler in Mono uses the object

format native to the target platform to store the AOT compiled code [27]. On ELF supported platforms, it generates a shared object file (.so) also called an AOT image. However, on other platforms it generates an assembly (.s) file which in turn can be assembled and linked into a shared object file.

For method compilation, Mono uses the JIT compiler, which generates compiled code and relocation patches [27]. The global offset table (GOT) and the procedure linkage table (PLT) used in the Mono Runtime are similar to the ELF Specification [27]. The AOT compiler considers all the function calls in the program as one type of patch and stores it into the PLT. Everything, other than function calls, is regarded as "other" types of patches, which are stored in the GOT. The indirection in the GOT, required by the AOT compiler, is deemed as a potential performance issue.

Other than storing the position independent code generated by the AOT compiler, the AOT image also stores cached metadata. The cached metadata is a variety of information like instance size, type initializer, etc., that is useful for class loading at runtime. Computing the cached metadata is time consuming and requires creation of runtime data structures. Hence, the required information is computed during AOT compilation and stored in the AOT image, into a *class_info* array. In Mono AOT, a shared object containing only the metadata can be produced using the *metadata-only* compilation option.

3.3 Oracle HotSpot

Another prominent Java runtime is the HotSpot JVM by Oracle [16]. Static AOT compilation in the HotSpot JVM employs the Graal framework for generating the AOT code [10]. The code container utilized for storing AOT code in the HotSpot JVM is the ELF shared library (.so) file. These ELF shared libraries are produced using the libelf library from the *GNU elfutils* project [19]. Hotspot uses the *javac* tool to compile a source file and generate a *.class* file. For AOT compilation, the tool *jaotc* is used to generate the native code of the processed class file. Commonly, the code generated in the HotSpot AOT compilation is treated as an extension of the existing HotSpot Code Cache. To AOT compile only specific methods, *jaotc* provides a *compileOnly* flag, whereas to exclude methods from compilation it offers an *exclude* flag.

Tiered Compilation (TC) mode in HotSpot is able to collect and store profiling information along with the code. The HotSpot JVM introduces an interesting notion of class fingerprinting [19]. This technique stores the fingerprint of each class after AOT compilation in the *.data* section of the ELF shared libary. The objective of this technique is to determine if the class has been changed after AOT compilation. This is done by matching the current fingerprint of the class with the one stored in the shared library during loading. The relocation data is stored similarly to Mono in its reliance on the GOT and PLT tables [29]. The HotSpot JVM needs the same runtime configuration for compilation and execution. In JDK 10 SE, AOT compilation is an experimental feature and is supported only on Linux-x64 [10].

4 AOT COMPILATION IN ECLIPSE OMR

The term *ahead-of-time compiled* is mentioned in several related works [7, 18], however Eclipse OMR does not provide full support for it, the feature is in its early development state [18]. The current AOT infrastructure provides an interface called *AOTStorageInterface*, to connect to a storage interface. *AOTStorageInterface* defines two methods, *storeEntry* and *loadEntry*, keyed by method name and containing the size of the data. These methods are required to be extended in order to provide an implementation for storing and loading from a code container.

There are two in-development approaches for persisting code and data in Eclipse OMR. The first one is the Shared Cache [30] module, inspired by the Eclipse OpenJ9 Shared Class Cache, described in Section 3.1. The second option is to extend ELF file support.

4.1 Eclipse OMR ELF Infrastructure

The Eclipse OMR Compiler Technology provides fundamental support for generation of ELF object files. In generating an ELF object in OMR, there are two types of ELF objects supported, the executable and the relocatable (.o) object [14]. The executable object file has an extra/auxiliary program header, whereas the relocatable object file includes an extra section to hold relocation related information. Both implementations support 32-bit and 64-bit architectures. Compiled code stored in the Eclipse OMR object is retrieved directly from the code cache segment of Eclipse OMR [14]. Hence, the size of code cache segment is required before instantiating the object file generation operation.

4.1.1 Limitations. In Eclipse OMR, relocatable object files contain *.text*, *.data*, *.rela.txt*, *.symtab*, *.dynstr* and *.shstrtab* sections. Whereas, the executable object file incorporates program header, *.text*, *.data*, *.symtab*, *.dynstr* and *.shstrtab* sections. These sections are in accordance with the semantics of the ELF Specification. Currently, both variants are not comprehensive in Eclipse OMR, but hold adequate information to serve the primary purpose of generating an object file. The primary issue with executable and relocatable ELF objects in Eclipse OMR is its sharing abilities. The code stored in these objects cannot be shared between multiple virtual machines at runtime. Due to the absence of *.got* and *.plt* sections in relocatable ELF objects in Eclipse OMR, operations like dynamic linking and dynamic loading cannot be performed. Additionally, lack of a framework to modify or patch an existing ELF object makes it difficult to consume the current ELF infrastructure as-is.

5 PROPOSED SOLUTION

To elevate ELF support in Eclipse OMR, the ELF shared object file can be adapted. As described earlier, this enhancement can provide various benefits like code reusability and sharing between multiple runtime instances at the same time. In our approach, the native code generated after AOT compilation will be stored in the shared library as depicted in Figure 1. This shared library can be dynamically loaded into memory by the executable referring to it at runtime. For example, in a Java runtime, the *.class* bytecode can be AOT compiled using the Eclipse OMR AOT compiler and stored into a shared object. This shared object can be linked and loaded when the JVM is invoked.

In the initial development, each program will generate a single library file. This approach will have small-sized libraries that could easily link to and be loaded in memory. However, the code reuse will not be possible in such a case. In later developments, the support

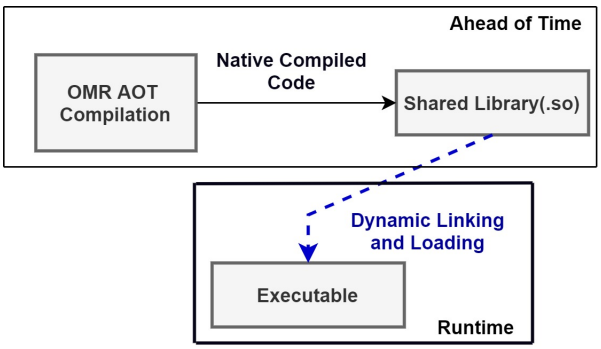


Figure 1: Schematic representation for proposed shared libraries use case within runtime environments utilizing Eclipse OMR AOT Compiler

will be extended to generate a single shared library file for a set of programs. Such extensions will require continuous appending and patching as described in Section 5.1.

The research proposes to allow the Eclipse OMR AOT compiler to communicate with a storage controller class through the *AOTStorageInterface* interface. Our design is preliminary and requires little implementation effort to attach to the existing storage controller. Alternatively, the proposed design allows for choosing between storage options available in OMR to suit developer needs.

5.1 Challenges

In this section, potential challenges associated with implementation intricacies are summarized.

5.1.1 Persisting the Code. The idea is to persist the code stored in the ELF shared object. For instance, after compiling multiple programs, the binary code and related data generated for each program should be stored in a single shared object file. This demands affixing the shared object file post-compilation for each program. As per the ELF specification and the existing implementation in the Eclipse OMR repository, the binary code is stored in the *.text* section. In this case, *.text* section needs to be updated along with the offsets of following sections. Once the modification is complete, the ELF shared object file needs to be re-emitted.

5.1.2 Patching. The process of patching is altering the stored code and data of a modified program in the shared object files. Patching is vital, in case of a class or method definition change after compiling and storing for the first time. If not done, the alteration would not be reflected in the shared object and in all subsequent runs post-changing, the definition will use the incorrect code to execute, leaving the application in an inconsistent state. A simple approach for this problem is to replace the existing code and data with the newly compiled code and data for each method encountered during compilation. A complicated operation, like only replacing the method code that was modified, can be performed. In order to perform such operations, the code that is to be modified should to be identified from the section where it is stored, for example, the *.text* section. Replacing the existing code will require alteration of the section size and offset. Therefore, it is necessary to rewrite and emit the ELF shared library in both approaches.

5.1.3 Symbol Resolution. Symbol resolution is one of the two activities performed by the dynamic linker, the other is relocation [21]. In the symbol resolution step, each symbol reference is associated with exactly one symbol definition. As mentioned earlier, the tables that hold dynamic symbol-related information are the global offset table, procedure linkage table, dynamic symbol table, string table and hash table. To aid the process of symbol resolution, it is indispensible for these tables to store appropriate entries. To add a symbol in the shared object file, first an entry needs to be inserted in the dynamic symbol table (*.dynsym* section). Then, a corresponding entry of the null-terminated string (name) and hash index should be created in the string and hash tables. Similarly, *.got* and *.plt* sections are required to be flooded with global variables and function calls respectively. The symbols that have entries in all these sections can be resolved and relocated effectively at the runtime. In Eclipse OMR, the method names stored in the *AOTMethodHeader* can be treated as symbols and could be stored in the string table. Whereas, the address of compiled code associated with each symbol is to be included in the dynamic symbol table.

5.1.4 Relocation and Validation. The relocations step is performed after the symbol resolution. The process of relocation is assigning a runtime address to each symbol and adjusting the code and data in the program to reflect the assigned addresses. Handling relocations is complicated as it involves several types of relocations including processor specific relocations. In Eclipse OpenJ9 and Eclipse OMR, the compiler generates the external relocations [12, 20]. The AOT relocations infrastructure creates iterated external relocations, serializes the offsets in a buffer with the compiled code offset and finally, writes them to a cache. Ideally, the generated relocation entries should be inserted into the relocation tables, so while linking they can be relocated. The dynamic linker relies on these relocation entries to modify the symbol references in the bodies of the code and data. A workaround to consume the serialized buffer entries as it stands could be to insert the buffer entries directly to a brand new arbitrary section in the ELF shared library. The downside of such a technique is that, it would not be in compliance with the ELF specification of handling relocations. Nevertheless, the underlying principle of storing relocations can still be attained and can allow sharing object files to be usable by Eclipse OMR.

5.1.5 Dynamic Overhead. While dynamic linking and loading offers several benefits, it also has its limitations. The activities involved in dynamic linking and loading can infuse performance overhead [21]. In comparison with statically linked shared libraries, dynamically linked shared libraries are easier to create and update [21]. Load time relocation and symbol resolution can incur extra costs, making the shared libraries slow. The other leading cause of performance degradation is the referencing introduced by the PIC. In order to overcome these problems, Levine [21] introduces pre-relocated libraries and caching. The concept of pre-relocating and pre-linking is adapted in embedded systems [11] to improve performance.

5.2 Portability

Portability is the usability of software or a program in different environments without requiring major reworking. The ELF object file's modus operandi of storing program-related information is similar for almost all architectures. ELF object files store control-related data in a machine independent format, making it feasible to identify and interpret the architecture-related contents in a conventional manner. With enough information to identify the target architecture, the ELF files qualify for cross-compilation and cross-linking [21]. However, the remaining code/data follows the encoding of the target processor, regardless of the machine on which the ELF object file was created [6, 26]. The PIC allows the compiled ELF shared library to be reused across multiple applications without having different copies. At the time of dynamic loading and linking, the PIC code can be relocated at any address in memory. Hence, the PIC plays a role in enhancing portability in the shared libraries. In short, the ELF file contents, like sections and metadata can be accessed on any platform, regardless of the original targeted platform. The binary code, however depends on the target instruction set architecture (ISA). Hence, portability prospects between the operating systems providing ELF support and sharing identical ISA is another viable research question.

6 CONCLUSION

There are language runtimes that implement their own versions of AOT compilation, which include a key component to store the compiled code for later reuse. The ELF file format is a common denominator for several containers used for persisting AOT generated code. The approach proposed in this paper outlines the possible extension of ELF shared libraries support in Eclipse OMR. The usage of shared libraries is based on the related work and benefits related to its dynamic and platform support. Measurement and comparison in load/store times, memory utilization for runtimes that utilize Eclipse OMR AOT module is one of the possible future work directions. The results of comparative analysis may provide additional insights into our approach and guide further explorations.

ACKNOWLEDGMENTS

This research was conducted within the Centre for Advanced Studies–Atlantic, Faculty of Computer Science, University of New Brunswick. The authors are grateful for the colleagues and facilities of CAS-Atlantic in supporting our research. The authors would like to acknowledge the funding support provided by the Atlantic Canada Opportunities Agency (ACOA) through the Atlantic Innovation Fund (AIF) program. Furthermore, we would also like to thank the New Brunswick Innovation Foundation for contributing to this project. Thanks to Stephen MacKay for his invaluable help with editing the paper.

REFERENCES

[1] John Aycock. 2003. A Brief History of Just-in-Time. *ACM Comput. Surv.* 35, 2 (June 2003), 97–113. https://doi.org/10.1145/857076.857077
[2] N. Barkakati. 2005. *Red Hat Fedora Linux Secrets* (1st ed.). John Wiley & Sons.
[3] Michael Barr and Anthony Massa. 2011. Programming Embedded Systems: With C and GNU Development Tools. (01 2011).
[4] The GNU Compiler Collection and GNU Toolchain. [n.d.]. *ELFUTILS.* Retrieved June 17, 2020 from https://sourceware.org/elfutils/
[5] The GNU Compiler Collection and GNU Toolchain. 2008. *GNU Binutils.* Retrieved June 17, 2020 from https://www.gnu.org/software/binutils/
[6] TIS Committee. 1995. *Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification.* https://refspecs.linuxfoundation.org/
[7] Eclipse OMR Community. 2020. *OMR Architecture Meeting 20200604.* Retrieved June 17, 2020 from https://www.youtube.com/watch?v=FMQ846-Ztg0
[8] IBM Corporation. 2020. *Class Data sharing.* Retrieved June 17, 2020 from https://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.vm.80.doc/docs/shrc.html#shrc
[9] IBM Corporation. 2020. *Eclipse OpenJ9 virtual machine.* Retrieved June 17, 2020 from https://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.80.doc/user/java_jvm.html
[10] JDK 10 Documentation. 2018. *Ahead-of-Time Compilation, Java HotSpot Virtual Machine Performance Enhancements.* Retrieved June 17, 2020 from https://docs.oracle.com/javase/10/vm/java-hotspot-virtual-machine-performance-enhancements.htm#JSJVM-GUID-F33D8BD0-5C4A-4CE8-8259-FD9D73C7C7C6
[11] W. Dong, C. Chen, X. Liu, J. Bu, and Yunhao Liu. 2009. Dynamic linking and loading in networked embedded systems. In *2009 IEEE 6th International Conference on Mobile Adhoc and Sensor Systems.* 554–562.
[12] Irwin D'Souza. 2018. Ahead Of Time Compilation: Relocation. Retrieved June 17, 2020 from https://blog.openj9.org/2018/10/26/ahead-of-time-compilation-relocation/
[13] Eclipse foundation. 2016. *OMR project proposal.* Retrieved June 17, 2020 from https://projects.eclipse.org/proposals/omr
[14] Eclipse foundation. 2018. *ELFGenerator, Eclipse OMR.* Retrieved June 17, 2020 from https://github.com/eclipse/omr/blob/master/doc/compiler/runtime/ELFGenerator.md
[15] W3C Community Group. 2015. *Should we use ELF as a container format?, Issue 74.* Retrieved June 17, 2020 from https://github.com/WebAssembly/design/issues/74
[16] Ludovic Henry. 2019. *AOT Compilation in HotSpot: Introduction.* Retrieved June 17, 2020 from https://devblogs.microsoft.com/java/aot-compilation-in-hotspot-introduction/
[17] Eclipse OMR IBM Corporation. 2017. *Produce an object file with Compiler/JitBuilder, Issue 1170.* Retrieved June 17, 2020 from https://github.com/eclipse/omr/issues/1170
[18] IBM Corporation, Eclipse foundation. 2019. Eclipse OMR™ Cross platform components for building reliable, high performance language runtimes. https://github.com/eclipse/omr.
[19] Vladimir Kozlov. 2018. *JEP 295: Ahead-of-Time Compilation, OpenJDK.* Retrieved June 17, 2020 from https://openjdk.java.net/jeps/295
[20] Georgiy Krylov, Gerhard W. Dueck, Kenneth B. Kent, Daryl Maier, and Irwin D'Souza. 2019. Ahead-of-time compilation in OMR: overview and first steps. In *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering, CASCON 2019, Markham, Ontario, Canada, November 4-6, 2019.* ACM, 299–304. https://dl.acm.org/doi/abs/10.5555/3370272.3370305
[21] John R. Levine. 1999. *Linkers and Loaders* (1st ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
[22] Linux Programmer's Manual. 2017. *dlopen(3).* Retrieved June 17, 2020 from https://man7.org/linux/man-pages/man3/dlopen.3.html
[23] Linux Programmer's Manual. 2019. *ld.so, ld-linux.so - dynamic linker/loader.* Retrieved June 17, 2020 from https://www.man7.org/linux/man-pages/man8/ld.so.8.html
[24] Wolfgang Mauerer. 2008. *Professional Linux Kernel Architecture* (1st ed.). Wiley Publishing, Inc.
[25] Microsoft. 2016. *Windows Subsystem for Linux Overview.* Retrieved August 10, 2020 from https://docs.microsoft.com/en-us/archive/blogs/wsl/windows-subsystem-for-linux-overview
[26] Santa Cruz Operation. 2013. *System V Application Binary Interface.* Retrieved June 17, 2020 from http://www.sco.com/developers/gabi/latest/contents.html
[27] Mono Project. 2016. *Ahead of Time Compilation (AOT), The Mono Runtime.* Retrieved June 17, 2020 from https://www.mono-project.com/docs/advanced/runtime/docs/aot/
[28] Mono Project. 2020. *AOT.* Retrieved June 17, 2020 from https://www.mono-project.com/docs/advanced/aot/
[29] Mark Thom, Gerhard W. Dueck, Kenneth B. Kent, and Daryl Maier. 2018. A survey of ahead-of-time technologies in dynamic language environments. In *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering, CASCON 2018, Markham, Ontario, Canada, October 29-31, 2018.* 275–281.
[30] Mark Thom, Kenneth B. Kent, Gerhard Dueck, and Daryl Maier. 2019. Pervasive Sharing of Language Runtimes in Eclipse OMR. Internal technical report.
[31] T. Xinyu, Z. Changyou, L. Chen, K. Aourra, and L. YuanZhang. 2017. A Code Self-Relocation Method for Embedded System. In *2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, Vol. 1. 688–691.

MicroJIT: A Case for Templated Just-in-Time Compilation in Constrained Environments

Eric Coffin
Faculty of Computer Science
University of New Brunswick
Fredericton, NB, Canada
eric.coffin@unb.ca

Scott Young
Faculty of Computer Science
University of New Brunswick
Fredericton, NB, Canada
scott.young@unb.ca

Harpreet Kaur
Faculty of Computer Science
University of New Brunswick
Fredericton, NB, Canada
harpreet.bamrah@unb.ca

Julie Brown
Faculty of Computer Science
University of New Brunswick
Fredericton, NB, Canada
julie.brown@unb.ca

Marius Pirvu
Java JIT Compiler Development
IBM Canada
Toronto, ON, Canada
mpirvu@ca.ibm.com

Kenneth B. Kent
Faculty of Computer Science
University of New Brunswick
Fredericton, NB, Canada
ken@unb.ca

ABSTRACT

Modern software libraries and applications often need to be shared across environments that do not always share common architectures. The solution to this code sharing has often been to target managed runtime environments, or High-Level Language Virtual Machines, such as the Java Virtual Machine. These runtimes are often implemented using a process called interpretation. Interpreters are significantly slower than natively executed code. One way that runtimes have addressed this problem is by including a compiler that compiles the input programs into native code at runtime. These Just-in-Time compilers can be categorized into two classes, optimizing and templated. Optimizing compilers take longer to compile and require many supporting data structures to allow for their optimizations, but usually produce (often significantly) faster code. Templated compilers always generate the same code for a given block of input. This template-generated code is faster than interpreting, but usually (often significantly) slower than the code generated by an optimizing compiler. However, these compilers do not require the heavier data structures of optimizing compilers, and produce code much more quickly. For constrained environments, such as those found in containers and Internet of Things devices, where the resources available to the managed runtime are limited, the ability to perform lightweight JIT compilation may be desirable. In this paper, we introduce a templated compiler called MicroJIT into the Eclipse OpenJ9 JVM and compare it to an interpreter-only solution in a resource constrained environment. Although our bytecode coverage is not complete, we demonstrate significant performance improvements over the interpreter for our micro-benchmarks, while at the same time, compiling with less overhead than the default JIT compiler in Eclipse OpenJ9.

CCS CONCEPTS

• **Software and its engineering → Compilers**.

CASCON'20, November 10–13, 2020, Toronto, Canada

KEYWORDS

bytecode, constrained environment, Internet of things, Java virtual machine, Just-in-Time compilation, optimization

ACM Reference Format:
Eric Coffin, Scott Young, Harpreet Kaur, Julie Brown, Marius Pirvu, and Kenneth B. Kent. 2020. MicroJIT: A Case for Templated Just-in-Time Compilation in Constrained Environments. In *Proceedings of 30th Annual International Conference on Computer Science and Software Engineering (CASCON'20).* ACM, New York, NY, USA, 10 pages.

1 INTRODUCTION

With rise of the Internet of Things (IoT), the number of embedded, connected devices has seen a significant surge in growth over the past decade [15]. For instance, farmers are now employing low-powered sensor networks to help monitor crop health, while many electrical utilities are installing smart meters to assist with managing peak demand. With the increase in IoT applications comes an increased need for software development. According to a recent survey, the most critical issues for IoT software developers are security and connectivity [24]. According to the same survey, while C remains the most popular language for developers writing code for the most constrained devices[1], Java is the preeminent language for both gateways, which connect networks of constrained devices, and for server-based applications.

While software written in C may offer the best performance with the lowest overhead, it has its drawbacks: one must manage memory explicitly, there is a lack of memory security, and portability is limited to hosts that match the target architecture and provide the necessary runtime dependencies such as a compatible C Standard library. The value of operating software correctly within heterogeneous contexts should not be underestimated: the same code may need to execute on a workstation and a server[2], on x86 and ARM platforms, and its lifetime may span decades, making it likely that the underlying platform will change.

[1] In the survey, Java was ranked number four among most popular languages used for constrained devices [24].
[2] The platform-spanning capabilities that the JVM provides can also aid in the architecture of systems: the same code could be reused between clients and servers potentially reducing development effort.

Java, along with its underlying Java Virtual Machine (JVM), has addressed these challenges: memory is automatically managed and not directly accessible—increasing both safety and security, while portability is available to any host with a compatible JVM. That said, the memory safety, security and portability provided by the JVM adds additional overhead cost to the application. Given that Java is the programming language of choice for majority of the industry [41], for many applications, it is evident that this overhead is worth paying.

Given the importance placed on security for IoT applications, the JVM could potentially provide benefits for constrained applications too. Any constrained system executing JVM-based workloads will be concerned with overhead due to many sources, such as dynamic class loading, interpretation of the application, mapping of program state to memory, initial overhead associated with the JIT compiler to compile methods into native instructions [2, 10], safety checks of certain memory access operations during runtime (such as null-reference and index out-of-bounds checks) and automatic memory management (which requires a garbage collector) [25].

In this work, we will consider environments that have the resources to run a modern, Java SE compatible JVM, yet are constrained enough to pay careful attention to these sources of overhead, particularly, to the JIT compilation. The JIT compiler in Eclipse OpenJ9—Testarossa (TRJIT)—is a highly-tunable, method-based, optimizing compiler. While TRJIT supports fast compilation, generating non-optimized code, it requires an intermediary stage where an intermediate language (IL) is generated. It is possible that the IL generation phase may introduce too much overhead for constrained environments where low-optimized code is sufficient. One approach to solve this issue could be to add a second, lighter-weight JIT compiler to the virtual machine, which could be used in place of TRJIT. Instead of generating an IL, this JIT compiler would translate bytecodes using predefined machine code templates, which would be stored and could be executed later. By eliminating the IL phase, we expect that JIT compilation will have a smaller footprint and will take less time to perform[3].

We propose adding this lightweight JIT compiler to Eclipse OpenJ9, which could be used instead of, or alongside the regular JIT compiler for constrained environments. In the spirit of the original MicroJIT [42], this compiler will be template-based with the goal of generating native methods as efficiently and quickly as possible. It should be noted that while our previous work focused on porting the original MicroJIT from IBM J9 to Eclipse OpenJ9 [4], we have since realized that this would entail a near-complete rewrite to achieve the required compatibility. With that said, we will continue using the moniker MicroJIT for this new, template-based JIT compiler. To guide the order of bytecode-template implementation, we will record the execution frequency of bytecodes in several standard benchmarks and to establish a baseline, we will measure the overhead of the interpreter operating without any JIT compiler. Then, we will measure the overhead associated with TRJIT and MicroJIT and compare the throughput of several applications while running in interpreter-only mode against the interpreter with MicroJIT. Finally, we will analyze the results to identify the window where template-based JIT begins to outperform interpretation, but where optimizing compilation overtakes template-based compilation.

This paper reads as follows. Section 2 provides an overview of the background and Section 3 discusses the related work. In Section 4, we focus on the design of MicroJIT and how we integrated it into Eclipse OpenJ9. Section 5 details our evaluation and results. Section 6 outlines the future work, and finally, Section 7 concludes the work, summarizing our findings and offering recommendations.

2 BACKGROUND

In this section we discuss the following background topics: the Java Virtual Machine, JIT compilation, Eclipse OpenJ9, and finally constrained devices.

2.1 Java Virtual Machine

In the early 2000s, type-safe, runtime-based languages such as Java and C#, rose to prominence in the realm of enterprise applications [40, 41]. The designs of enterprise applications, often with complex domain models, rules and requirements are aided by these high-level, object-oriented languages and the virtual machines that execute them. Rather than compiling directly to executable code, applications written in these languages are designed to be portable, compiling to intermediary formats designed to be interpreted on any machine providing a compatible runtime environment. Typically, the programs written in the Java language compile to class files[4] and execute on Java Virtual Machines. For an in-depth discussion of the Java language, please refer to the Java Language Specification [14].

This focus on portability popularized the phrase "write once, run anywhere" [5]. To ensure cross-platform compatibility, the JVM must adhere to the Java Virtual Machine Specification [29], which describes the program layout, linking and loading, program verification, stack machine descriptions, class file format and bytecode listing. In addition to portability, Java prevents the application from explicitly managing or interacting with the underlying memory. While this limitation may preclude developers from choosing Java to solve lower-level tasks, it works well for high-level applications. By removing explicit memory management, something that takes increased care as application complexity grows, developer productivity can improve and the chances of inadvertently introducing memory errors decrease.

A runtime that implements the Java Virtual Machine Specification and provides the required Application Program Interfaces (APIs) should be able to run any compatible Java application. For maximum performance, these virtual machines are typically written using system-level languages, such as C, C++ and assembly. The underlying host supports these virtual machines: process controls, virtual memory, concurrency, exceptions and I/O must map from the JVM to the underlying operating system (OS) and, in turn, to the underlying Instruction Set Architecture (ISA) [39]. With this view in mind, interactions are made from the application to the underlying OS through the Java API libraries, such as when calling `System.out.println` and having ASCII characters print to the command-line. Alternatively, interactions between the JVM and the underlying OS can be made through the Application Binary

[3] In terms of throughput or work performed during a period of time, we expect that the code generated by the lightweight compiler will be faster than the interpreter, but an order of magnitude slower than the code generated by TRJIT.

[4] Java programs are compiled using the Java compiler or `javac`.

Interface (ABI), such as when making a system call such as *fork* on Linux to spawn a new thread.

Some examples of production-grade JVMs include Eclipse OpenJ9 [33], Oracle's HotSpot [8] and Azul Zing [22]. It is worth noting that the JVM has become an essential platform for languages beyond Java. Languages such as Scala, Kotlin and Clojure are all popular languages that compile to JVM compatible meta-data and bytecode. Class files, the format of which is defined by the JVM specification, contain the program structure, behaviour, data structures and various metadata required by the JVM for the program to execute. The executable code, contained in parameterized functional units called *methods*, is defined by the series of bytecode instructions. Each bytecode is one or more bytes in length, with the first byte containing the instruction opcode and the following bytes containing the instruction arguments. Data types include primitives such as integers, doubles and booleans, as well as object reference types. The JVM provides a stack-based machine for the application to operate within. Each application thread has a stack, which in turn contains stack frames. Each stack frame contains an operand stack, a program counter and a local storage array. The exact size or shape of the stack frame is described in the meta-data contained in the class file. Operations described by the bytecodes affect the operand stack, which maintains the state of the method.

During execution, an application can store values in two places: in the current stack frame, or on the heap. For values of known sizes, such as primitives, the stack is an ideal location for allocation as the memory will be automatically released when the stack frame is popped. Any values that are dynamic in size, or non-local, will be allocated on the program heap. As this memory cannot be explicitly freed, over time, as the program continues to execute, the heap will continue to fill. Garbage collection is the process by which the objects in the heap that are no longer referenced are freed [25]. For a collector to be viable, it must satisfy the requirement that all garbage will eventually be "collected".

2.2 Just-in-Time Compilation

Although Java applications were at one time entirely interpreted, advances in Just-in-Time (JIT) compilation have improved performance by orders of magnitude, allowing them to be much closer to that of their Ahead-of-Time (AOT) compiled counterparts written in C or C++ [26]. In addition to portability and safety, the high-performance garbage collectors and JIT compilers found in production-grade JVMs have helped place Java at the top of languages used for server-based workloads [41].

According to Rau [36], application binaries can be divided into three groups: those composed of directly executable machine instructions, those composed of higher-level instructions that must be parsed and interpreted before execution and those composed of directly interpretable instructions. While the latter two groups offer opportunities for portability and memory safety, the performance penalty incurred by interpretation may preclude them from specific workloads. This work will focus on the final group, i.e., programs composed of directly interpretable instructions.

While interpreters should be designed to be efficient—minimizing the number of indirect branches [11, 39]—at their simplest, they involve a cycle of fetching a single instruction, decoding that instruction, and then dispatching the instruction for execution using native instructions [39]. Even when the same instruction is executed repeatedly, this same cycle is performed. One way to improve the performance of such workloads is to add a runtime compiler to the execution framework. By compiling, or translating blocks of instructions into native instructions, and then later executing those generated blocks instead of interpreting them, significant performance gains may be achieved. This process, known as Just-in-Time compilation, is often combined with profiling and/or analysis to optimize the generated code [2, 39].

Considering JIT compilation as an expensive activity, in order to allow for the continued execution of the workload, Just-in-Time compilation is often performed selectively on only the most frequently executed blocks of code [28]. As described by Hansen in his work on Adaptive Fortran [17], one way to perform selective compilation is to associate with each code block a counter containing an invocation threshold value. Each time the code block is executed, the counter is decremented. When the counter reaches zero, the block can be compiled. For a block to be JIT-compiled means that for a source block X, there is a generated block of machine instructions, Y, that lives in a code storage area called the code cache [18, 39]. During execution, when the execution engine encounters a call to block X, it performs a check to see if Y exists in the code cache, and if it does, the interpreter transitions to the native code instead. This invocation threshold should be made tuneable for the user, as setting a low threshold will result in more JIT compilation during the startup of the application, which may be undesirable for some workloads. Within the context of Java, we will use entire methods as our code block or unit of compilation (see Figure 1).

JIT compilation can be viewed on a continuum: on one end, there is unoptimized code that is fast to generate but slow to execute. In contrast, on the other end, there is optimized code that is slow to generate but fast to execute. To lessen the impact on the startup time of an application, one could perform initial compilation with little or no optimization, that is, generating slow code quickly. As a code block continues to execute, higher thresholds may be reached, potentially triggering more expensive, optimized recompilations [20]. For JIT compilers with multiple optimization levels, it may prove useful to allow a user to specify heuristics per optimization level.

In order to aid in performing optimizations, it is typical for a JIT compiler to build an intermediate representation of the code to analyze and transform. Some of the optimizations we see for JVM-based optimizing JIT compilers are similar to those commonly found in AOT compilers [7, 27, 39], such as dead-code elimination, constant propagation, constant folding, strength reduction, code hoisting, loop unrolling, inline substitution and peephole optimizations.

2.3 Eclipse OpenJ9

This work will expand upon the JVM implementation of Eclipse OpenJ9 for Java 8 [33]. In particular, we will focus on its implementation for Linux on the x86-64 architecture. This open-source project, which implements the OpenJDK specification and is available under the AdoptOpenJDK project [1], is designed for low overhead, quick startup and high throughput. Also, OpenJ9, which originated from

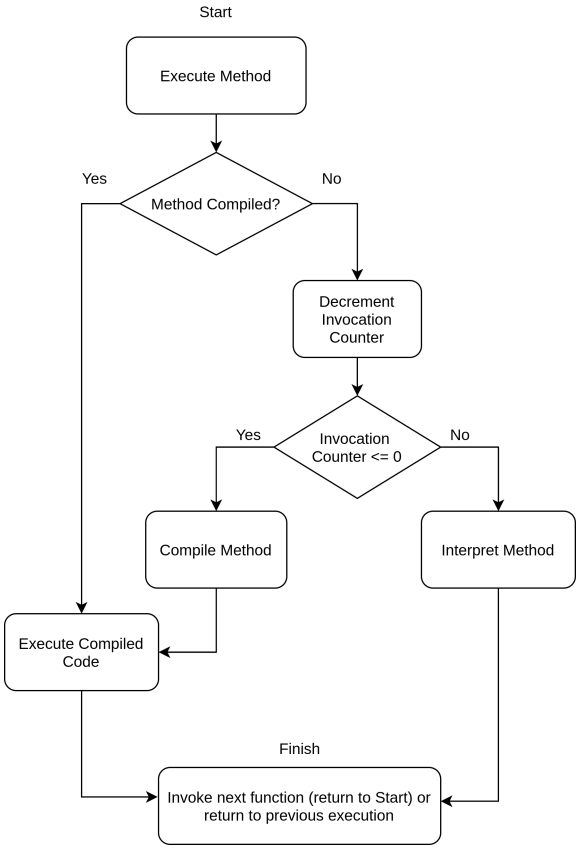


Figure 1: When a method is to be executed, the execution engine first checks if the method has already been JIT-compiled. In this example, compilation and, in turn, execution occurs synchronously, potentially occurring on the currently executing application thread.

IBM's J9 JVM, is built upon the open-source runtime component framework named Eclipse OMR. The OMR framework allows an existing runtime to integrate the production-grade components such as garbage collection, JIT compiler, an API for quickly utilizing the JIT compiler, named JitBuilder, cross-platform support for threads and signals, as well as threading support [13, 30]. Much of the infrastructure driving Eclipse OpenJ9 links to OMR components.

The JIT compiler found in this project, and thus in OpenJ9, is named Testarossa (TR) [43]. In TR, the compilation is method-based and triggered using invocation thresholds for regular methods as well as methods containing loops. TR supports several levels of optimization ranging from the initial optimization level *cold*, with roughly 20 optimizations applied, through *warm*, then *hot* and *very-hot* levels, all the way to the highest level—*scorching*, where as many as 170 optimizations can be applied [34, 37]. There is another compilation level named *noOpt*, which applies no optimizations and may be used to improve application startup time in large applications [34].

Command-line arguments for TR can be passed to the JVM with the `-Xjit` option group [32]. Some options significant to this work are listed as follows:

- `count` - The invocation threshold for standard methods. The default value is 3000.
- `bcLimit` - The bytecode limit size for methods to compile in bytes.
- `codetotal` - Available memory for generated code in KB.
- `limit` - A debugging option to list the methods that TR should include and at what compilation level it should compile them.
- `exclude` - A debugging option to list the methods that TR should exclude.
- `breakOnEntry` - A debugging option generating a breakpoint instruction (`int3` on x86-64) at start of the generated code.

During compilation, fewer resources are available on the platform for executing workload. For certain constrained environments, the overhead of the TR JIT compilation may be too high to be effective. In this work, we will add a second, lighter-weight JIT compiler, MicroJIT, to Eclipse OpenJ9.

2.4 Constrained Devices

In this work, we consider *constrained devices* to be any computing device that has limited processing capability compared to other devices performing work. We use the term *constrained environment* to denote a context within which work is done on a constrained device. Thus, this term can encapsulate embedded systems, connected IoT devices, and even containers that are running in the cloud. This limited capability may be the result of several things: having limited CPU cores and processing speed, having limited memory, maintaining optimal power efficiency, having limited I/O bandwidth, or having limited connectivity [19]. The type of workload performed on the device may also be indicative of a constrained environment.

As the market for IoT continues to increase, the number of these constrained devices is growing substantially. Indeed, IoT devices are being installed for nearly every conceivable scenario where data can help inform decisions. With this increase in devices, comes the increased need for software and thus software developers. Meeting this demand in a timely fashion with secure and robust software will be a significant challenge for the industry. Revisiting the Eclipse IoT developer survey [24] from 2019, we see that although C is the most popular language for most of the constrained devices, Java is in the top five. Furthermore, developers use Java over any other language for less constrained devices—gateways and servers. With the benefits of Java and the JVM in mind, it is conceivable that interest in using them for running workloads on constrained devices will continue to increase. For many of these workloads, performance will be a crucial factor, and as such, JIT compilation will play an essential role.

3 RELATED WORK

Different solutions have been proposed in the past in this direction. One such work was the experimental support for WebAssembly—a new runtime and compilation target for the web—in V8 [47]. This implementation used the TurboFan compiler—V8's powerful

optimizing compiler—and much of the existing JavaScript virtual machine infrastructure. After roughly a year, the V8 team launched a new JavaScript execution pipeline for V8 v5.9 [45], which led to improvements in performance and memory consumption of JavaScript applications. This pipeline used Ignition for interpretation and TurboFan for compilation. Liftoff is a baseline compiler for WebAssembly, which is included in V8 v6.9 and now enabled by default on desktop systems [16]. It reduces the startup times of WebAssembly applications significantly by adding another compilation tier. The new compilation pipeline with Liftoff is much simpler compared to the existing compilation pipeline with TurboFan. Even if the existing compilation process is straightforward, it still consumes considerable time and memory. So, Liftoff is able to generate code much faster than TurboFan.

SpiderMonkey—Mozilla's JavaScript runtime—uses IonMonkey for JIT optimization [6]. It applies various strategies to optimize operations, such as property accesses and function calls. Shudo et al. [38] developed a Java Just-in-Time compiler involving low compilation and development costs. The optimization methods implemented in the compiler included instruction folding, exception handling with signals and code patching. Another work by Iliasov [21] demonstrated that dynamic code generation from templates created using a C compiler can be employed to build a simple, portable JIT compiler. At the same time, a template-based compiler requires more memory than an interpreter and also, it implies certain limitations on the instruction sets.

Sogaro et al. [42] investigated whether using two different JIT compilers in the same JVM can improve startup times. They concluded that integrating MicroJIT—a lightweight JIT system—with the default J9 JIT, could improve startup times in some configurations of the JVM. Later, another work was carried out by Coffin et al. [4] by incorporating MicroJIT into OpenJ9 to offer similar improvements to startup times, while offering applications in resource constrained environments a lightweight JIT alternative. Some changes were proposed to the earlier implementation of MicroJIT to improve its flexibility and performance, such as porting MicroJIT to the 64-bit x86 architecture, extending bytecode support, added support for asynchronous compilation and profiling to the generated code.

In our current work, an effort has been made to improve the throughput of certain workloads in constrained environments running with MicroJIT versus running in interpreter-only mode. While trying to port the project, we realized that it could involve more effort and risk than rewriting the compiler from scratch mainly because the mechanism in the previous MicroJIT for returning to the interpreter when an unsupported bytecode was encountered would not work in Eclipse OpenJ9 and also, the shape of the stackframe the original MicroJIT maintained was not compatible with Eclipse OpenJ9. While these issues signalled that a rewrite of the previous MicroJIT was necessary, we did borrow several aspects from its design such as continuing to use methods as the unit of compilation for MicroJIT, iterating or walking the bytecodes, copying a snippet of machine code to the code cache for each bytecode it encountered while generating machine code for the body of a method, and use of `extra2` field similar to the previous MicroJIT.

An alternative approach to reduce JVM overhead from JIT compilation is to utilize AOT-compiled code. Considering that the Java Virtual Machine specification states that linking and loading must happen dynamically [14], the quality of AOT-compiled code will be limited as the compiler must assume that all references are unresolved. The quality of AOT-compiled code can be improved by storing dynamically generated code from the JIT compiler and metadata in an offline store, or cache. This cache can improve startup time and performance over the interpreter, potentially leading to less reliance on the JIT compiler for some workloads. OpenJ9 offers this improvement through the use of the shared class cache, a memory-mapped file, enabled automatically through the use of the `-Xshareclasses` command-line option. The cache can be configured to be persistent and shared between multiple instances of the JVM [9]. For long-lived environments capable of persisting runtime generated code between executions of the Java application, this approach provides a very efficient mechanism for reducing startup time [35]. On the other hand, for short-lived environments, such as those running ephemeral containers where applications will only execute once, this approach may not easily offer a viable solution [46], although options are available to pre-warm an image during the building of a Docker Image [23].

4 DESIGN

MicroJIT is a template-based JIT compiler for Eclipse OpenJ9 designed to generate code quickly with low overhead. Using methods as its unit of compilation, MicroJIT translates the bytecodes of a Java method-body into a block of generated binary instructions stored for later execution in a code cache. The compilation is performed selectively through the use of invocation counters. Once the number of times a method has been interpreted reaches a specified threshold, the method will be passed to MicroJIT for compilation. If the method is able to be compiled by MicroJIT, later, when the method is invoked again by the interpreter, execution will instead transition to the JITed code.

4.1 Integrating MicroJIT and Eclipse OpenJ9

While MicroJIT may be viewed as an independent module, it is closely integrated with Testarossa (TR). This coupling allows us to take advantage of numerous facilities provided by TR, including:

- **Code Cache Management**: The code cache management facility provides us with a convenient, cross-platform mechanism for allocating and managing executable memory. While we could have directly used the system call `mmap` to provide our process with an executable memory space to copy our templates, this would have limited us to the Linux platform. By leveraging the Code Cache Manager API, MicroJIT can interact indirectly with the memory systems of other operating systems as well.
- **Asynchronous Compilation**: Rather than compiling directly in the same thread as the executing method, which would delay the execution of the workload—especially for large methods—a method is placed on a queue for later compilation, allowing the interpreter to continue interpreting the method. The compilation queue allows increased parallelism, as we can now perform compilation on multiple threads and also to prioritize some methods over others. By utilizing the existing compilation process up to but not including the IL-phase, we gain support for asynchronous compilation.

- **Debugging Utilities**: TR provides command-line options to limit compilation to particular methods. TR's ability to log the generated code in a human-readable format was also valuable when designing the MicroJIT code-generator. By sharing much of the pre-compilation and post-compilation code paths with TR, MicroJIT benefits from these debugging and tracing facilities.
- **Bytecode Iterator**: The Bytecode Iterator class, which implements the iterator pattern [12], provides a convenient mechanism for iterating through a method's bytecode stream. The class saved us the effort of parsing Java bytecode instructions and operands while providing us with their associated mnemonics for both programming and debugging[5].
- **Exceptions**: Finally, the close integration of MicroJIT with TR provides us with cross-platform support for interrupt handling. Through the Port library, provided through Eclipse OMR, signal handlers are registered for JITed code. In the event our JITed instructions generate an exception, for example, when attempting to divide by zero, the registered handler will receive the signal, and then begin the process of unwinding the stack from the JITed frame to find the appropriate Java exception handler.

In order to enable and configure MicroJIT within Eclipse OpenJ9, we added the following command-line options to the `-Xjit` top-level option:

- `mjitEnabled` - Set to 1 to enable MicroJIT.
- `mjitCount` - The number of invocations a method requires before triggering a compilation.

4.2 Architecture

In the following sections we discuss how we select and generate code, as well as considerations for our target platform—Linux on x86-64.

4.2.1 Selective Compilation. The pre-compilation phase begins when the interpreter executes a method that has not been JIT-compiled. The invocation counters are adjusted, thresholds are checked and compilation is triggered when the counters have reached zero. MicroJIT adds a second field, `extra2`, to each method metadata structure `J9Method`. This field performs a role similar to the `extra` field, which is used by TR to store the invocation count, the address of the JIT-compiled method, or special values indicating that compilation should not be attempted again. When compilation is triggered, and asynchronous compilation is enabled, the method is placed in the compilation queue. The compilation thread then dequeues a pending method compilation and continues the pre-compilation phase during which metadata structures are populated, and various compiler options are initialized. Once the pre-compilation completes, the compilation proceeds to MicroJIT.

4.2.2 Code Generation. When MicroJIT compilation begins, a 1024-byte segment of memory is first allocated through the code cache manager. Currently, if the number of generated bytes exceeds this, the compilation will fail, and the segment will be freed through the code cache manager. Next, we inspect the incoming parameters to generate code for populating the stack frame and the local storage area, and for ensuring later root-set compatibility with garbage collection. While the slots in our operand stack are each eight bytes, i.e., large enough to satisfy any primitive datatype we will encounter, for certain `wide` operations involving floats and doubles, we use two slots. Likewise, following the Java specification, for a local array, we allocate two slots for both float and double types in order to simplify compatibility.

The initial machine code generated by MicroJIT contains the preprologue followed by the prologue. The preprologue contains code to check for stack overflow as well as to potentially relinquish control to the interpreter to perform any necessary JVM processes. The prologue contains code to push the previous base pointer and the preserved registers onto the stack and then sets up the new base and stack pointers. Instructions are then generated to copy the incoming parameters to the local array, after which the next step is to generate the body of the method.

While iterating the method bytecodes, if an unsupported bytecode is encountered, the compilation is aborted, the code cache memory is freed, and a value is stored in the `extra2` field to signal the method failed and should not be compiled by MicroJIT again. On the other hand, if all the bytecodes of the method body were supported, the code buffer will now contain the machine code templates for each bytecode instruction. Branching is supported through the use of two structures: a table mapping each bytecode index to the generated code address, and a jump table containing an entry for each branch instruction. Each jump table entry contains the branch or jump's target bytecode index and the address in the generated code to target with a patch operation after the template is copied. After the bytecodes have been iterated, we iterate each jump table entry finding the generated address for the target bytecode in the bytecode index table and then patching the generated branch or jump address with it. After the body has been generated and patched, the epilogue instructions are generated, which clean up the stack and restore the preserved registers. Finally, the address of the preprologue is written to the `extra2` field within the method metadata structure. Later, when the method is invoked by the interpreter, this field is checked, and if it contains a valid program counter address, execution then transitions to the JITed code.

4.2.3 Platform. Our first target platform for MicroJIT is Linux running on the 64-bit x86 architecture. The choice of this platform stems from our previous work attempting to port MicroJIT to Linux from Windows while extending its instructions from 32-bit to 64-bit. With the choice to rewrite the compiler, we maintained the target platform. Values are passed to and from the generated code via registers according to the following calling convention defined by Eclipse OpenJ9:

- `RAX, RSI, RDX, RCX` - Argument registers for the first four integer method parameters where the first argument is found in `RAX`. Other parameters will be found in the caller stack frame.
- `XMM0-XMM7` - Floating point method parameters.

[5]The iterator pattern is a design pattern for iterating over a collection of elements. The methods provided are `first()`, to get the first element, `next()`, to get the next element of the iterator, and `hasNext()`, returning a boolean value if there is another element to iterate over.

```
template_start iAddTemplate
mov r11, [r10] ; pop first value off java stack
add r10, 8 ; reduce stack size by 1 slot
mov r12, [r10] ; copy second value to the value reg
add r11, r12 ; add the values
mov [r10], r11 ; write the accumulator to the stack
template_end iAddTemplate
```

Listing 1: Bytecode x86-64 template for the `iadd` bytecode.

- `EAX, RAX, XMM0` - Return value registers; `EAX` is used for 32-bit integers, `RAX` for 64-bit integers and `XMM0` for float or double values.
- The return address will be already on the stack. After the epilogue executes, this value will be used by the action to return to the previous frame.

Within the generated code, similar to the JVM specification, we use a memory-based operand stack for state, and a local array for storage. We also provide a side-table for internal mapping between the bytecode and generated code, which is used for patching branch and jump instructions. We use the following x86-64 registers within the generated code:

- `RSP`: Base pointer for the Java stack frame
- `R10`: Stack pointer for the Java stack
- `R11`: Stores the accumulator or stores a pointer to an object
- `R12`: Stores any value which will act on the accumulator, stores the value to be written to an object field or stores the value read from an object field
- `R13`: Holds addresses for absolute addressing; used when loading references or fields
- `R14`: Holds a pointer to the start of the local array
- `R15`: Stores values loaded from memory for storing on the stack

4.3 Bytecodes

The majority of the bytecodes we implemented map to a unique template written in NASM-style assembler, although several bytecodes, notably `load`, `store` and `ret` were handled generically. Listing 1 shows the assembler for the `iadd` template. We use the `template_start` and `template_end` macros to simplify calculating the size of the template. During code-generation, as we iterate through a method's bytecode stream, we simply copy these templates into the allocated code cache segment. For those bytecodes that require index-based addressing, we write placeholder bytes and patch them after the body has been completely generated. With the assistance of side-tables, a similar mechanism is used for branching as well as for the `goto` instruction. The operands of these bytecodes specify signed offsets from the current bytecode, though we translate them into indexes from 0.

4.3.1 Implementation Strategy. At the time of writing, MicroJIT does not provide full bytecode coverage. As mentioned, when we encounter an unsupported bytecode, we prevent further attempts at compiling the method with MicroJIT. While we have enough bytecodes implemented to compile the Fibonacci programs described in the Results section, MicroJIT lacks support for many important bytecodes including `invokevirtual`, `new`, `invokespecial`, `newarray` and `athrow`. To guide our implementation, and to track our progress, we test our bytecode coverage, against the DaCapo benchmark suite [3] (see Table 1 for results with `avrora`). Across all the benchmarks, `invokevirtual` and `invokespecial` are the most common unsupported bytecodes.

Bytecode	Count	% of Unsupported	% of Total
JBinvokevirtual	2518	18.53	5.25
JBputfield	1647	12.12	3.44
JBinvokespecial	1530	11.26	3.19
JBifeq	773	5.69	1.61
JBnewdup	632	4.65	1.32
JBldc	575	4.23	1.20
JBaconstnull	366	2.69	0.76
JBathrow	351	2.58	0.73
JBifnull	338	2.48	0.70
JBnop	267	1.96	0.55
Total Bytecodes	47,877		
Unsupported	13,585		

Table 1: The bytecode frequency for DaCapo is used to help guide implementation choices. Listed are the 10 most used unsupported bytecodes for `avrora` with `mjitCount=20`.

4.3.2 Testing. To aid in the development of MicroJIT, we designed a regression test framework. This framework allows us to perform assertions on the results of Java test methods while at the same time confirming that they were compiled by MicroJIT. These tests are then executed with the aid of the JUnit framework [44] on Eclipse OpenJ9. We ensure our test methods are executed by providing the same invocation threshold to both the test framework and to the JVM. When the tests have completed, we have the expected results from JUnit, and we scan the compiler log for special messages signalling that compilation was completed for each particular method. A failure could thus be caused by one of two things: either a method was not compiled by MicroJIT, which would indicate a failure within the code generator, or the result was incorrect, indicating an issue with a bytecode template. The test framework is designed to execute automatically on a continuous integration server before a branch is merged into the mainline, or trunk.

5 RESULTS

We tested the performance of MicroJIT with bespoke micro-benchmarks that calculated the Fibonacci series iteratively and recursively using static methods, for which we provided full bytecode coverage. `IterativeFib.fib(I)I`, which compiles to 37 bytecodes, computes the series in a single method call using a for loop and a temporary variable, while `RecursiveFib.fib(I)I` compiles to 21 bytecodes, and computes through repeated calls to itself. The micro-benchmarks were run on an embedded board with an Intel Atom 1.44 GHZ 4-core processor, 4GB of RAM, running Debian 5.6.7-1.

First we will look at the time spent compiling the methods with MicroJIT, and with TR with the compilation levels of `noOpt` (TR-noOpt) and `cold` (TR-cold) respectively. The times, shown in microseconds, are based on 200 fresh executions of the Java test programs for each of the three compiler settings, with the 20 lowest and

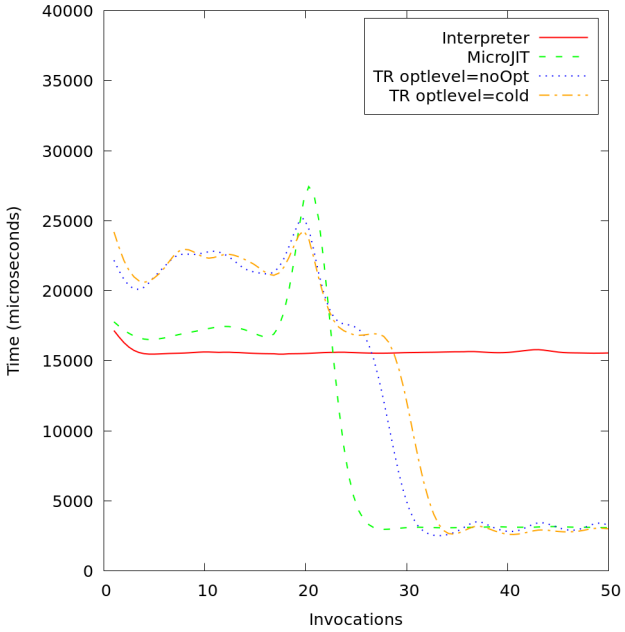
20 highest results discarded to help eliminate outliers. Also, using the `limitFile` option, each benchmark was limited to compiling the single bespoke method. Table 2 shows that by eliminating the IL phase, MicroJIT was able to compile the methods roughly 2.3 times faster than TR-noOpt, with `RecursiveFib` almost 5 times faster than TR-cold level. Considering that we are copying machine code templates instead of building and transforming an intermediate form, this is the expected result. Table 3 shows the memory in KB used by JIT compilers to compile the methods in the same experiment[6]. We see that MicroJIT uses a fraction of the memory that either TR-noOpt or TR-cold use: it is able to compile the micro-benchmark methods with just a single 64 KB segment. One result that stands out is TR-noOpt requiring more memory than TR-cold for `IterativeFib`, but less for `RecursiveFib`. Looking at the optimization logs from TR, we see that the generated code size is 147 bytes for TR-noOpt and 141 bytes for TR-cold. We also see that in the post-optimization IL-trees, TR-noOpt, which had only a single tree-simplification optimization, has 44 nodes, while TR-cold, which had several optimizations performed, has 38 nodes. Comparing this to the `RecursiveFib` results, we see that the generated code size is 103 bytes for TR-noOpt and 265 bytes for TR-cold. Also, looking at the post-optimization IL-trees, we see that TR-noOpt has 23 nodes, while TR-cold, which had several optimizations performed, including inlining, has 103 nodes. One possible explanation for the larger footprint when compiling `IterativeFib` at the `noOpt` level is the overhead associated with the larger IL-trees.

Figure 2a shows the execution time of calculating the 30th Fibonacci number using `IterativeFib` from 100 fresh executions of the Java program. With each compiler using 20 as its invocation threshold, we see that MicroJIT completes its compilation first and thus sees the earliest improvement in throughput. Figure 2b shows similar results for the recursive method, however we note that the invocation count is reached earlier due to recursion. In both figures, we see that the baseline performance of interpreter remains fairly static. The interpreter runs were executed in interpreter-only mode without TR (`-Xint`), meaning that no profiling data was gathered. On the other hand, for the TR-specific runs—TR-noOpt and TR-cold—the interpreter did collect profile information, which is then used later to inform the JIT compilations.

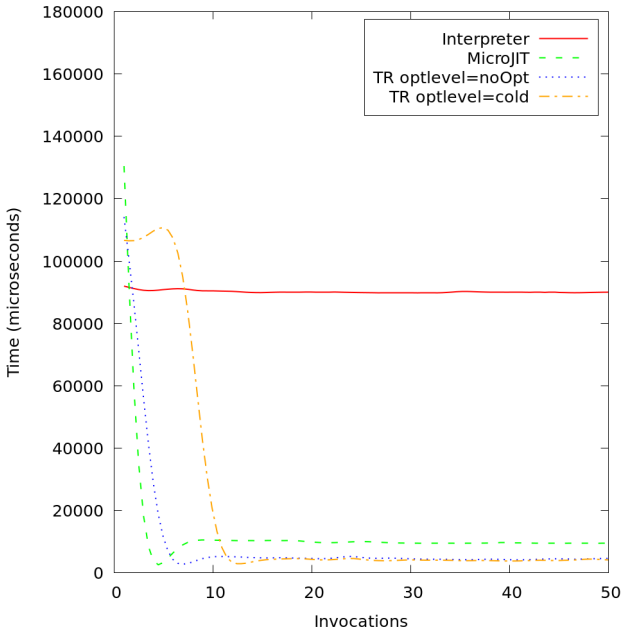
Finally, Tables 4 and 5 show the throughput of over 1 million invocations for `IterativeFib` and `RecursiveFib` respectively. The experiment was run 100 times. The % column shows the percentage of time spent relative to the interpreter. In Table 4, we see that MicroJIT is able to complete 1 million calculations in 1.38 seconds, or 9.55% of the time it took for the interpreter. As expected both TR-noOpt and TR-cold are able to perform the same task significantly faster, with TR-noOpt completing 1 million calculations in 0.897 seconds. It may be worth noting that, despite its name, TR-noOpt performs tree simplification and the code generation may include low-level optimizations.

In Table 5, we see that more computational effort is required to calculate the Fibonacci sequence recursively, with the interpreter requiring 88.55 seconds to complete 1 million calculations of the 10th number. We can also see that significant improvement in throughput can be achieved through TR and its IL phase. While MicroJIT is

[6]The memory values did not change across the repeated executions.



(a) `IterativeFib` calculating 30th number in sequence, i.e., fib(30)



(b) `RecursiveFib` calculating 10th number in sequence, i.e., fib(10)

Figure 2: Comparison of Execution Times (in microseconds) of first 50 iterations of `IterativeFib` and `RecursiveFib` with invocation threshold 20

able to complete the task in 8.69% of the time that was required by the interpreter, TR-noOpt required 2.67% of the time and TR-cold required just 2.10% of the time. This performance discrepancy can be attributed to optimizations of `invokestatic` in TR's generated

	MicroJIT			TR-noOpt			TR-cold		
	mean	median	std.dev	mean	median	std.dev	mean	median	std.dev
`IterativeFib`	1025.44	1253.00	356.84	2446.13	2498.00	258.99	3100.34	3132.50	239.79
`RecursiveFib`	1016.49	1266.00	368.59	2376.59	2432.00	247.69	4976.59	4980.00	271.50

Table 2: JIT Compilation Time (in microseconds)

	MicroJIT	TR-noOpt	TR-cold
`IterativeFib`	64	1024	960
`RecursiveFib`	64	960	1280

Table 3: Memory (in kilobytes) for JIT Compilation

	Time (in seconds)			Operations per second	
	mean	std.dev	%	mean	std.dev
Interpreter	14.426	0.0324	100.00	69,315.85	153.76
MicroJIT	1.378	0.0047	9.55	725,672.55	2428.24
TR-noOpt	0.897	0.0039	6.22	1,114,194.82	4849.61
TR-cold	0.847	0.0045	5.87	1,180,377.12	6299.74

Table 4: Time to execute 1 million iterative invocations of fib(30)

	Time (in seconds)			Operations per second	
	mean	std.dev	%	mean	std.dev
Interpreter	88.55	0.105	100.00	11,292.26	13.33
MicroJIT	7.70	0.320	8.69	129,925.61	4688.08
TR-noOpt	2.37	0.008	2.67	420,178.73	1534.97
TR-cold	1.86	0.014	2.10	536,712.94	4257.15

Table 5: Time to execute 1 million recursive invocations of fib(10)

code: while the code generated by TR has the `call` instruction addresses patched directly to the JITed code, the `call` instructions in MicroJIT first jump to an intermediary routine to check if the callee has been JITed by TR, by MicroJIT, or not compiled at all (control returns to the interpreter). This additional step is currently required to maintain interoperability between MicroJIT and TR, though in the future, a MicroJIT-only build may eliminate it. Looking at the quality of the generated code, TR-noOpt has far fewer instructions than MicroJIT (28 versus 115), and for `IterativeFib`, the numbers are similar (36 versus 116)[7]. While we present a large improvement over the interpreter, we will continue to look for inexpensive operations we can apply to reduce the number of instructions generated, as well as to reduce the overhead when calling.

6 FUTURE WORK

MicroJIT is currently a limited solution, which is missing support for several key bytecodes. Our top priority is improving bytecode coverage, so we can support more micro-benchmarks, eventually transitioning to majority method support in established benchmarks, such as DaCapo. We have been making steady progress towards this goal and expect to reach full, or near-full coverage within the next several months.

TR compiles methods with varying levels of optimizations. Some of these levels, particularly those at the highest level of optimization, rely on profiling data gathered during interpretation. By compiling code so early with MicroJIT, this data set becomes much smaller and therefore less representative. One of our future research objectives is to emit profiling instructions into MicroJITed code and see if a templated JIT can create equivalent profiling data sets faster than the interpreter, without negatively impacting the performance of TR as an optimizing compiler. These penalties for optimizations are a real cost in terms of development time, processing power and memory usage; and not having them is one of the reasons why MicroJIT compiles so fast. However, some compiler optimizations are fairly inexpensive to implement. For example, a peephole optimization can be used to eliminate unnecessary loads [31], drastically improving performance, but can be done with a single look ahead operation for every store instruction in the source. Some future work can be focused on this field of optimizations, like what cheap optimizations can be added to a templated code generator or can we actually make those optimizations cheaper?

As a replacement for TR in resource constrained environments, MicroJIT needs to support the architectures used most commonly in that field. Today, those architectures are the AArch32 and AArch64 architectures. As a tool to facilitate faster warm-up times for TR, MicroJIT needs to support the architectures that run server-based Java applications. Those architectures include the x86-64 Architecture, the 64-bit PowerPC Architecture and the z/Architecture. As MicroJIT evolves, we want to extend its support to cover all these architectures and make it a feature available for all Eclipse OpenJ9 platforms.

7 SUMMARY

For some constrained workloads limited by memory and/or processor power, a lightweight JIT compiler may be the ideal mechanism to achieve performance improvements at runtime. This paper presents our work of adding a template-based JIT compiler, MicroJIT, to Eclipse OpenJ9 towards that end. While our bytecode coverage is thus far limited, the compilation results are promising; for our micro-benchmarks, we demonstrate significant improvements in performance over the interpreter, while at the same time, compiling more quickly and with less memory overhead than the default JIT compiler in Eclipse OpenJ9.

[7]The difference in code-size can be attributed to our reliance on an in-memory operand stack, as well as to our operations for register preservation.

ACKNOWLEDGMENTS

This research was conducted within the Centre for Advanced Studies—Atlantic, Faculty of Computer Science, University of New Brunswick. The authors are grateful for the colleagues and facilities of CAS Atlantic in supporting our research. The authors would like to acknowledge the funding support provided by the Atlantic Canada Opportunities Agency (ACOA) through the Atlantic Innovation Fund (AIF) program. Furthermore, we would also like to thank the New Brunswick Innovation Foundation for contributing to this project.

REFERENCES

[1] AdoptOpenJDK. 2020. *Prebuilt OpenJDK Binaries for Free!* Retrieved 2020-02-27 from https://adoptopenjdk.net/

[2] John Aycock. 2003. A Brief History of Just-in-Time. *Comput. Surveys* 35, 2 (June 2003), 97–113. https://doi.org/10.1145/857076.857077

[3] Stephen M. Blackburn, Robin Garner, Chris Hoffmann, Asjad M. Khang, Kathryn S. McKinley, Rotem Bentzur, Amer Diwan, Daniel Feinberg, Daniel Frampton, Samuel Z. Guyer, Martin Hirzel, Antony Hosking, Maria Jump, Han Lee, J. Eliot B. Moss, Aashish Phansalkar, Darko Stefanović, Thomas VanDrunen, Daniel von Dincklage, and Ben Wiedermann. 2006. The DaCapo Benchmarks: Java Benchmarking Development and Analysis. *ACM SIGPLAN Notices* 41, 10 (Oct. 2006), 169–190. https://doi.org/10.1145/1167515.1167488

[4] Eric Coffin, Scott Young, Kenneth B. Kent, and Marius Pirvu. 2019. A Roadmap for Extending MicroJIT: a Lightweight Just-in-Time Compiler for Decreasing Startup Time. In *Proceedings of 29th Annual International Conference on Computer Science and Software Engineering (CASCON '19)*. IBM Corp., Riverton, NJ, USA, 293–298.

[5] ComputerWeekly.com. 2002. *Write Once, Run Anywhere?* Retrieved 2019-06-17 from https://www.computerweekly.com/feature/Write-once-run-anywhere/

[6] MDN Contributors. 2020. *JIT Optimization Strategies.* Retrieved 2020-06-12 from https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/JIT_Optimization_Strategies

[7] Keith D. Cooper and Linda Torczon. 2011. *Engineering a Compiler* (2nd. ed.). Elsevier.

[8] Oracle Corporation. 2020. *Java SE HotSpot at a Glance.* Retrieved 2020-03-30 from https://www.oracle.com/technetwork/java/javase/tech/index-jsp-136373.html

[9] Ben Corrie and Hang Shao. 2018. *Class Sharing in Eclipse OpenJ9.* Retrieved 2020-04-19 from https://developer.ibm.com/tutorials/j-class-sharing-openj9

[10] L. Peter Deutsch and Allan M. Schiffman. 1984. Efficient Implementation of the Smalltalk-80 System. In *Proceedings of the 11th ACM SIGACT-SIGPLAN symposium on Principles of programming languages (POPL '84)*. Association for Computing Machinery, New York, NY, USA, 297–302. https://doi.org/10.1145/800017.800542

[11] M. Anton Ertl and David Gregg. 2003. The Structure and Performance of Efficient Interpreters. *Journal of Instruction-Level Parallelism* 5 (Nov. 2003), 1–25.

[12] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software* (1st. ed.). Addison-Wesley Longman Publishing Co., Inc.

[13] Matthew Gaudet and Mark G. Stoodley. 2016. Rebuilding an Airliner in Flight: A Retrospective on Refactoring IBM Testarossa Production Compiler for Eclipse OMR. In *Proceedings of the 8th International Workshop on Virtual Machines and Intermediate Languages (VMIL '16)*. Association for Computing Machinery, New York, NY, USA, 24–27. https://doi.org/10.1145/2998415.2998419

[14] James Gosling, Bill Joy, Guy Steele, Gilad Bracha, and Alex Buckley. 2014. *The Java Language Specification, Java SE 8 Edition* (1st. ed.). Addison-Wesley Professional.

[15] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. *Future Generation Computer Systems* 29, 7 (Sept. 2013), 1645–1660. https://doi.org/10.1016/j.future.2013.01.010

[16] Clemens Hammacher. 2018. *Liftoff: A New Baseline Compiler for WebAssembly in V8.* Retrieved 2020-06-10 from https://v8.dev/blog/liftoff

[17] Gilbert Joseph Hansen. 1974. *Adaptive Systems for the Dynamic Runtime Optimization of Programs.* Ph.D. Dissertation. Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, USA.

[18] Kim Hazelwood and Michael D. Smith. 2002. Code Cache Management Schemes for Dynamic Optimizers. In *Proceedings of the Sixth Annual Workshop on Interaction between Compilers and Computer Architectures (INTERACT '02)*. IEEE Computer Society, Washington, DC, USA, 102–102.

[19] John L. Hennessy and David A. Patterson. 2011. *Computer Architecture: A Quantitative Approach* (5th. ed.). Elsevier.

[20] Urs Hölzle and David Ungar. 1996. Reconciling Responsiveness with Performance in Pure Object-Oriented Languages. *ACM Transactions on Programming Languages and Systems* 18, 4 (July 1996), 355–400. https://doi.org/10.1145/233561.233562

[21] Alex Iliasov. 2003. Templates-based Portable Just-in-Time Compiler. *ACM SIGPLAN Notices* 38, 8 (Aug. 2003), 37–43. https://doi.org/10.1145/944579.944588

[22] Azul Systems Inc. 2020. *Zing Runtime for Java.* Retrieved 2020-03-30 from https://www.azul.com/products/zing/

[23] Docker Inc. 2020. *Docker Image Build.* Retrieved 2020-06-22 from https://docs.docker.com/engine/reference/commandline/image_build/

[24] Eclipse Foundation Inc. 2019. *IoT Developer Survey 2019 Results.* Retrieved 2020-04-29 from https://iot.eclipse.org/community/resources/iot-surveys/assets/iot-developer-survey-2019.pdf

[25] Richard Jones, Antony Hosking, and Eliot Moss. 2016. *The Garbage Collection Handbook: The Art of Automatic Memory Management* (1st. ed.). CRC Press.

[26] Iffat H. Kazi, Howard H. Chen, Berdenia Stanley, and David J. Lilja. 2000. Techniques for Obtaining High Performance in Java Programs. *Comput. Surveys* 32, 3 (Sept. 2000), 213–240. https://doi.org/10.1145/367701.367714

[27] Ken Kennedy and John R. Allen. 2001. *Optimizing Compilers for Modern Architectures: A Dependence-based Approach* (1st. ed.). Morgan Kaufmann Publishers Inc.

[28] Prasad A. Kulkarni. 2011. JIT Compilation Policy for Modern Machines. In *Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications (OOPSLA '11)*. Association for Computing Machinery, New York, NY, USA, 773–788. https://doi.org/10.1145/2048066.2048126

[29] Tim Lindholm, Frank Yellin, Gilad Bracha, and Alex Buckley. 2014. *The Java Virtual Machine Specification, Java SE 8 Edition* (1st. ed.). Addison-Wesley Professional.

[30] Daryl Maier and Xiaoli Liang. 2017. Supercharge a Language Runtime!. In *Proceedings of the 27th Annual International Conference on Computer Science and Software Engineering (CASCON '17)*. IBM Corp., Riverton, NJ, USA, 314–314.

[31] William M. McKeeman. 1965. Peephole Optimization. *Commun. ACM* 8, 7 (July 1965), 443–444. https://dl.acm.org/doi/pdf/10.1145/364995.365000

[32] Eclipse OMR. 2020. *OMR Command-line Options.* Retrieved 2020-04-19 from https://github.com/eclipse/omr/blob/master/compiler/control/OMROptions.cpp

[33] Eclipse OpenJ9. 2020. *Eclipse OpenJ9 Repository.* Retrieved 2020-02-27 from https://github.com/eclipse/openj9

[34] Eclipse OpenJ9. 2020. *The JIT Compiler.* Retrieved 2020-04-20 from https://www.eclipse.org/openj9/docs/jit/

[35] Marius Pirvu. 2018. Optimize JVM Startup with Eclipse OpenJ9. Retrieved 2019-06-19 from https://developer.ibm.com/articles/optimize-jvm-startup-with-eclipse-openjj9

[36] B. Ramakrishna Rau. 1978. Levels of Representation of Programs and the Architecture of Universal Host Machines. *ACM SIGMICRO Newsletter* 9, 4 (Nov. 1978), 67–79. https://doi.org/10.1145/1014198.804311

[37] Ricardo Nabinger Sanchez, José Nelson Amaral, Duane Szafron, Marius Pirvu, and Mark Stoodley. 2011. Using Machines to Learn Method-Specific Compilation Strategies. In *International Symposium on Code Generation and Optimization (CGO 2011)*. IEEE, 257–266. https://doi.org/10.1109/CGO.2011.5764693

[38] Kazuyuki Shudo, Satoshi Sekiguchi, and Yoichi Muraoka. 2004. Cost-Effective Compilation Techniques for Java Just-in-Time Compilers. *Systems and Computers in Japan* 35, 12 (Nov. 2004), 10–24. https://doi.org/10.1002/scj.10564

[39] James E. Smith and Ravi Nair. 2005. *Virtual Machines: Versatile Platforms for Systems and Processes* (1st. ed.). Elsevier.

[40] TIOBE The software quality company. 2019. *The C# Programming Language.* Retrieved 2019-06-17 from https://www.tiobe.com/tiobe-index/csharp/

[41] TIOBE The software quality company. 2019. *The Java Programming Language.* Retrieved 2019-06-17 from https://www.tiobe.com/tiobe-index/java/

[42] Federico Sogaro, Eric Aubanel, Kenneth B. Kent, Vijay Sundaresan, Marius Pirvu, and Peter Shipton. 2017. MicroJIT: A Lightweight, Just-in-Time Compiler to Improve Startup Times. In *Proceedings of the 27th Annual International Conference on Computer Science and Software Engineering (CASCON '17)*. IBM Corp., Riverton, NJ, USA, 140–150.

[43] Toshio Suganuma, Takeshi Ogasawara, Mikio Takeuchi, Toshiaki Yasue, Motohiro Kawahito, Kazuaki Ishizaki, Hideaki Komatsu, and Toshio Nakatani. 2000. Overview of the IBM Java Just-in-Time Compiler. *IBM Systems Journal* 39, 1 (Jan. 2000), 175–193. https://doi.org/10.1147/sj.391.0175

[44] The JUnit Team. 2020. *JUnit.* Retrieved 2020-05-12 from https://junit.org

[45] V8 Team. 2017. *Launching Ignition and TurboFan.* Retrieved 2020-06-10 from https://v8.dev/blog/launching-ignition-and-turbofan

[46] Mark Thom, Gerhard W. Dueck, Kenneth Kent, and Daryl Maier. 2018. A Survey of Ahead-of-Time Technologies in Dynamic Language Environments. In *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering (CASCON '18)*. IBM Corp., Markham, Ontario, Canada, 275–281. http://dl.acm.org/citation.cfm?id=3291291.3291320

[47] Seth Thompson. 2016. *Experimental Support for WebAssembly in V8.* Retrieved 2020-06-10 from https://v8.dev/blog/webassembly-experimental

Designing and Evaluating New Instructions that Accelerate Sigmoid-Based Machine Learning

Lucas M. Dutton
duttonl@mcmaster.ca
McMaster University
Hamilton, Ontario, Canada

Curtis d'Alves
dalvescb@mcmaster.ca
McMaster University
Hamilton, Ontario, Canada

Wolfram Kahl
kahl@mcmaster.ca
McMaster University
Hamilton, Ontario, Canada

Robert F. Enenkel
enenkel@ca.ibm.com
IBM Canada Ltd.
Markham, Ontario, Canada

Christopher K. Anand
anandc@mcmaster.ca
McMaster University
Hamilton, Ontario, Canada

ABSTRACT

Activation functions (such as sigmoid and tanh) are an expensive operation in the neural network algorithms used in deep learning. Previous work has investigated different ways of accelerating the table lookup and exceptional-branch handling involved in such functions, with simple Instruction Set Architecture extensions. In this paper we report on the logic design for such new instructions, capable of computing sigmoid, exponentials, reciprocal and division in single precision. Such instructions could be accommodated as inexpensive add-ons to current CPU and GPU designs. We explain how to estimate the expected performance based on the resource constraints for the processor. Under reasonable assumptions for the addition of such instructions to the IBM POWER8 architecture, we estimate that for well-scheduled loops containing single-precision sigmoid functions, each sigmoid result would require 1.75 cycles asymptotically, more than a 4X acceleration over a sigmoid function implemented with primitives from an existing commercial math library (IBM Mathematical Acceleration SubSystem).

CCS CONCEPTS

• **Mathematics of computing** → **Continuous functions**; • **Hardware** → **Arithmetic and datapath circuits**; • **Computing methodologies** → *Machine learning algorithms.*

KEYWORDS

sigmoid, activation function, special function, instruction set architecture, instruction scheduling, floating point

ACM Reference Format:
Lucas M. Dutton, Curtis d'Alves, Wolfram Kahl, Robert F. Enenkel, and Christopher K. Anand. 2020. Designing and Evaluating New Instructions that Accelerate Sigmoid-Based Machine Learning. In *CASCON'20: Conference of the Centre for Advanced Studies on Collaborative Research.* ACM, New York, NY, USA, 9 pages.

CASCON'20, November 10-13, 2020, Toronto, Canada

1 INTRODUCTION

Machine learning via Neural Networks has made remarkable progress on classification problems [1], especially of images, and is increasingly being applied to generation problems, like voice synthesis [2] and forecasting population density [3]. Such rich applications with millions of pixels or sound samples result in heavy processor loads. This leads researchers and commercial processor designers to look for ways of accelerating deep learning. Two aspects of deep learning stand out for optimization: tensor algebra and activation functions. Tensor algebra can be accelerated using multi-core CPUs and GPUs[4], or going beyond that to the design of custom processors or accelerators. The Tensor Processing Unit [5] is already being used at scale. Activation functions are the next frontier in optimization [6, 7].

Accelerating mathematical functions has a long history [8–14], and previous work on accelerating exponential and reciprocal functions is directly applicable to two important activation functions, sigmoid and hyperbolic tangent,

$$\mathrm{sigmoid}(x) = \frac{1}{1+e^{-x}}, \text{ and } \tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}.$$

It is a matter of continuing research how accurate activation functions need to be [15]. Pure hardware implementations can achieve very high performance when giving up some accuracy [7, 16]. On the other extreme, activation functions can be built using vector libraries which obtain efficiency by pipelining the computation of standard math functions applied to a whole array at a time. Somewhere in the middle are the options to implement those standard math functions (particularly exponential and reciprocal) as new hardware instructions [17] which then benefit all users of standard libraries [18]; and the option of adding instructions which replace multiple instructions in existing algorithms. This is the approach we will pursue in this paper.

The original work was based on [19], which was done in the context of the Cell/B.E. SPU, compute engines with double precision, without working out the details for other precisions or doing actual scheduling. Sigmoid was not considered in the original paper. This paper considers single-precision [1] sigmoid, giving a specification for the novel instructions in the form of C code, in Section 2. Given the superscalar capabilities of POWER8 versus the SPU, and out-of-order execution, it was necessary to schedule

[1] Single precision and even lower are used in neural networks, as described in [20]

and simulate execution (Section 3) of the scheduled code to have confidence in the predicted speedup. This paper also compares estimated areas for implementing different approaches, in Section 4.

2 IMPLEMENTATION

The goal of our approach, based on the work of [19], is to simplify special function evaluation with the addition of two types of new instructions: `lookupfp`, a support instruction which detects exceptional cases and calculates reduction and restoration factors, and `fmax`, a modified floating-point multiply-add instruction. Both types of instructions are used to compute sigmoid in Figure 2. The new instructions have double outlines, and the lookup instructions have readable names. The equivalent computation in POWER assembly code is in Figure 1.

```
lxv vIn,0(rArrayIn)
vmaddfp vScaledOffset,vIn,vLog2Neg,vOffsetBump
xxland vIdxMaskedOut,vScaledOffset,vMask
vsubfp vFrac1,vIdxMaskedOut,vOffset
vmsubfp vFrac,vNegInvLog2,vIn,vFrac1
lookupfp c0,vScaledOffset,vScaledOffset,8,1     # 8=exp,1=reduce
lookupfp exp2c0,vScaledOffset,vScaledOffset,8,2
# 8=exp,2=restore
fmaxfp vFracMOffset,vOneX,vFrac,c0
vmaddfp poly0,vFrac,expCoeff1,expCoeff0
vmaddfp poly1,vFrac,expCoeff2,poly0
vmaddfp poly2,vFrac,expCoeff3,poly1
vmaddfp expNegxP1,poly2,exp2c0,vOne
lookupfp multReduc,expNegxP1,expNegxP1,9,1 # 9=recip,1=reduce
lookupfp approxRecip,expNegxP1,expNegxP1,9,2
# 9=recip,2=restore
fmaxfp recipFracMOffset,multReduc,expNegxP1,vNegOne
vmaddfp poly0r,recipFracMOffset,recipCoeff1,recipCoeff0
vmaddfp poly1r,recipFracMOffset,recipCoeff2,poly0r
vmaddfp poly2r,recipFracMOffset,recipCoeff3,poly1r
fmaxfp almost,multReduc,poly2r,vZero
vmaddfp vOut,almost,recipFracMOffset,approxRecip
stxv vOut,0(rArrayOut)
# constants
# vLog2Neg   = splat -1.442695f
# vOffsetBump = splat 33151.5f
# vOffset     = splat 33151f
# vMask       = splat 0 xffffff00
```

Figure 1: Scheduled assembly code corresponding to codegraph of Fig. 2

The algorithm is assumed to be performed on an architecture with Out-Of-Order execution and instruction prefetching (such as POWER8 which performs branch prediction when prefetching and flushes instructions from mispredicted paths [21]). We will explain the algorithm in terms of the graph, by line number (latencies are given per the POWER8 User Manual [21]):

(1) **fma** (Latency 6 cycles FPR, 9 cycles CR)calculates the index of an accurate-table reduction factor and positions it in bits 8–14 (little endian), incorporating a sign change .

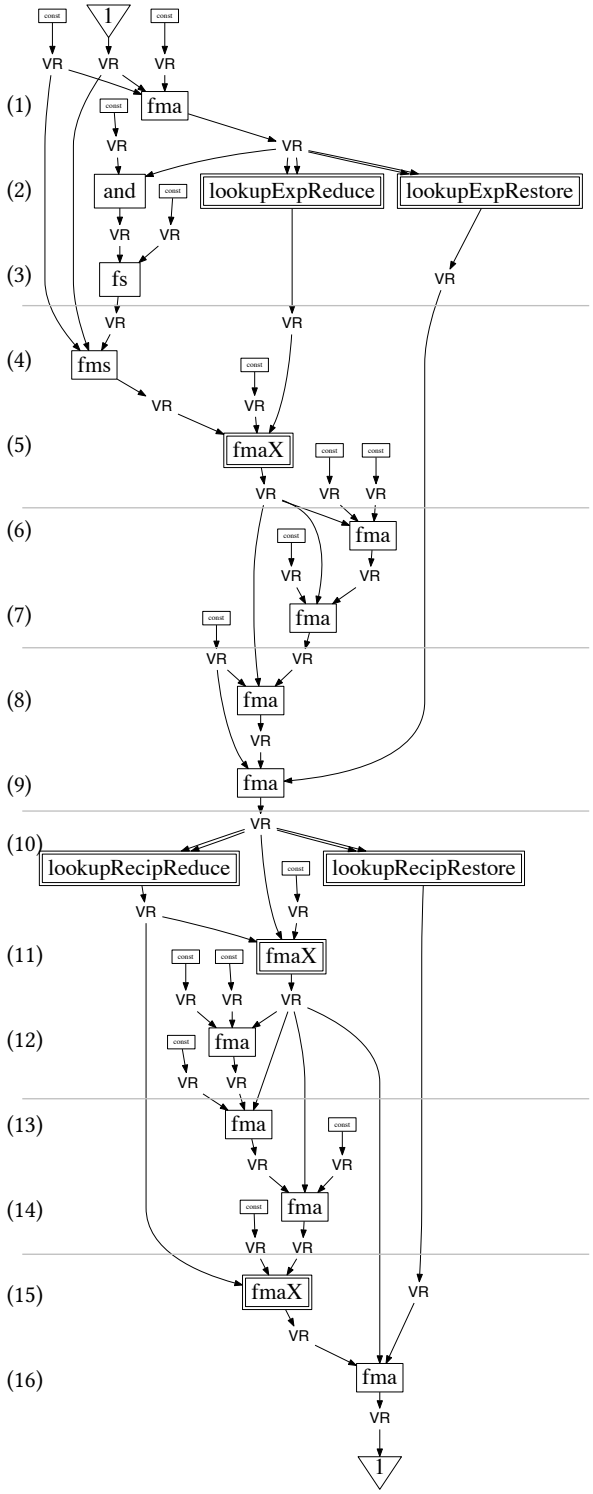


Figure 2: Branch-free codegraph for sigmoid computation. Values stored in vector registers are indicated by VR, input and output by triangles, constants by unreadably small rectangles, and machine instructions by large rectangles. New instructions are doubly-boxed. The horizontal grey lines divide groups for scheduling, with two floating-point operations per group.

(2) **and** (Latency 1-2 cycles GPR, 6 cycles CR) masks out bits 0–7, which are needed for the fraction computation. Values for range reduction and restoration, including exception handling described below .
(3-4) **fs** and **fms** (Latency 6 cycles FPR 9 cycles CR) calculate the fractional part, also incorporating a sign change.
(5-8) Calculate exponential of the residual value using a polynomial.
(9) The **fma** restores the result to the original range and adds 1, yielding $e^{-x} + 1$.
(10) Range reduction and restoration values for reciprocal are calculated.
(11) **fmaX** performs range reduction and exception handling.
(12-14) Calculate a polynomial approximation of reciprocal on the reduced interval.
(15) **fmaX** performs a final multiply, but prevents NaN creation in corner cases.
(16) Restores the result to the full range, yielding $1/(e^{-x} + 1)$.

2.1 Extended-Range Fused Multiply-Add

One of the advantages of the new instructions is the method used to handle exceptional floating point cases, especially at intermediate stages of the floating point computation. Conventional methods use branching and/or predicated execution, which may stall the instruction pipeline and increase overall latency.

We make use of the proposed new instruction `fmaX`, which is an extended-range fused multiply-add. This instruction differs from a standard `fma`, as the first argument is an extended floating point number, which is the return value of the range reduction lookup instructions. However, the second multiplicand, the addend and results are standard IEEE single floating point numbers. By internalizing the handling for exceptional cases, we avoid the use of branching and predication. IEEE specifies signalling and quiet NaNs and includes undefined bits which specific implementations can use for signalling. We use those bits to signal special values.

Exceptions are handled by supplying a special quiet NaN (QNaN) as the first multiplicand of the `fmaX` which is detected and causes a specified constant to be substituted for the result. The behaviour of `fmaX` is not specific to either the exponent or reciprocal function, and detecting a small number of QNaNs is sufficient to support a wide range of functions.

The extended-range, non-IEEE-754, 32-bit floating point number, used in the first argument to `fmaX`, also has an extra exponent bit, doubling the represented range, at the expense of one significand bit[2]. By using an extended-range range reduction factor, computations which would over/underflow are made safe.

The `fmaX` is a small modification to existing `fma` hardware, requiring an insignificant addition to the chip area. The main barrier to implementation would be the availability of an opcode.

C specification of `fmaX` with `uint_32` input and output components being the binary representation of the floating-point values:

```
vector uint_32 fmaX(vector uint_32 a, vector uint_32 b,
                    vector uint_32 c) {
  vector uint_32 result =
    {fmaX_int32(UINT32(a[0]),UINT32(b[0]),UINT32(c[0])),
     fmaX_int32(UINT32(a[1]),UINT32(b[1]),UINT32(c[1])),
     fmaX_int32(UINT32(a[2]),UINT32(b[2]),UINT32(c[2])),
     fmaX_int32(UINT32(a[3]),UINT32(b[3]),UINT32(c[3]))}

  return result;
}

uint_32 fmaX_int32(uint_32 a, uint_32 b, uint_32 c) {
   float_internal rmult    = mult(fromEXT(a),fromIEEE(b));
   float_internal raddition = add(rmult,fromIEEE(c));
  return specialCases(a,toIEEE(raddition));
}

uint_32 specialCases(uint_32 arg0, uint_32 value) {
  switch (arg0) {
    case 0x7fc00000:        \\ +∞ext → +∞IEEE
      return 0x7f800000;
    case 0xffc00000:        \\ −∞ext → −∞IEEE
      return 0xff800000;
    case 0x7fe00000:        \\ |NaN0|ext → |NaN|IEEE
      return 0x7fc00000;
    case 0x7fe00003:        \\ |NaN3|ext → 0IEEE
      return 0;

    default: return value;
  }
}
```

2.2 Lookup Functions

The C code below details the lookup functions for exp. The input to the function is a positive offset of the original input, with adjustments based on the base of the exponent, but we are only concerned with using Euler's number as our base, as it feeds into the overall computation of the sigmoid function.

```
//table constant is parameterized here.
int n = 5;

uint32_t ExpLookupReduce(uint32_t in){
    // Get lookup key from input and execute the lookup
    uint32_t lookupKey = bits32(8-n,8,in);
    uint32_t c0 = exp2tableSP[lookupKey][0];

    // Special cases
    if (isNaN32(in)) {
      return 0;
    }

    if (bits32 (31,32,in) == 1) {
      return 0x7fc00000; // Inf
    }

    if (in < 0x4700ea80){
      return 0x7fc00000; // Inf
    }

    if (in > 0x4701ff80){
      return 0;
    }

    // Negation of the reduction value from the table
    else return (uint32_t) (pow32(2,31) + c0);
}

uint32_t ExpLookupRestore(uint32_t in){
    uint32_t lookupKey, expBits, exp_c0, exp2IpC0, lastBitExpC0,
          mantissaExpC0;
```

[2]Using the Accurate Table Method[8] compensates for this.

```
    // Get input exponent bits and lookup key
    expBits = bits32(8,17,in);
    lookupKey = bits32 (8-n,8,in);

    // Execute the table lookup
    exp_c0 = exp2tableSP[lookupKey][1];

    // From the restoration value, get the LSB
    // of the exponent and the mantissa
    lastBitExpC0 = bits32(23,24,exp_c0);
    mantissaExpC0 = bits32(0,23,exp_c0);

    // The two cases below adjusts the return value
    // based on whether the input was subnormal or not
    if (expBits > 0x101) {
      exp2lpC0 = (bits32(0,8,expBits) - 1 + lastBitExpC0) *
           pow32(2,23) + mantissaExpC0;
    }

    // If the input is less than -125.5, we need to
    // construct the subnormal numbers for 2^[x]+c0.
    else {
      exp2lpC0 = roundShiftR ((0x102 - expBits - lastBitExpC0),(
           pow32(2,23) + mantissaExpC0));
    }

    // Return special outputs for invalid inputs
    if (isNaN32(in)){
      return 0x7fc00000; // Inf
    }

    if (bits32 (31,32,in) == 1) {
      return 0;
    }

    // This, and the condition below, checks if
    // input is out of representable range
    if (in < 0x4700ea80){
      return 0;
    }

    if (in > 0x4701ff80){
      return 0x7f800000;
    }
    else return exp2lpC0;

 }
```

The code below shows the computation for the `recip` lookup, which is used in the last stage of the sigmoid computation. There are two `fmaX` constants declared at the beginning of the code that are used to handle exceptional cases.

```
int n = 6; // The table size is (2^n) + 1 = 65

// fmaX constants
uint32_t nan3X = 0x7fe00003;
uint32_t infinityX = 0x7fc00000;


uint32_t RecipLookupReduce(uint32_t input) {
  uint32_t sign, rest, exponent, mantissa, leading0s,
              adjustedExponent, adjustedMantissa,
              lookupKey, fullTableVal, lastBitRecipC,
              mantissaRecipC, exponentComplemented,
              exponentFstLookup, fstLookup,
              fstBitMantissa, sndBitMantissa;

    // Extract sign, exponent and mantissa bits from the
    // input these operations extract the bits, so they
    // do not require any gates
    sign = input >> 31;
    rest = input % pow32(2,31);
    exponent = rest >> 23;
    mantissa = rest % pow32(2,23);

    // From mantissa, get the first two msbs - will be
    // used as conditionals for final answer
    fstBitMantissa = bits32(22,23,mantissa);
    sndBitMantissa = bits32(21,22,mantissa);

    // Count leading zeros in the mantissa
    leading0s = countLeading0(23,23);

    // Calculate an adjusted exponent and mantissa for
    // subnormals. Otherwise, the exponent and mantissa
    // remains unchanged from the input
    if (exponent == 0) {
        adjustedExponent = 23 - leading0s;
        adjustedMantissa = (mantissa << (leading0s+1)) % pow32
             (2,23);
    }
    else {
        adjustedExponent = exponent;
        adjustedMantissa = mantissa;
    }

    // We have the information needed to lookup into the
    // table. From the table value which only contains a
    // singleton, we can obtain two distinct values.
    lookupKey      = bits32(23-n,23,adjustedMantissa) + 1;
    fullTableVal   = recipTableSP[lookupKey];
    lastBitRecipC  = bits32(23,24,fullTableVal);
    mantissaRecipC = bits32(23-n,23,fullTableVal);

    // Construction of the extended range multiplicative
    // reduction factor.
    exponentComplemented = adjustedExponent ^ 0xff;
    exponentFstLookup    = exponentComplemented + 0x7e +
          lastBitRecipC;

    fstLookup = sign * pow32(2,31) + mantissaRecipC * pow32
          (2,22-n);
    if (exponent == 0)
        fstLookup += (exponentFstLookup+23) * pow32(2,22)
    else
        fstLookup += exponentFstLookup * pow32(2,22);

    // Return the final answer based on input parameters
    if (exponent == 0 && fstBitMantissa == 1) return fstLookup;
    if (exponent == 0 && fstBitMantissa == 0 && sndBitMantissa
          == 1) return fstLookup;
    if (exponent == 0 && fstBitMantissa == 0 && sndBitMantissa
          == 0) return nan3X;
    if (exponent == 0xfd || exponent == 0xfe) return fstLookup;
    if (exponent == 0xff) {
        if (mantissa == 0) return nan3X;
        else return 0;
    }
    return fstLookup;
}

// The only difference between the two lookups is the
// final answer. We test the same conditionals, but
// return the range restoration value instead.
uint32_t RecipLookupRestore(uint32_t input) {
```

```
uint32_t sign, rest, exponent, mantissa, leading0s,
  adjustedExponent, adjustedMantissa, lookupKey, fullTableVal,
  lastBitRecipC, mantissaRecipC, exponentComplemented,
  exponentFstLookup, fstLookup, fstBitMantissa,
       sndBitMantissa;

// Extract sign, exponent and mantissa bits from the
// input these operations extract the bits, so they do
// not require any gates.
sign = input >> 31;
rest = input % pow32(2,31);
exponent = rest >> 23;
mantissa = rest % pow32(2,23);

// First two bits used in conditionals.
fstBitMantissa  =  bits32(22,23,mantissa);
sndBitMantissa  =  bits32(21,22,mantissa);

// Count leading zeros in the mantissa.
leading0s = countLeading0(23,23);

// Calculate an adjusted exponent and mantissa for
// subnormals, other cases remain the same.
if (exponent == 0) {
    adjustedExponent = 23 - leading0s;
    adjustedMantissa = (mantissa << (leading0s+1)) % pow32
         (2,23);
}
else {
    adjustedExponent = exponent;
    adjustedMantissa = mantissa;
}

// We have the information needed to lookup into the
// table. From the table value which only contains a
     singleton
// we only need mantissaRecipC as opposed to
     recipLeftLookup
lookupKey      = bits32(23-n,23,adjustedMantissa) + 1;
fullTableVal   = recipTableSP[lookupKey];
mantissaRecipC = bits32(23-n,23,fullTableVal);

// Construction of the extended range multiplicative
// reduction factor is the same way like all
// the logLookup instruction.
exponentComplemented = adjustedExponent ^ 0xff;

// Return the final answer based on input parameters
if (exponent == 0 && fstBitMantissa == 1)
    return sign*pow32(2,31) + 0xfd*pow32(2,23) +
         mantissaRecipC*pow32(2,23-n);
if (exponent == 0 && fstBitMantissa == 0 && sndBitMantissa
     == 1)
    return sign*pow32(2,31) + 0xfe*pow32(2,23) +
         mantissaRecipC*pow32(2,23-n);
if (exponent == 0 && fstBitMantissa == 0 && sndBitMantissa
     == 0)
    return sign*pow32(2,31) + 0xff*pow32(2,23);
if (exponent == 0xfd)
    return sign*pow32(2,31) + (((pow32(2,n) +
         mantissaRecipC) * pow32(2,23-n)) >> 1);
if (exponent == 0xfe)
    return sign*pow32(2,31) + (((pow32(2,n) +
         mantissaRecipC) * pow32(2,23-n)) >> 2);
if (exponent == 0xff) {
    if (mantissa == 0) return 0;
    else return infinityX;
}
    return sign*pow32(2,31) + (exponentComplemented - 2) *
         pow32(2,23) + mantissaRecipC * pow32(2,23-n);
}
```

3 SIMULATION

We have simulated the instructions in software by compiling the C specifications in Section 2 using native integer operations. The maximum error for sigmoid, when compared to the double-precision result computed using libm exponential, on 10000 randomly generated values in the interval $[-6, 6]$, was 1.502 ulps (The minimum possible theoretical error is 0.5 ulps).

3.1 Performance

Since these instructions have not been implemented in a processor, and do not depend on vendor-specific features, it is worth exploring the impact of other processor design decisions on the performance of the target sigmoid function. To keep the presentation compact, we will report the ideal throughput in cycles/float for manually scheduled loops with instructions in order, a software-pipelined loop with seven stages, and an unrolled loop software-pipelined with eight stages. As part of the manual schedule, we arrange the instructions into dispatch groups. This is the ideal throughput, because it does not take into account limits on Out of Order (OoO) resources and memory latency, only on floating-point issue bandwidth. We will assume a machine which can dispatch two floating-point 128-bit SIMD instructions, four load/store instructions and two other instructions per cycle. Given that our loop body contains 14 floating-point instructions, our theoretical peak performance would be 1.75 cycles per float input.

If the machine in question has efficient linear load prefetching, and sufficient OoO resources, then the ideal throughput will be approached. We report the OoO requirements as the number of in-flight instructions, counting instructions in-flight from the end of the last decode/dispatch cycle until the result is available. This simplified model does not take into account restrictions on the retirement order. We also assume that the instructions in Figure 2 are dispatched in groups given by taking the instructions in order of height, and continuing until two floating point instructions are contained in each group. In the scheduled version as shown in Figure 3, group divisions are shown as horizontal grey bars. The constants can either be loaded as needed, or loaded in advance, or a combination of the two, depending on register pressure, and whether a single sigmoid or a vector of sigmoids is being computed. We also assume that up to six instructions can be dispatched from the instruction fetcher to the functional queues per cycle. To simulate performance under software pipelining, we treat those 7 groups as stages, and reverse their order within the body of the loop. This is a valid software-pipelined schedule, not optimal, but our simulations will show it to be close to the lower bound on execution time.

For simplicity, we assume that the `load` and `store` have a forwarding latency of two cycles and the `and` a latency of one cycle, meaning that the result is available to a dependent instruction on the cycle after issue; and we will assume that all other instructions have a latency of ℓ. Examining Figure 2, we notice that all instructions other than the lookup instructions are strictly ordered. The lookup instructions have a natural order, given by their dependant

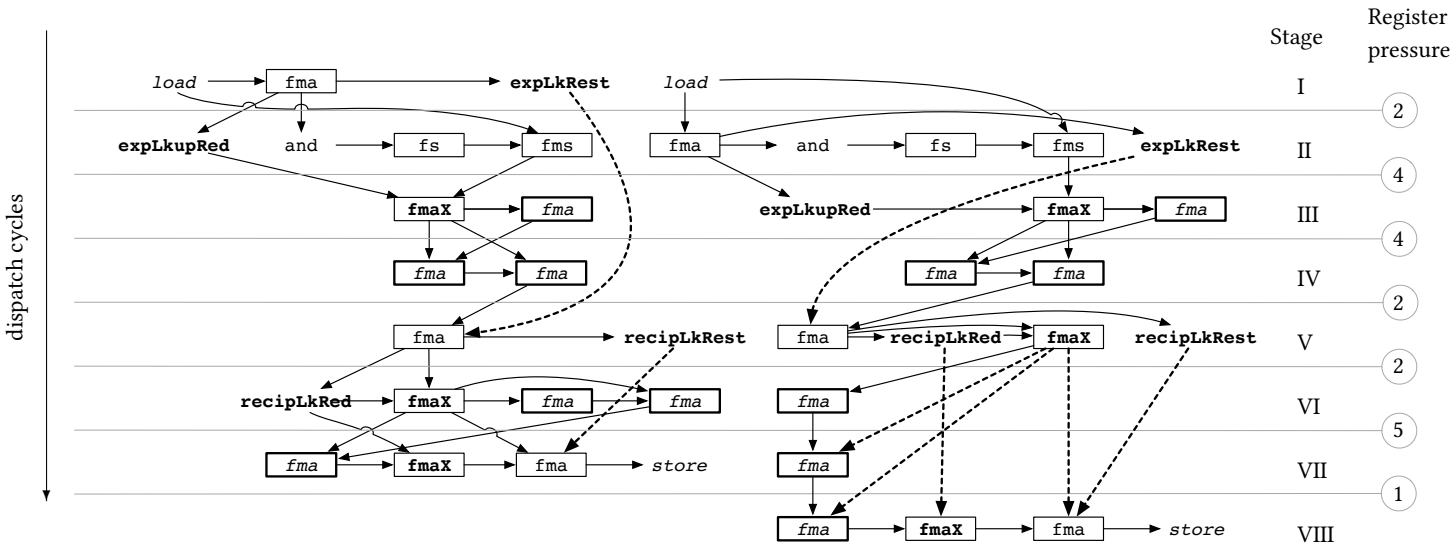


Figure 3: Scheduled instructions for a CPU capable of dispatching two FP or SIMD-FP instructions per cycle, not showing constant loads. The FP instructions are boxed, with the `fma` instructions involved in polynomial evaluations in slanted text. The grey lines indicate dispatch group boundaries, and the number of registers values which are live across group boundaries is indicated in the circle on the right, for a total of 19 registers required for other code. On a machine with 32 128-bit SIMD registers and infinite OoO capabilities, this would result in a throughput of 1.75 cycles/float. On a machine with fewer OoO resources, it would be necessary to use software pipelining and/or unrolling.

instructions, and to reduce register pressure, they should be scheduled after the `and` and `fmaX` respectively. Finally, there are two pinch points in the code, which correspond to the `fms`s which compute the intermediate $\exp(-x) + 1$ and the final result. Scheduling these instructions together with the immediately preceding `fma` or `fmaX` also reduces register pressure. From these observations, there is a natural grouping of instructions into dispatchable groups shown in two versions in Figure 3. The entire Figure 3 gives a grouping used for software pipelining for a twice-unrolled loop body. Either connected component could be used for software pipelining without unrolling. In this figure, rows correspond to dispatchable groups, and are named I, II up to VIII. Dashed arrows represent dependencies which cross multiple group boundaries, which must be treated carefully when software pipelining, as explained below. Solid arrows are dependencies requiring a logical register allocation, and counting the number of sources for those arrows which cross a group boundary tells us the register pressure at that point if the groups were scheduled in order. These register pressures are indicated in the grey circles, and since they sum to 20, register allocation without spilling would succeed on both the in-order schedules, and pipelined loop bodies, with the groups being scheduled in any order.

We will investigate three scheduling scenarios:

I-O In-order scheduling following Figure 2, which requires no code duplication, but has the highest requirements for OoO execution. Register pressure is very low in this case (a maximum of five registers are required at group boundaries, if constants are loaded within the group in which they are consumed). This indicates that it would be easy for compilers to in-line single-vector computations, and interleave computations with instructions from the caller.

P Software pipelining based on the left connected component in Figure 3, with the instruction groups in reverse order.

Px2 Software pipelining of both components of Figure 3, corresponding to an unroll factor of two.

In the second two cases, in addition to values which are assigned registers for their lifetimes, there are dashed edges which are produced in one group and not consumed by the end of the next group. If these dashed edges were stored in registers, after software pipelining, values from one iteration would be overwritten by the value from the next iteration before the first value is ready to be consumed. There are different ways of avoiding this, including hardware support for rotating registers and ferrying values through a series of registers. Our implementation uses a fixed array with rotating read and write pointers offset by the number of groups separating the producer and consumer. It is possible to have multiple consumers with different offsets. In architectures with segregated vector registers and general-purpose registers used for addresses and offsets, this solution works well, because general-purpose registers and load/store bandwidth are not bottlenecks. In our example, the number of rotating buffers is two or five, which imposes no constraints on scheduling.

We simulated execution of the three scheduled loops over 200- and 1000-element arrays. The performance is better for larger arrays, and the unrolled and pipelined loop is very close to the lower bound execution time over the full range of instruction latencies, as shown in Figures 4 and 5, illustrating that the size of the arrays

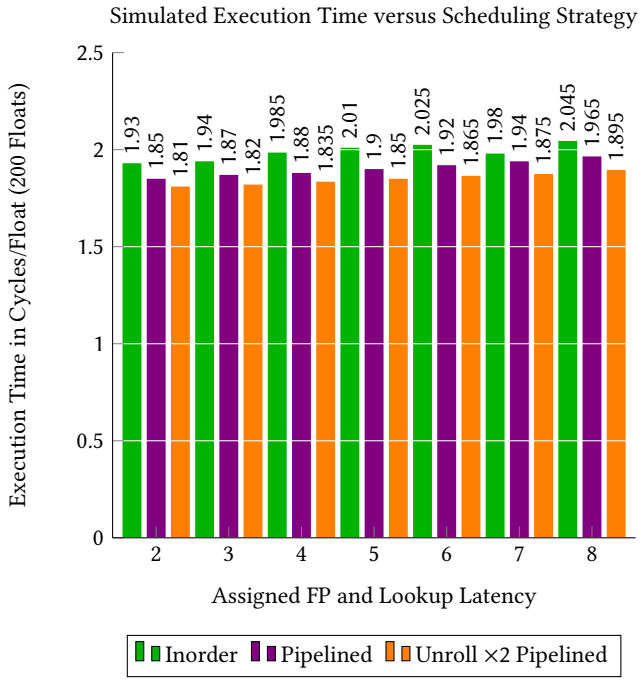


Figure 4: Average execution time per float, when computing $x \mapsto 1/(\exp(-x)+1)$ over a vector of length 200, simulated under different scenarios, assuming latencies of 2 to 8 cycles for both floating-point operations and the lookup instructions, and three different schedules: in-order using the order given in Figure 2; pipelined with prologue, epilogue and loop body given by assigning a different stage to each of the groups in the left component of Figure 3, and ordering the groups within the loop body in reverse order; and finally, pipelined after unrolling once, as represented by the groups/stages in Figure 3.

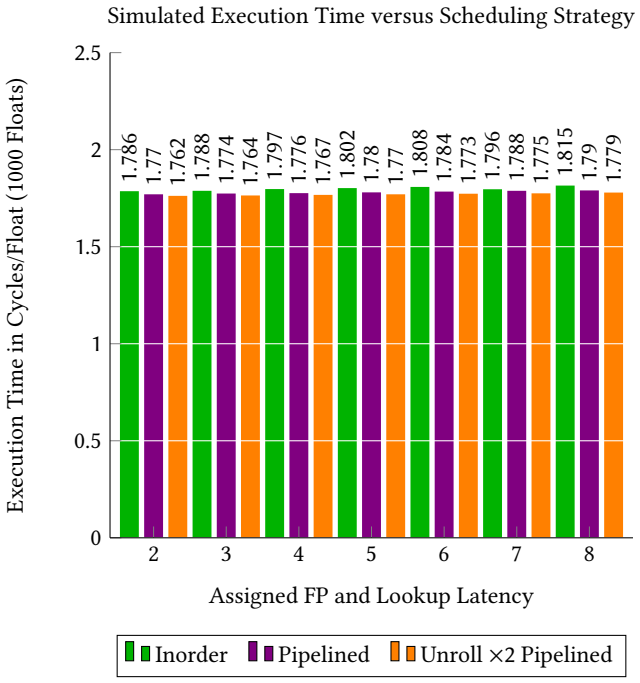


Figure 5: Average execution time per float, when computing $x \mapsto 1/(\exp(-x)+1)$ over a vector of length 1000.

does not significantly change the performance. But these results depend on sufficient OoO execution resources. Figure 6 shows the number of in-flight instructions which must be supported to reach this performance. For the unrolled, pipelined loop, this requires from 22 to a modest 44 instructions, and grows linearly with the latency, but the unpipelined loop requires up to 112 in-flight instructions, and is not very predictable. Figure 7 shows how the number of inflight instructions varies regularly over the course of the loop in the unrolled, pipelined case.

4 ALTERNATIVES

As we have said, there are multiple alternatives for accelerating activation functions. Compared to our expected throughput on a POWER-like processor of 1.75 cycles per float for sigmoid, we have benchmarked the actual performance for a vector sigmoid function constructed from `vsexp` and `vsrec`, which are vector exponential and reciprocal functions operating on a vector of single-precision

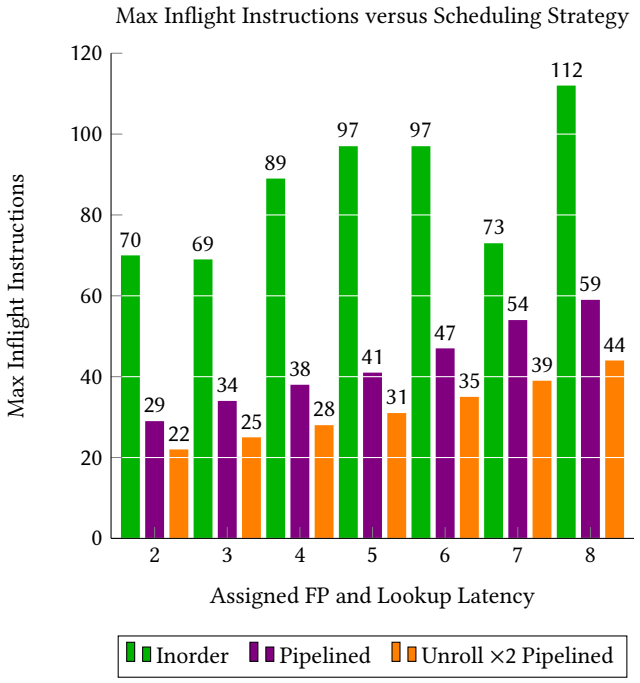


Figure 6: Peak number of inflight instructions for each strategy. The amount of sigmoid calculations assigned for each strategy remain the same as in Figure 4

[3]Calculated using the number of transistors in IBM POWER5 (276 million) and the area of the die (389 mm^2) [22].

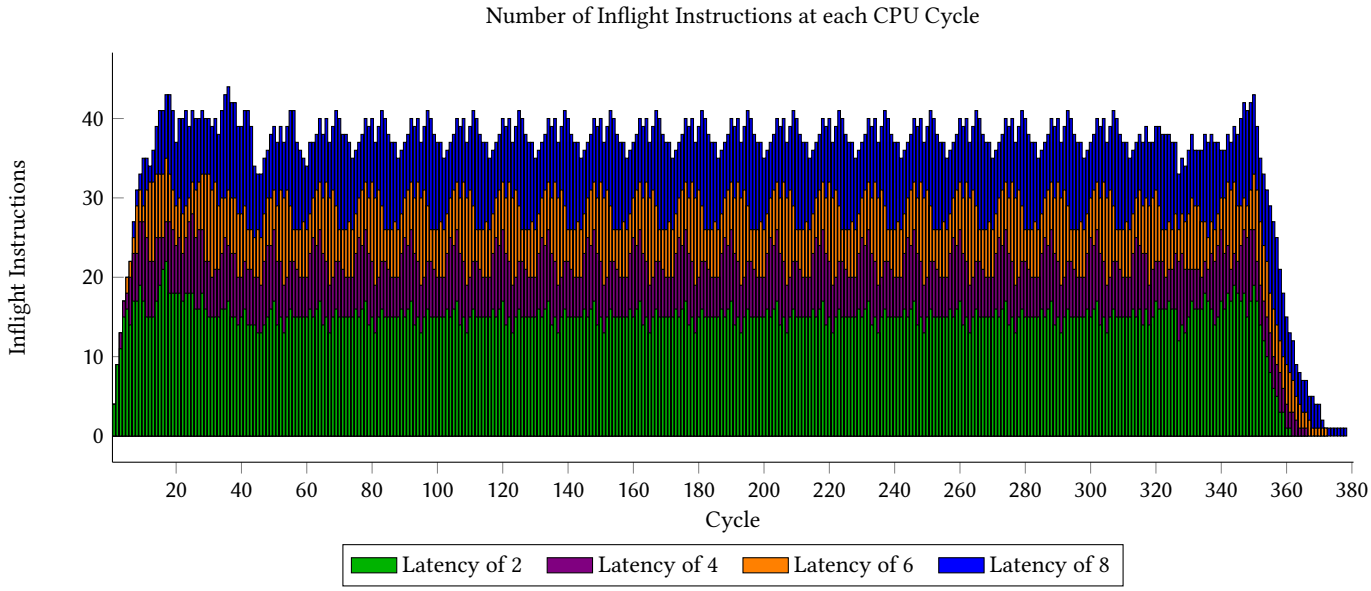


Figure 7: Number of inflight instructions at each cycle for the unrolled and pipelined loop.

	expRed	expRest	recipRed	recipRest
gates	548	3562	465	382
height	16	39	22	21
table	32×24	32×24	65×23	65×23
estimated area[3]in μm^2	772	5020	655	538

Table 1: Complexity of the proposed instructions in gates, circuit height, and size of the lookup table in bits.

arguments. We found that performance approached 8 cycles per float on POWER8[23]. Since we measured 8 cycles/float for MASS-based sigmoid, and simulated 1.75 cycles/float for the new approach (8/1.75=4.57), therefore we are confident of achieving a 4X throughput increase.

	divReduce	divRestore	divTotal	KD[24]
gates	450	452	902	n/a
height	22	22	22	n/a
table	65×23	65×23	65×46	128×21
estimated area in μm^2	634	637	1271	25016

Table 2: Comparison of our division implementation, which is similar to recip, to an existing solution.

Table 1 shows actual circuit size, height and embedded table size, as well as an estimate of circuit area for a POWER5-era 90nm process. By using this process, we can compare our estimated circuit size to the single-precision divisor [24]. In Table 2, we compare a combined recip/div lookups to the design in [24]. We consider these costs reasonable because the bigger tables already exist in processors including translation look-aside buffers, and dedicated memory is used in GPUs and digital signal processors capable of storing these tables and the number of transistors required is tiny compared to the 4.2 billion transistors in the POWER8 architecture [25].

The reported circuits have maximum fan-in and fan-out of 4. Our implementations make use of AND, OR, NOR and NOT gates as basic building blocks, and more complicated circuit designs leverage these gates as part of their design, such as multiplexors, custom shifters and counting leading zeros. We do not count the lookup table and lookup instruction as part of the design, as this will be implementation or processor specific, but we show the sizes of the table in the number of bits, which designers can use to estimate the costs of implementing the lookup tables.

5 CONCLUSION

We have demonstrated that a light-weight ISA extension which accelerates special functions, can achieve significant accelerations for sigmoid, and would achieve similar accelerations for other activation functions. The number of gates, and the table sizes are very modest, while the resulting code is easily scheduled and ripe for in-lining. We recommend a similar design for special-function acceleration on RISC-like architectures, and think it represents a good example of the type of architecture extension recommended by Hennessy [26] to overcome physical barriers to performance scaling.

ACKNOWLEDGMENTS

We thank the IBM Centre for Advanced Studies and NSERC for financial support.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[2] S. O. Arik, M. Chrzanowski, A. Coates, G. Diamos, A. Gibiansky, Y. Kang, X. Li, J. Miller, A. Ng, J. Raiman *et al.*, "Deep voice: Real-time neural text-to-speech," *arXiv preprint arXiv:1702.07825*, 2017.

[3] A. Wanto, A. P. Windarto, D. Hartama, and I. Parlina, "Use of binary sigmoid function and linear identity in artificial neural networks for forecasting population density," *IJISTECH (International Journal of Information System & Technology)*, vol. 1, no. 1, pp. 43–54, 2017.

[4] D. I. Lyakh, "An efficient tensor transpose algorithm for multicore CPU, Intel Xeon Phi, and NVidia Tesla GPU," *Computer Physics Communications*, vol. 189, pp. 84–91, 2015.

[5] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*. IEEE, 2017, pp. 1–12.

[6] B. Zamanlooy and M. Mirhassani, "Efficient vlsi implementation of neural networks with hyperbolic tangent activation function," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 1, pp. 39–48, 2013.

[7] A. H. Namin, K. Leboeuf, R. Muscedere, H. Wu, and M. Ahmadi, "Efficient hardware implementation of the hyperbolic tangent sigmoid function," in *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*. IEEE, 2009, pp. 2117–2120.

[8] S. Gal, "Computing elementary functions: A new approach for achieving high accuracy and good performance," in *Proceedings of the Symposium on Accurate Scientific Computations*. London, UK: LNCS 235, Springer-Verlag, 1986, pp. 1–16.

[9] M. Püschel, J. M. F. Moura, B. Singer, J. Xiong, J. Johnson, D. Padua, M. Veloso, and R. W. Johnson, "Spiral: A generator for platform-adapted libraries of signal processing algorithms," *Int. J. High Perform. Comput. Appl.*, vol. 18, no. 1, pp. 21–45, Feb. 2004. [Online]. Available: http://dx.doi.org/10.1177/1094342004041291

[10] M. Puschel, J. M. Moura, J. R. Johnson, D. Padua, M. M. Veloso, B. W. Singer, J. Xiong, F. Franchetti, A. Gacic, Y. Voronenko *et al.*, "Spiral: Code generation for dsp transforms," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 232–275, 2005.

[11] C. K. Anand and W. Kahl, "An optimized Cell BE special function library generated by Coconut," *IEEE Transactions on Computers*, 2009.

[12] S. Chevillard, M. Joldeş, and C. Lauter, "Sollya: An environment for the development of numerical codes," in *International Congress on Mathematical Software*. Springer, 2010, pp. 28–31.

[13] N. Brunie, F. d. Dinechin, O. Kupriianova, and C. Lauter, "Code generators for mathematical functions," in *2015 IEEE 22nd Symposium on Computer Arithmetic*, June 2015, pp. 66–73.

[14] N. Brunie, "Modified fused multiply and add for exact low precision product accumulation," in *Computer Arithmetic (ARITH), 2017 IEEE 24th Symposium on*. IEEE, 2017, pp. 106–113.

[15] S. Elfwing, E. Uchibe, and K. Doya, "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning," *Neural Networks*, 2018 (on-line first). [Online]. Available: https://doi.org/10.1016/j.neunet.2017.12.012

[16] C.-W. Lin and J.-S. Wang, "A digital circuit design of hyperbolic tangent sigmoid function for neural networks," in *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*. IEEE, 2008, pp. 856–859.

[17] A. Sodani, "Knights landing (knl): 2nd generation Intel® Xeon Phi processor," in *Hot Chips 27 Symposium (HCS), 2015 IEEE*. IEEE, 2015, pp. 1–24.

[18] "Intel® MKL 2019 vector mathematics performance and accuracy data."

[19] A. Sharma and C. K. Anand, "A domain-specific architecture for elementary function evaluation," *International Journal of Mathematics and Mathematical Sciences*, vol. 2015, 2015.

[20] N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers," in *Advances in neural information processing systems*, 2018, pp. 7675–7684.

[21] *POWER8 Processor User's Manual for the Single-Chip Module*, IBM.

[22] J. G. Clabes, J. Friedrich, M. Sweet, J. DiLullo, S. Chu, D. W. Plass, J. Dawson, P. Muench, L. Powell, M. Floyd, B. Sinharoy, M. Lee, M. Goulet, J. Wagoner, N. S. Schwartz, S. L. Runyon, G. Gorman, P. Restle, R. N. Kalla, and J. Steve Dodson, "Design and implementation of the POWER5™ microprocessor," 01 2004, pp. 670–672.

[23] B. Sinharoy, J. Van Norstrand, R. J. Eickemeyer, H. Q. Le, J. Leenstra, D. Q. Nguyen, B. Konigsburg, K. Ward, M. Brown, J. E. Moreira *et al.*, "IBM POWER8 processor core microarchitecture," *IBM Journal of Research and Development*, vol. 59, no. 1, pp. 2–1, 2015.

[24] T.-J. Kwon and J. Draper, "Floating-point division and square root implementation using a taylor-series expansion algorithm with reduced look-up tables," in *Circuits and Systems, 2008. MWSCAS 2008. 51st Midwest Symposium on*. IEEE, 2008, pp. 954–957.

[25] E. J. Fluhr, J. Friedrich, D. Dreps, V. Zyuban, G. Still, C. Gonzalez, A. Hall, D. Hogenmiller, F. Malgioglio, R. Nett *et al.*, "5.1 power8 tm: A 12-core server-class processor in 22nm soi with 7.6 tb/s off-chip bandwidth," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, 2014, pp. 96–97.

[26] D. Patterson, "An interview with Stanford University president John Hennessy," *Commun. ACM*, vol. 59, no. 3, pp. 40–45, Feb. 2016.

Evaluating the Effectiveness of Static Word Embeddings on the Classification of IT Support Tickets

Yasmen Wahba
Dept. Of Comp. Sc.
Western University
London, ON, Canada
ywahba2@uwo.ca

Nazim H. Madhavji
Dept. Of Comp. Sc.
Western University
London, ON, Canada
madhavji@gmail.com

John Steinbacher
IBM Canada Lab.
Toronto, ON, Canada
jstein@ca.ibm.com

ABSTRACT

Support tickets are service requests, initiated by a system's end-users when they encounter issues with their system. With a wide user-base and system issues, there will be an ongoing influx of generated support tickets. Manual classification and prioritization is effortful and error-prone, that can lead to incorrect routing and delays in the resolution of the issues.

Recently, various state-of-the-art machine learning and deep learning methods have been applied to automate the process of text classification. Because the quality of these methods highly depends on the quality of the associated "features", in this paper we focus on the "feature engineering" step in the classification process. In particular, we evaluate the effectiveness of using different static word embeddings on the accuracy of classifying IT support tickets.

In collaboration with an industrial partner, we were able to train and evaluate our machine learning model on 1.6 million support tickets and 32 ticket categories.

The experimental results show that the traditional Term Frequency Inverse Document Frequency (TFIDF) bag-of-words along with Support Vector Machines (SVM) provides competitive results and sometimes outperforms static word embedding models such as word2vec while maintaining low computational cost.

CCS CONCEPTS

•Computing methodologies~Artificial intelligence~Natural language processing~Lexical semantics•Computing methodologies~Machine learning

KEYWORDS

Customer support tickets, Machine learning, Word embedding, Feature engineering, Natural language processing

ACM Reference format:

Yasmen Wahba, Nazim H. Madhavji and John Steinbacher. 2020. Evaluating the Effectiveness of Static Word Embeddings on the Classification of IT Support Tickets. In *Proceedings of ACM CASCON conference (CASCON'20). Toronto, Ontario, Canada, 9 pages.*

CASCON'20, Nov 10-13, 2020, Toronto, Canada

1 Introduction

Service agents spend a large amount of time on manually classifying the incoming tickets. Unfortunately, this process is complicated, and the support agents have no reference to best practices based on historical data. With the massive growth of data, the need to automate ticket classification becomes crucial. Based on the ticket description, the support agents determine the category of the problem and triage the ticket to the appropriate team for resolving the issue.

A typical ticket description is unstructured and hence this makes it challenging for natural language processing. Also, the ticket may contain typos as well as abbreviations which adds to the complexity. Ticket classification is an important process that ensures that tickets get routed to the right support agent. Otherwise, there can be delays, customer dissatisfaction, escalation to management, and reactionary fixes at high cost [1].

Recently, neural networks and deep learning models have surpassed traditional machine learning approaches by delivering state-of-the-art results in several text classification tasks, including spam filtering, sentiment analysis and question answering. Hence, these models have become a favorable choice for any text classification or clustering task. However, this comes with the cost of increased computational complexity and therefore increased model training time [2].

Word embeddings are one of the popular uses of neural networks for handling natural language text. These embeddings are able to place words in a vector space that contains semantic information about the words. Thus, similar words will be placed close to each other. Capturing word semantics in different contexts is what differentiates between a static and a dynamic word embedding.

This paper evaluates static word embedding models which have been shown to be very effective in addressing various NLP tasks including document classification [3] [4] [5]. However, to our knowledge, no work has compared the performance of these word embeddings against old methods like bag-of-words. Thus, the key question being addressed in this paper is: How effective is using static word embeddings in the task of IT ticket classification?

Overall, the findings of our study suggest that the problem of classifying IT support tickets can be addressed efficiently using old traditional methods such as TFIDF bag-of-words that does not involve the complexity found in neural network models.

The rest of the paper is organized as follows. Section 2 describes background. Section 3 describes related work. Section 4 describes our project context. Section 5 describes the methodology and Section 6 presents research results. Section 7 concludes the paper.

2 Background

The quality of machine learning/deep learning model comes from extensive feature engineering than from the learning technique itself, as the quality of these methods highly depends on the quality of available features [6]. To apply machine learning algorithms, human text must be converted to numeric form through what is known as vector representation [7].

Handling vector representations is of the challenges of natural text processing. This is because the same set of words can convey different meanings in different contexts or if given in a different order. This is known as polysemy, which is the association of one word with two or more distinct meanings [8]. This level of sophistication in understanding text and coming up with the best vector representation for words is why word embeddings emerged in this research direction as an alternative to the bag-of-words (BOW) vector model [9].

The core idea behind word embeddings is that words that are used in similar contexts will be given similar representations, thus capturing word semantics. Two of the popular word embeddings that attracted many researchers are Word2Vec [10] trained on Google News and Glove [11] which is trained on Wikipedia. These methods generate word vectors by training the word embedding algorithm against a huge corpus of text. However, these embeddings are referred to as 'static', in the sense that each word is represented by only one vector regardless of the context. Thus, the word bank in "I went to the bank to withdraw money before going fishing at the river bank "will have the same embedding. To mitigate this problem, dynamic representations or so-called contextualized embeddings emerged as a replacement for static word embeddings and improved many NLP tasks [12][13][14]. These embeddings aim to capture word semantics in different contexts to address the issue of context-dependent nature of words.

3 Related Work

Since our work aims to investigate the effectiveness of using word embeddings to classify IT support tickets, we first give an overview of the existing literature studies on the problem of ticket classification, and then we examine some of the studies on the effectiveness of domain-specific word embeddings.

Diao [15] leveraged large expert communities with domain knowledge to develop a rule-based approach, where experts author the classification rules to classify problem tickets.
Paramesh's [16] followed the ensemble approach to improve the accuracy of their ticket classifier system, by combining the predictions of Bagging, Boosting and Voting ensemble on four base classifiers. Similarly, the work in [17] tackled the problem using an ensemble of SVM classifiers and re-sampling techniques to handle the problem of data imbalance.

Authors in [18] investigated different classification algorithms to classify incident tickets, SVM was reported to perform well on all data samples. However, [19] reported Multinomial Naive Bayes (MNB) to outperform Softmax Regression Neural Network (SNN) for classifying help desk tickets.

In contrast to 'Flat' text classification, hierarchal classification has also been addressed. Authors in [20] proposed a novel architecture for hierarchical classification that extends the strengths of SVM classifiers to leverage prior knowledge about class relationships. While authors in [21] investigated hierarchal multi-label classification of incident tickets by leveraging the known hierarchical relationship between categories using a novel greedy algorithm 'GLabel' to label the predicting ticket. Adding to the previous work, authors in [22] proposed an algorithm to utilize the knowledge from domain experts. Note that all these papers focus on the final stage of text classification pipeline, which is model building and machine learning algorithms.

With the introduction of static word embeddings in 2013 by Mikolov [23] that leveraged neural networks, Natural Language Processing (NLP) tasks have changed dramatically. Accordingly, text classification methods were classified into those which use neural networks and the ones that do not. Authors in [24] reported that recurrent neural networks (RNNs) using word embeddings data outperform the classic solutions for the task of classifying data from customer service systems and task trackers. Similarly, the work in [25] leveraged deep networks, where convolutional neural network (CNN) was reported to achieve the best performance for the task of classifying IT tickets without much feature engineering. While [26] achieved an improved classification accuracy using a linear support vector machine

(SVM) along with term weighted Word2Vec model. In contrast to using pre-trained word vectors, authors in [27] provided additional semantic information by enriching the vectors by Part-of-Speech (POS) tags.

Despite the success of the general domain word embeddings like Word2Vec in many NLP tasks, domain specific terms always represent a challenge, since these embeddings are trained over general corpora like books or Wikipedia. Some researchers suggested fusing domain-specific data with general data for a better performance [28] [29]. While the work in [30] introduced 'SO_Word2Vec', a domain specific word embedding that is trained over 15GB of textual data from Stack Overflow posts. Similarly, authors in [31] presented Annotation Word Embedding (AWE) which incorporates different kinds of domain knowledge. The model's performance outperformed state-of-the-art baselines on two cybersecurity applications. The work in [32] reported an increase by 17 percent in accuracy compared to state-of-the-art methods when using domain specific word embedding to classify patent applications.

Upon critical analysis of the literature, we note that it is not clear at all how effective static word embeddings are in solving the task of IT support ticket classification. This problem has a caveat that it contains IT-related terminologies (e.g., mongoDB, kubernetes, and logdna) and unique fragments of text (e.g., HTML code, IP addresses, XML code) and specific abbreviations (e.g., paas, vlan, and iam). We note the current trend of using neural and deep learning architectures for solving text classification problems. This imposes us to think whether it is worth to use sophisticated and computationally expensive neural or deep learning architectures for the task of classifying support tickets.

This was thus a motivation for us to investigate the usefulness of word embeddings over the simple TFIDF in solving the IT support ticket classification problem.

4 Project Context

This section describes the nature of our dataset. This is followed by an analysis of the problem context in Section 4.2

4.1 Dataset

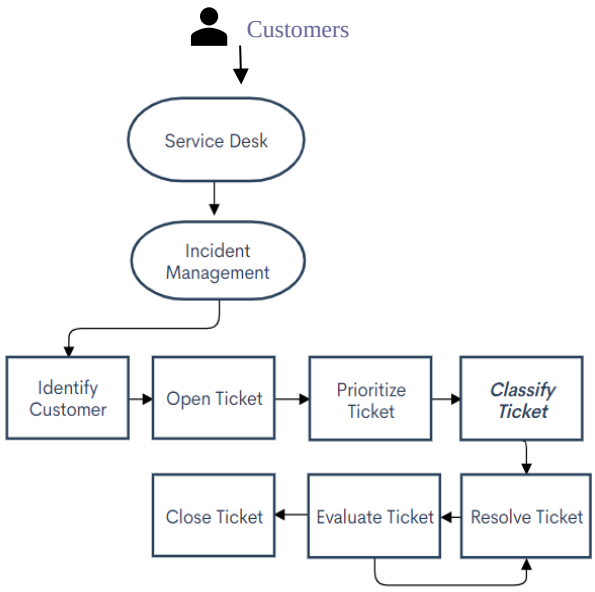
Our dataset is considered a large-scale dataset containing over 1.6 million support tickets classified into 32 different ticket categories. For customers to submit a new ticket, they have to give a short and a full description of their issue. We noted that predominantly only the short description field is used (as shown in Table 1). This problem is handled in the pre-processing stage by concatenating both fields into a new one. Also, the description entered by the customer is unstructured containing non-English characters, dates and typos.

Table 1: Example snapshot of the dataset

DESCRIPTION	CATEGORY	SHORT_DESCRIPTION	DV_CATEGORY
	2	Virtual Server Cancellatio	Infrastructure
	4	Mass Data Migration - Pc	Project Office
	2	Virtual Server Cancellatio	Infrastructure
	2	[ABNS] ë°ì´í„° ì „ì†¡ì	Infrastructure
	2	Payment is late	Infrastructure
	2	Test	Infrastructure
	2	Server Cancellation - 08/	Infrastructure
	2	ì‹ ìš©ì¹´ë“œ ë§Œê¸°ê¸°	Infrastructure
	2	Virtual Server Cancellatio	Infrastructure

4.2 Problem Analysis

In a typical IT organization, customers raise an issue (i.e., open a ticket), through the IT service desk. IT service management (ITSM) is responsible for dealing with the resolution of these tickets. Figure 1 depicts the standard process of incident management that starts with ticket generation, which is followed by prioritization, categorization and then resolution of the ticket by an IT specialist. If the customer is satisfied, then the ticket is closed.



Legend: Elipse – entity; Rectangle – task; Arrow – flow.

Figure 1: Process flow of IT service management

As can be seen from the processing pipeline, classification plays a substantial role. Wrong manual ticket classification will prevent the tickets from being triaged to the appropriate support team. In turn, this can lead to the problem of time delays in ticket resolution, violation of service-level agreements, and customer dissatisfaction.

Thus, machine learning based methods for automation is considered crucial for the overall incident management efficiency. Millions of support tickets can be sorted in a fraction of the time spent manually for this task, thus freeing the agents to focus on more important or other tasks. In addition to reducing the number of escalations triggered by unhappy customers.

Ticket classification is one of the use cases of document classification where ticket's description submitted by customers represent a document and the ticket category is the document label. Therefore, the steps for classifying a support/issue ticket are the same steps followed in a typical document classification problem. The following figure (Figure 2) shows the main steps for a text classification model.

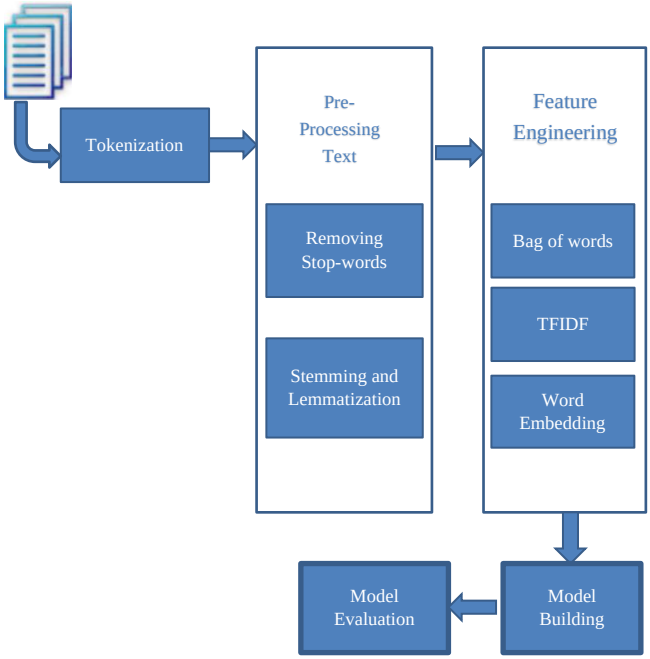


Figure 2: Text classification steps

There are a few types of text classification based on the number of classes/categories to predict:

- Binary classification: When the total number of classes is two, any prediction can contain either one of those classes.
- Multi-class classification: Involves classifying instances into more than two classes, where each instance can be classified into one of those classes.
- Multi-label classification: Involves classifying instances into more than two classes, where each instance can be classified into one or more categories at the same time.

Our work is considered as a Multi-class classification task, where the support tickets are classified into 32 different ticket categories (e.g. Infrastructure, Project Office, Sales, Databases, etc. – please see later in Figure 3 for more).

5 Methodology

This section gives an overview of the dataset we used in our study and the pre-processing steps performed to clean the data. This is followed by the experimental steps and the word embeddings used in this study.

5.1 Dataset Preparation

The first stage in building a text classification model is cleaning the data (pre-processing stage). This stage aims to reduce the vocabulary size and remove noise found in the input documents. This is anticipated to help in maximizing the classifier's performance [33] [34].

For a natural language text, noise can be spelling errors, abbreviations, character repetitions, missing punctuations, non-standard words, etc. In our work, we applied the regularly used operations in text mining in addition to domain-specific operations that we perform based on the ticket descriptions and domain experience from the support agents of our industrial partner. Given the ticket structure in Table 2, we are only interested in ticket description and its corresponding category; all other fields are thus ignored.

Table 2: Typical support ticket data

Ticket number	Ticket category	Ticket priority	Ticket state	Ticket description
CS177	Services	Medium	In Progress	IP Billing address missing
CS190	Infrastructure	High	Open	Payment late

The following are the common pre-processing tasks that we carried out in this order:

1. Removing missing data.
2. Removing numbers and special characters.
3. Converting text to lowercase.
4. Word tokenization.
5. Removing stop words.
6. Lemmatization.
7. Removing non-English words.

While we applied such operations, we also found the need to employ some domain-specific steps. For example, upon careful examination of our ticket descriptions, we noticed the presence of Chinese characters. Hence, we performed the regular step of non-English words removal. A side-effect was that some important domain-specific words were removed in the process. Thus, we created a list of words that could have an impact on ticket classification and called it 'to_keep' list. For this purpose, we incorporated domain knowledge from our support agents along with some common knowledge of some IT terminologies.

For example, words such as “watson” and “vmware” are kept and not removed during the pre-processing step.

Since the focus of this research is more on feature engineering and pre-processing steps. We carefully examined the list of discarded words during the step of non-English words removal, and, to our surprise, we found a huge list of common English words. For example, words such as *groups*, *questions*, *requests*, and *chatbot* were removed.

There are two reasons behind this. First, the “Words” Corpus from NLTK [35] that we used is a delimited list of dictionary words [36], hence, words are stored in their singular form. Second, this Corpus is not an exhaustive list of all English words, so some words might be missing [37] (e.g. blog, chatbot). To mitigate the first problem, we performed lemmatization which ensures that words are kept in their dictionary or base form, known as “lemma”. This step is done prior to removing non-English words. While for the second problem, we added the missing words to our “to_keep” list and used another lexical database for English called “WordNet” that is published by Princeton University [38].

Our dataset described in Section 4.1 suffers from a severe imbalance, where the distribution of class samples is uneven by a large amount in the training dataset (e.g. 1:100 or more) as shown in Figure 3. This imbalance makes the classification algorithm biased towards the major categories and ignore the minor ones, leading to poor classification for these classes [39] [40].

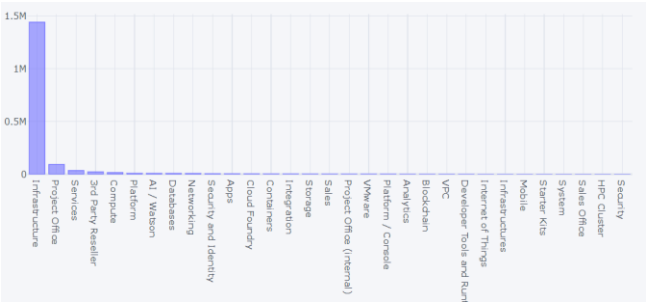


Figure 3: Distribution of classes and the severe imbalance

There are several approaches for handling this imbalance, and they can be grouped into four categories [41]: (i) algorithm-level, (ii) data-level, (iii) cost-sensitive and (iv) ensemble learning. Since our work is focused on the pre-processing stage, ‘data-level’ approaches such as oversampling techniques [42] [43] and undersampling [44] are more relevant for our purpose. However, these methods have major drawbacks [41] and sometimes reported to be ineffective and may often cause negative effect on multiclass tasks [45]. Hence, we decided to keep the original distribution while in the future we intend to gather more data for the minor classes.

5.2 Empirical Study

In this section, we describe the empirical study that we conducted. In particular, we describe data characteristics, infrastructure used, word vectorization models used and the performance measures.
We collected over 1.6 million tickets from a large cloud-based ticketing system, classified into 32 different categories. Our experimental algorithms are written in Python 3.8.3. The testing machine is Windows 10 with Intel Core i7 CPU 2.71 GHz and 32GB of RAM.

The following are the different word vectorization models used in this study:
1. GN_Word2Vec [46]: This is a neural network–based implementation that is provided by Google and is trained on a part of the Google News dataset (about 100 billion words). The model contains 300-dimensional vectors for 3 million words and phrases.
2. SO_Word2Vec [47]: This is a domain-specific Word2Vec model that is trained on Stack Overflow posts which is a generic model of Software Engineering knowledge containing 200-dimensional vectors.
3. CO_Word2Vec: This is the Word2Vec algorithm trained on our corpus of support tickets using a size of 100 dimensional vectors. Thus, we call it Corpus Word2Vec (CO_Word2Vec).
4. TFIDF[1] bag-of-words: This is the simplest yet a powerful technique for vectorizing text documents [48].

An important parameter that we considered when applying the TFIDF vectorizer is N-grams. An ‘N-gram’ is simply a sequence of N words that predicts the occurrence of a word based on the occurrence of its (N – 1) previous words. Uni-grams or single words are the default setting. In our study, we set ngram_range to (1,3) which means that we included feature vectors consisting of all unigrams, bigrams, and trigrams.

For the machine learning models we chose two popular and simple classification algorithms:

1. Support Vector Machines (SVM): reported as one of the best algorithms for text classification [49] [50].
We chose the LinearSVC algorithm in Scikit-learn library [51]. Reason is that this algorithm implements “one-vs-the-rest” or what is known as one-versus-all (OVA) multi-class strategy, which is suitable for high dimensional data and, has a very low running time [52].
2. Logistic Regression (LR): a simple linear classifier that uses maximum likelihood for estimation method [53].

[1] TFIDF stands for Term Frequency-Inverse Document Frequency, which is a combination of two metrics:
1. Term frequency (*tf*): a measure of how frequently a term, t, appears in a document, d.
2. Inverse document frequency(*idf*): a measure of how important a term is. It is computed by dividing the total number of documents in our corpus by the document frequency for each term and then applying logarithmic scaling on the result.

To evaluate the performance of the above mentioned two classification algorithms, we used the standard information retrieval (IR) measures, Precision[2], Recall[3] and F-measure[4] or F-score. As mentioned before, our task is a multi-classification one, and our data is hugely imbalanced, so, we used the F-score metric that is the harmonic mean value of precision and recall. This measure is suitable for multi-classification tasks.

6 Results

The experimental results obtained after experimenting with different static word embeddings are presented in the following chart.

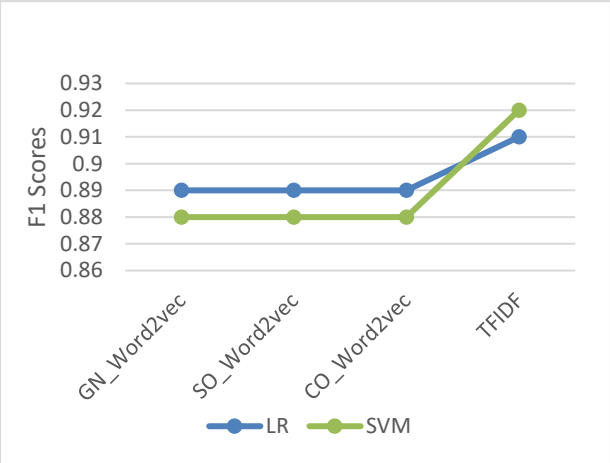


Figure 4: Performance of different Word2Vec embeddings versus TFIDF bag-of-words

Figure 4 shows the weighted-averaged F1 score of each word embedding model evaluated using two base classifiers: Linear SVM and Logistic Regression. Surprisingly, the traditional TFIDF bag-of-words model achieved a competitive accuracy of 92% using SVM classifier and 91% using Logistic Regression. While the three static word2vec models achieved a close classification accuracy of 89% trained using the Logistic Regression classifier, however, with a high computational cost.

Also, since our dataset is highly skewed, it was expected that the classification algorithm will be biased towards the major classes, leading to a high classification accuracy for the two major ticket categories (Infrastructure & Project Office), while showing poor accuracies towards the minor ones. This is shown clearly in the classification report presented in Figure 5. For the precision and recall values, note that some classes show zero or very low F-scores. This is because the number of instances collected for these categories were below 50 records; hence, the classification algorithm failed to classify them.

It must be noted that, when using TFIDF bag-of-words model, trying different n-grams is important. In our study, we experimented different n-grams and recorded the performance for all 32 classes. Results showed that using tri-grams (1,3) enhanced the F-score of almost all minor classes.

Accuracy for LinearSVC : 0.9241538025635081

class	precision	recall	f1-score	support
0	0.31	0.04	0.07	222
1	0.32	0.15	0.21	2899
2	0.97	0.99	0.98	865043
3	0.44	0.45	0.44	21268
4	0.75	0.74	0.74	55678
5	0.31	0.05	0.08	1430
6	0.00	0.00	0.00	12
7	0.00	0.00	0.00	50
8	0.24	0.04	0.08	179
9	0.39	0.23	0.29	5418
10	0.56	0.56	0.56	5139
20	0.45	0.17	0.24	878
30	0.69	0.50	0.58	803
40	0.42	0.27	0.33	2520
50	0.48	0.52	0.50	9801
60	0.52	0.60	0.55	2143
70	0.60	0.64	0.62	4954
80	0.36	0.14	0.20	587
90	0.56	0.49	0.52	2097
100	0.37	0.07	0.11	223
110	0.55	0.07	0.12	87
120	0.49	0.45	0.47	4533
130	0.24	0.05	0.08	1353
140	0.80	0.53	0.64	3277
150	0.22	0.07	0.10	73
160	0.54	0.42	0.47	1682
170	0.55	0.25	0.35	1414
180	0.44	0.11	0.18	1487
190	1.00	0.05	0.10	19
200	0.68	0.11	0.18	13399
210	1.00	0.04	0.08	25
600	0.67	0.52	0.59	705
accuracy			0.92	1009398
macro avg	0.50	0.29	0.33	1009398
weighted avg	0.91	0.92	0.92	1009398

Figure 5: Classification report of TFIDF showing precision and recall of linear SVM for all 32 classes

In Figure 6, we describe the performance of the four vectorization models used in the study to classify each of the classes. It is clear that the imbalance problem is affecting the classifier's performance to recognize the minor classes. However, the performance of the traditional TFIDF bag-of-words to classify the minor classes outperformed that of the three static word embeddings. This is demonstrated in Figure 6(d). The performance of the static word embeddings (Figures 6(a), 6(b) and 6(c)) to classify the minor classes is almost the same with neglectable differences.

[2] $Precision = \frac{True\ Positive\ (TP)}{True\ Positive(TP)+False\ Positive\ (FP)}$

[3] $Recall = \frac{True\ Positive\ (TP)}{True\ Positive\ (TP)+False\ Negative\ (FN)}$

[4] $F1-Score = 2 \times \frac{Precision \times Recall}{Precision+recall}$

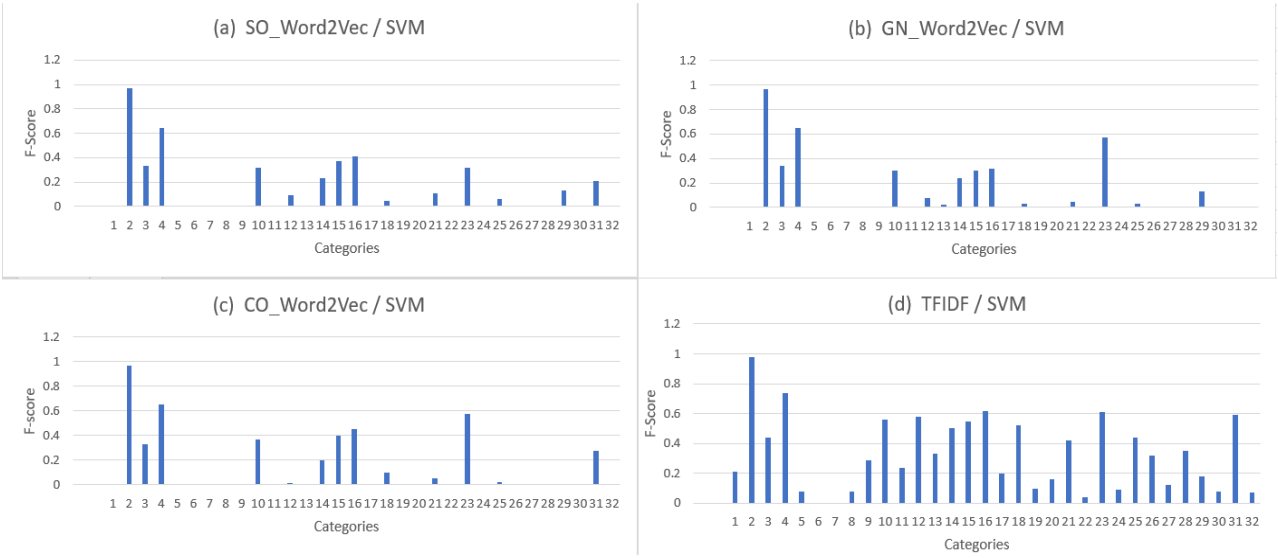


Figure 6: Classification accuracy of SVM using different embedding models for all 32 classes

Both classification algorithms (SVM & LR) showed very similar results, for the sake of space, we included only the results for SVM classifier.

However, in an effort to examine the representational power of the domain-specific word embedding SO_Word2Vec versus the general word embedding GN_Word2Vec in capturing some of ticket-specific keywords, we retrieved the top 5 similar words for some of the four frequently appearing keywords that we noticed while pre-processing our support tickets. These results are shown in Table 2.

Table 2: Examples of top 5 related words in SO Word2Vec and Google News model

Keyword	Most similar in SO Word2Vec	Most similar in GN Word2Vec
cloud	clound, cloud-based, azure, gcp, iaas	clouds, cloud, cloud_computing Abu_Risha_assassination , Carefully_cautiously
fetch	retrieve, fetching, fetched, fetches, retrieved	fetchesd, fetches, fetching, Sotheby_auction, presale_estimate
watson	nlc, nlu, stt, speech-to-text, luis	thompson, walsh, bennett, armstrong, crawford
abort	aborts, aborting, aborted, interrupted, terminate	aborting, aborted, aborts, abort, abort_fetus

As can be seen from the table, the domain-specific word embedding trained on a software engineering domain (Stack Overflow), was able to capture semantically related words better than the general pre-trained model on Google news. It's also clear that they are very effective in identifying domain specific ambiguities.

One important observation to note here is that the task of classification of support tickets can be automated using simple traditional methods such as TFIDF with a high classification accuracy and a very low computational power compared to complex algorithms that are often hard to interpret. While the problem of poor accuracies for minor classes can be mitigated efficiently by collecting more data for the minor categories, or by using a closed feedback loop between the support agents and the ML algorithm, which continuously improves the model by adding new ticket information for minor classes.

7 Conclusion and Future Work

Classifying support tickets plays an important role in any help disk system. Automation of the tickets' classification should improve the resolution time significantly and minimise errors in the escalation process. In this paper, we describe the effectiveness of different static word embeddings including a domain-specific word embedding for the software engineering domain (SO_Word2Vec) on the task of classifying IT support tickets of a real-world dataset.

Results showed that unlike general document classification, IT support tickets do not benefit much from using static word embeddings. This is due to the domain-specific words that are

considered as Out of Vocab (OOV) words for pre-trained embeddings. Also, the level of polysemy (i.e., the coexistence of many possible meanings for a word or phrase) in IT technical text is very low which is the reason why the traditional TFIDF bag-of-words provided comparable performance and sometimes outperformed static word embeddings with a low computational cost and fast training time. For future work, we plan to apply contextual word embeddings (e.g. BERT, ELMO) and investigate its effectiveness in improving the accuracy of our minor classes. Also, we intend to address the hierarchical classification problem of support tickets and we intend to include other datasets to our experiments to strengthen our results.

ACKNOWLEDGMENTS

We thank our collaborator Chris Jeffs sincerely for his help with the dataset and for sharing domain knowledge. Also, our sincere appreciation to Timothy Tan for his generous time and expertise in addressing many of our questions. Also, we are grateful for the helpful comments from the reviewers which helped to improve this paper. We are very grateful to NSERC of Canada and IBM Canada Ltd. for their generous support to carry out this research.

REFERENCES

[1] V. S. Sheng, B. Gu, W. Fang, and J. Wu, "Cost-sensitive learning for defect escalation," *Knowledge-Based Systems*, vol. 66, pp. 146–155, 2014, doi: 10.1016/j.knosys.2014.04.033.

[2] W. Fu and T. Menzies, "Easy over hard: A case study on deep learning," in Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Aug. 2017, vol. Part F130154, pp. 49–60, doi: 10.1145/3106237.3106256.

[3] S. Kadam, A. Gala, P. Gehlot, A. Kurup, and K. Ghag, "Word Embedding Based Multinomial Naive Bayes Algorithm for Spam Filtering," in *Proceedings - 2018 4th International Conference on Computing, Communication Control and Automation, ICCUBEA 2018*, Jul. 2018, doi: 10.1109/ICCUBEA.2018.8697601.

[4] M. Shi, K. Wang, and C. Li, "A C-LSTM with word embedding model for news text classification," in *Proceedings - 18th IEEE/ACIS International Conference on Computer and Information Science, ICIS 2019*, Jun. 2019, pp. 253–257, doi: 10.1109/ICIS46139.2019.8940289.

[5] O. B. Deho, W. A. Agangiba, F. L. Aryeh, and J. A. Ansah, "Sentiment analysis with word embedding," in *IEEE International Conference on Adaptive Science and Technology, ICAST*, Oct. 2018, vol. 2018-August, doi: 10.1109/ICASTECH.2018.8506717.

[6] H. Liu, *Feature Engineering for Machine Learning and Data Analytics*. CRC Press, 2018.

[7] C. Orsenigo, C. Vercellis, and C. Volpetti, "Concatenating or Averaging? Hybrid Sentences Representations for Sentiment Analysis," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Nov. 2018, vol. 11314 LNCS, pp. 567–575, doi: 10.1007/978-3-030-03493-1_59.

[8] A. Sennet, "Polysemy," Aug. 2016, doi: 10.1093/OXFORDHB/9780199935314.013.32.

[9] Z. S. Harris, "Distributional Structure," WORD, vol. 10, no. 2–3, pp. 146–162, Aug. 1954, doi: 10.1080/00437956.1954.11659520.

[10] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations ofwords and phrases and their compositionality," in *Advances in Neural Information Processing Systems*, Oct. 2013, Accessed: May 27, 2020. [Online].

[11] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global Vectors for Word Representation." Accessed: May 27, 2020. [Online]. Available: http://nlp.

[12] Y. Liu *et al.*, "Deep Contextualized Word Embeddings for Universal Dependency Parsing," *ACM Trans. Asian Low-Resour. Lang. Inf. Process*, vol. 19, no. 9, 2019, doi: 10.1145/3326497.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," Oct. 2018, Accessed: May 14, 2020. [Online]. Available: http://arxiv.org/abs/1810.04805.

[14] Z. Yao, Y. Sun, W. Ding, N. Rao, and H. Xiong, "Dynamic word embeddings for evolving semantic discovery," in *WSDM 2018 - Proceedings of the 11th ACM International Conference on Web Search and Data Mining*, Feb. 2018, vol. 2018-Febuary, pp. 673–681, doi: 10.1145/3159652.3159703.

[15] Y. Diao, H. Jamjoom, and D. Loewenstern, "Rule-based problem classification in IT service management," in *CLOUD 2009 - 2009 IEEE International Conference on Cloud Computing*, 2009, pp. 221–228, doi: 10.1109/CLOUD.2009.80.

[16] S. P. Paramesh, C. Ramya, and K. S. Shreedhara, "Classifying the Unstructured IT Service Desk Tickets Using Ensemble of Classifiers," in *Proceedings 2018 3rd International Conference on Computational Systems and Information Technology for Sustainable Solutions, CSITSS 2018*, Dec. 2018, pp. 221–227, doi: 10.1109/CSITSS.2018.8768734.

[17] J. Xu, L. Tang, and T. Li, "System situation ticket identification using SVMs ensemble," *Expert Systems with Applications*, vol. 60, pp. 130–140, Oct. 2016, doi: 10.1016/j.eswa.2016.04.017.

[18] S. P. Paramesh and K. S. Shreedhara, "Automated IT service desk systems using machine learning techniques," in *Lecture Notes in Networks and Systems*, vol. 43, Springer, 2019, pp. 331–346.

[19] G. Son, V. Hazlewood, and G. D. Peterson, "On automating XSEDE user ticket classification," in *ACM International Conference Proceeding Series*, 2014, pp. 1–7, doi: 10.1145/2616498.2616549.

[20] L. Cai and T. Hofmann, "Hierarchical document categorization with Support Vector Machines," in *International Conference on Information and Knowledge Management, Proceedings*, 2004, pp. 78–87, doi: 10.1145/1031171.1031186.

[21] C. Zeng, W. Zhou, T. Li, L. Shwartz, and G. Y. Grabarnik, "Knowledge Guided Hierarchical Multi-Label Classification over Ticket Data," *IEEE Transactions on Network and Service Management*, vol. 14, no. 2, pp. 246–260, Jun. 2017, doi: 10.1109/TNSM.2017.2668363.

[22] C. Zeng, T. Li, L. Shwartz, and G. Y. Grabarnik, "Hierarchical multi-label classification over ticket data using contextual loss," in *IEEE/IFIP NOMS 2014 - IEEE/IFIP Network Operations and Management Symposium: Management in a Software Defined World*, 2014, doi: 10.1109/NOMS.2014.6838267.

[23] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," Jan. 2013, [Online]. Available: http://arxiv.org/abs/1301.3781.

[24] V. Lyubinets, T. Boiko, and D. Nicholas, "Automated Labeling of Bugs and Tickets Using Attention-Based Mechanisms in Recurrent Neural Networks," in *Proceedings of the 2018 IEEE 2nd International Conference on Data Stream Mining and Processing, DSMP 2018*, Oct. 2018, pp. 271–275, doi: 10.1109/DSMP.2018.8478511.

[25] J. Han and M. Akbari, "Vertical Domain Text Classification: Towards Understanding IT Tickets Using Deep Neural Networks," Thirty-Second AAAI Conference on Articial Intelligence, 2018, Accessed: Aug. 12, 2020. [Online].

[26] J. Lilleberg, "Support Vector Machines and Word2vec for Text Classification with Semantic Features."

[27] B. A. Rabut, A. C. Fajardo, and R. P. Medina, "Multi-class document classification using improved word embeddings," in *ACM International Conference Proceeding Series*, Oct. 2019, pp. 42–46, doi: 10.1145/3366650.3366661.

[28] A. Z. Yen, H. H. Huang, and H. H. Chen, "Fusing domain-specific data with general data for in-domain applications," in *Proceedings - 2017 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2017*, Aug. 2017, pp. 566–572, doi: 10.1145/3106426.3106473.

[29] F. Wu, Y. Huang, and Z. Yuan, "Domain-specific sentiment classification via fusing sentiment knowledge from multiple sources," *Information Fusion*, vol. 35, pp. 26–37, May 2017, doi: 10.1016/j.inffus.2016.09.001.

[30] V. Efstathiou, C. Chatzilenas, and D. Spinellis, "Word embeddings for the software engineering domain," in *Proceedings - International Conference*

on Software Engineering, May 2018, pp. 38–41, doi: 10.1145/3196398.3196448.

[31] A. Roy, Y. Park, and S. Pan, "Incorporating domain knowledge in learning word embedding," in *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*, Nov. 2019, vol. 2019-November, pp. 1568–1573, doi: 10.1109/ICTAI.2019.00226.

[32] J. Risch and R. Krestel, "Learning patent speak: Investigating domain-specific word embeddings," in *2018 13th International Conference on Digital Information Management, ICDIM 2018*, Sep. 2018, pp. 63–68, doi: 10.1109/ICDIM.2018.8846972.

[33] A. Krouska, C. Troussas, and M. Virvou, "The effect of preprocessing techniques on Twitter sentiment analysis," in *IISA 2016 - 7th International Conference on Information, Intelligence, Systems and Applications*, Dec. 2016, doi: 10.1109/IISA.2016.7785373.

[34] A. Barushka and P. Hajek, "The Effect of Text Preprocessing Strategies on Detecting Fake Consumer Reviews," 2019, doi: 10.1145/3383902.3383908.

[35] "NLTK Data." http://www.nltk.org/nltk_data/ (accessed Jun. 16, 2020).

[36] "words (Unix) - Wikipedia." https://en.wikipedia.org/wiki/Words_(Unix) (accessed Jun. 16, 2020).

[37] "Natural Language Processing with Python: Analyzing Text with the Natural ... - Steven Bird, Ewan Klein, Edward Loper - Google Books." https://books.google.com.eg/books?hl=en&lr=&id=KGIbfiiP1i4C&oi=fnd&pg=PR5&dq=Natural+Language+Proces&redir_esc=y#v=onepage&q=missing%20words&f=false (accessed Jun. 16, 2020).

[38] G. A. Miller, "WordNet: A Lexical Database for English," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, Nov. 1995, doi: 10.1145/219717.219748.

[39] S. Akkaradamrongrat, P. Kachamas, and S. Sinthupinyo, "Text Generation for Imbalanced Text Classification," in *JCSSE 2019 - 16th International Joint Conference on Computer Science and Software Engineering: Knowledge Evolution Towards Singularity of Man-Machine Intelligence*, Jul. 2019, pp. 181–186, doi: 10.1109/JCSSE.2019.8864181.

[40] J. Wang and M. L. Zhang, "Towards mitigating the class-imbalance problem for partial label learning," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Jul. 2018, pp. 2427–2436, doi: 10.1145/3219819.3220008.

[41] "Imbalanced Learning: Foundations, Algorithms, and Applications - Google Books." https://books.google.com.eg/books?id=CVHx-Gp9jzUC&pg=PT37&lpg=PT37&dq=%22oversampling%22+%22drawbacks%22&source=bl&ots=2hTjHpIsag&sig=ACfU3U1pbRbPST37VCAJAyC7M-1wzVwSXQ&hl=en&sa=X&ved=2ahUKEwiBjsLFxYvqAhVJ8uAKHRozBVQQ6AEwC3oECA0QAQ#v=onepage&q=%22oversampling%22%20%22drawbacks%22&f=false (accessed Jun. 17, 2020).

[42] C. Chiamanusorn and K. Sinapiromsaran, "Extreme anomalous oversampling technique for class imbalance," in *ACM International Conference Proceeding Series*, Dec. 2017, pp. 341–345, doi: 10.1145/3176653.3176671.

[43] T. Zhu, Y. Lin, and Y. Liu, "Synthetic minority oversampling technique for multiclass imbalance problems," *Pattern Recognition*, vol. 72, pp. 327–340, Dec. 2017, doi: 10.1016/j.patcog.2017.07.024.

[44] B. W. Yap, K. A. Rani, H. A. Abd Rahman, S. Fong, Z. Khairudin, and N. N. Abdullah, "An application of oversampling, undersampling, bagging and boosting in handling imbalanced datasets," in *Lecture Notes in Electrical Engineering*, 2014, vol. 285 LNEE, pp. 13–22, doi: 10.1007/978-981-4585-18-7_2.

[45] Z. H. Zhou and X. Y. Liu, "Training cost-sensitive neural networks with methods addressing the class imbalance problem," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 1, pp. 63–77, Jan. 2006, doi: 10.1109/TKDE.2006.17.

[46] "GitHub - mmihaltz/word2vec-GoogleNews-vectors: word2vec Google News model." https://github.com/mmihaltz/word2vec-GoogleNews-vectors (accessed Jun. 15, 2020).

[47] "GitHub - vefstathiou/SO_word2vec: A word2vec model trained over Stack Overflow." https://github.com/vefstathiou/SO_word2vec (accessed Jun. 15, 2020).

[48] D. Sarkar and D. Sarkar, "Text Classification," in *Text Analytics with Python*, Apress, 2016, pp. 167–215.

[49] T. Joachims, "Text categorization with Support Vector Machines: Learning with many relevant features," Springer, Berlin, Heidelberg, 1998, pp. 137–142.

[50] P. A. Telnoni, R. Budiawan, and M. Qana'a, "Comparison of Machine Learning Classification Method on Text-based Case in Twitter," in *Proceeding - 2019 International Conference on ICT for Smart Society: Innovation and Transformation Toward Smart Region, ICISS 2019*, Nov. 2019, doi: 10.1109/ICISS48059.2019.8969850.

[51] "1.4. Support Vector Machines — scikit-learn 0.23.1 documentation." https://scikit-learn.org/stable/modules/svm.html (accessed Jun. 04, 2020).

[52] V. K. Chauhan, K. Dahiya, and A. Sharma, "Problem formulations and solvers in linear SVM: a review," *Artificial Intelligence Review*, vol. 52, no. 2. Springer Netherlands, pp. 803–855, Aug. 15, 2019, doi: 10.1007/s10462-018-9614-6.

[53] "sklearn.linear_model.LogisticRegression — scikit-learn 0.23.1 documentation." https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html (accessed Jun. 17, 2020).

Deep learning approaches to classify the relevance and sentiment of news articles to the economy

Jingli Wang, Ashok Bhowmick, Mucahit Cevik, Ayse Basar
jingli.wang@ryerson.ca,ashok.bhowmick@ryerson.ca,mcevik@ryerson.ca,ayse.bener@ryerson.ca
Raymond Chang School and Data Science Laboratory, Ryerson University
Toronto, Ontario

ABSTRACT

We consider a text classification task over an open source dataset involving news snippets and their relevance to the US economy. Text classification and sentiment analysis have been performed using nine different classifiers among which three are the traditional machine learning models, namely, support vector machine, extreme gradient boosting and logistic regression, and six neural network-based methods. The neural net frameworks include long short-term memory (LSTM), bidirectional long short-term memory (BiLSTM) and an ensemble of one dimensional convolution network (1D CNN) with LSTM/BiLSTM. Both word-to-vector and term-frequency inverse-document-frequency vectors are used in our analysis with text and sentiment classification tasks. A detailed comparative study is provided to assess the relative performance of different classification approaches. It is observed that the ensemble with 1D CNN performs better in both binary and multiclass classifications. Specifically, in the multinomial sentiment classification, 1D CNN with BiLSTM has the best performance as opposed to 1D CNN with LSTM in the binary text classification. BiLSTM architecture which incorporates the backward dependencies turns out as superior to LSTM by a margin of 30% in multiclass classification even though the considered dataset is small and inherently challenging. Further analysis to evaluate the impact of successive increases in percentage of augmented data reveals that such augmentation has a limit up to 180% in this dataset beyond which the performance starts decreasing.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

BiLSTM, LSTM, 1D CNN, *tf-idf*, NLP, Text classification, Sentiment analysis, News snippet, US economy

ACM Reference Format:
Jingli Wang, Ashok Bhowmick, Mucahit Cevik, Ayse Basar. 2018. Deep learning approaches to classify the relevance and sentiment of news articles to the economy. In *Proceedings of CASCON'20, November 10 - 13 2020, Toronto, Canada.* IBM Corp., Riverton, NJ, USA, 10 pages.

1 INTRODUCTION

Text classification problems are often challenging as it is nontrivial to differentiate the real meaning of a text just from the common interpretation of each word because of extensive use of rhetoric which is by and large common in every language and an integral part of linguistic communication. For instance, a typical English phrase such as "how to play rising rates" in a news article seemingly dealing with economic issues, is using a question to express an unpleasant information. Thereby, it is plausible to apprehend that if a piece of text is difficult to read and understand by human, the same maybe true for a machine learning algorithm to interpret as well [12]. In addition, predictive modelling to classify the underlying tone as sentiment hidden in text data by the evaluation context of human understanding e.g. star rating of a review in a scale of 1 - 5 etc. are more involved task in natural language processing (NLP) compared to classifying the review texts as to whether it is positive or negative under a predefined common standard [13, 24, 55, 57]. The difficulty in such multinomial classification typically stems from the fact that for example, the texts reviewed as three stars often share many features to that scoring as four stars [11]. As the number of such classes that shares the same features in a dataset increases, so does the potential of confusion to predict the classes, and by that, the dataset becomes complex to work with [19]. In this regard, the machine learning tasks dealing with such complexities have become more and more dependent on sophisticated architectures of which the precision level of the outcome relies on fine tuning the hyperparameters when training on such data. Further, along with such data complexity [33], there may be the inherent issue of class diversity [50] and class balance [8] in a dataset having multiple classes.

In today's data driven world of business decisions, sentiment analysis has gained important role in the bulk of NLP tasks through wide range of applications in analyzing product reviews and social network postings to analytically understand social demands [37]. In recent years, economists and business people began to analyze sentiments in economic or financial news to comprehend consumer's confidence level that contributed to a sharpened prediction in the financial markets [16]. Machine readable news service powered by Reuters use raw news with metadata for intra-day trading and market surveillance [2]. On the other hand, financial news still receives limited attention compared to other social media information [1, 27]. A closer look on the development of various facets of NLP since 1940s shows four major phases with emphasis on areas including syntax and semantics. Before the 1980s, most approaches

were established on predetermined rules whereas the statistical approach that emphasises summarizing rules from texts has been widely considered since late 1980s [23].

The current techniques in predictive text classification may be grouped under two main streams: rule-based and machine learning-based techniques. Recently, the focus has been on sophisticated architectures of deep neural networks to deal with the level of complexities as outlined above. Recurrent Neural Networks (RNNs) [29] achieved high performance for the text classification because they take into account the relationships between words, unlike traditional machine learning approaches or more primitive neural networks. Long short-term memory networks (LSTM) [21], as an advanced recurrent neural network, reduced the impact of vanishing and exploding gradients in typical RNNs [37, 56]. Though significant improvements have been achieved in implementing LSTM in natural language processing, it still has a major drawback that it fails to consider the existence of backward dependencies. To overcome this weakness, bidirectional long short-term memory networks (BiLSTM) [48] has been developed which has an architecture to count both forward and backward dependencies [30, 37].

The degree of difficulty of a text dataset through its structure of having many classes requires sophisticated architectures of predictive algorithms [12]. Our aim in this work is to explore this issue by executing many text classification tasks with a news snippet dataset. The objective in this work is to undertake

(1) binary text classification aiming to predict the relevance of news snippets towards economy, and
(2) analyze the sentiment labelled in the tone of the relevant news snippets by executing multinomial classification over several classes.

We employ various statistical tests to validate the performance of nine different algorithms to assess how the degree of sophistication in the architecture of the algorithms might play a role in prediction performance.

We start our analysis by executing all the basic data cleaning measures and then examining the class imbalance issue in the dataset. Specifically, we balance the dataset by standard data augmentation approaches and then examine the class diversity after executing the binary classification task. Next, we execute the multinomial sentiment classification based on the labeling of the same dataset using news tonal quality. We consider the data augmentation for the multinomial classification as well, however, we keep the data diversity unaltered. Note that the data augmentation is prevalent in NLP, yet such introduction of artificiality into the dataset for having a better training of the classifier leading to superior prediction capability does not have a well defined general prescription on the suitability and extent of such practice. We empirically investigate following questions in our study. Does the data augmentation depend on the complexity or inherent structure of the dataset or on the algorithm being used for specific task? What is the critical limit in percentage augmentation in this dataset that should provide best sentiment prediction as quantified in terms of performance metrics such as accuracy and F1 score?

We consider nine different classification models in our analysis where, three are the base machine learning models, and remaining six are deep neural networks models. The three base models are: Support vector classifier (SVC), Extreme gradient boosting (XGBoost) and Logistic regression (LR). Three other models are variants of LSTM networks: LSTM, LSTM concatenated to term-frequency inverse-document-frequency (*tf-idf*) vectors [44] and the ensemble of one dimensional convolution net (1D CNN) with LSTM likewise concatenated to (*tf-idf*). Similarly, last three models are variants of bidirectional LSTM networks: BiLSTM, BiLSTM concatenated to (*tf-idf*) and the ensemble of 1D CNN with BiLSTM [31] likewise concatenated to (*tf-idf*).

2 RELATED WORKS

Given the rapid growth in volume of data, text classification and sentiment analysis have been receiving surged attention [11, 54]. The approaches to these studies currently have three streams: lexicon-based methods, machine-Learning based methods, and hybrid methods. The lexicon-based approaches utilize existing lexicons to calculate the polarity of each word or phrase [53]. In contrast, machine learning techniques often involve statistical models to learn and improve from training set automatically [25]. Traditional machine learning techniques, such as Naïve Bayes Classifier, Support Vector Machine, and Logistic Regression, were extensively used in past years. Notably, various studies have shown that the hybrid method of lexicon and machine learning shows synergies. Neural networks and specifically, deep learning has progressively gained importance in this respect over the traditional machine learning approaches due to the capability of more accurate prediction over large volume of information [29].

Recent machine learning applications in NLP shows interesting results which in fact, has all been inputs to gain momentum to this proposed work. Monika et al. [35] explored the application of LSTM with GloVe word embedding in sentiment analysis on US airlines tweets from Twitter. The results show a high accuracy of over 80% and 75% for the training and validation sets, respectively. Shamal et al. [49] explored LSTM with the word, emoji and social acronym embedding (Token2vec) in sentiment analysis on Amazon product reviews which achieved an accuracy of 88.2% that is greater than 80% achieved using conventional techniques. Lu et al. [32] discussed sentiment analysis using LSTM incorporated with the lexicon and attention mechanism on two different data sets, namely, movie reviews and Stanford Treebank reviews. Their analysis included CNN, LSTM, and BiLSTM in which CNN achieved the highest accuracy of 81.5% for movie reviews, and LSTM combined with lexicon and attention outperformed other techniques in Stanford Treebank reviews.

Several recent machine learning applications in NLP involve attention-based models. Bai [4] explored the LSTM model with attention and convolution layer in text classification on Chinese Opinion Analysis Evaluation Microblog data set using RNN, CNN, and LSTM. The results show that proposed architecture of LSTM with additional attention layer, outperforms the other three algorithms as measured by precision, recall, and F1 scores. Liu and Guo [31] presented attention-based BiLSTM with a convolution layer (AC-BiLSTM) on seven different data sets. The experiments showed better performance in the proposed models compared to the baseline algorithms such as SVM, Naïve Bayes, CNN, and simple RNN. Even though there is an abundance of studies on text classification,

so far the applications of LSTM/BiLSTM is still lacking in sentiment analysis in large extent as compared to various traditional machine learning approaches.

There are various NLP applications of machine learning in finance. Khedr et al. [26] predicted whether stock prices would increase or decrease, using K-nearest Neighbor algorithm (KNN) on data sets that consolidated the movement (positive/ negative/ equal) in historical stock closing prices. A binary sentiment classification (positive/ negative) has been computed using Naïve Bayes approach and it is concluded that the performance of Naïve Bayes as classifier is superior to that of KNN and SVM which claimed that there is positive correlation between news sentiment and stock prices. Sohangir et al. [52] discussed how deep learning could improve the performance of financial sentiment analysis in the StockTwits data set. This paper focused on LSTM and CNN, while keeping logistic regression as the benchmark. The results in this work show 90.93% accuracy applying CNN which gives best performance among the three algorithms. Specific studies on news sentiment analysis towards economy, such as the ones we are reporting here is still by and large lacking. Pröllochs et al. [43] studied sentiment analysis of financial news by negation scope detection; Ranco et al. [45, 46] reported sentiment analysis in tweets and web browsed data towards stock price and intra-day price dynamics in trading.

Collins et al. [12] showed that not only the sophistication of the model architectures but also the difficulty level of the dataset is an important factor in the statistical performance of text classification. The difficulty level of a dataset maybe measured as a weight factor of the total number of words and labels in particular dataset. They studied text classification at various levels over 27 different open datasets, many of which are studied by different researchers. Their work apparently includes the current dataset that we are reporting here, however obtained from a different source (FigureEight 2018)[1]. They did a binary classification study in their dataset in terms of relevance of the news snippets towards economy using a number of classifiers along with all the other 26 datasets and then reported the *macro-F1* score [42] with respect to the level of difficulty in the datasets. The *macro-F1* score is designed to take into account the class imbalance issue in the dataset and hence more general in case the classes in the datasets are not balanced by adopting some measure. They used 128-dimensional FastText embedding [6] for their neural net models which are, MLP, LSTM-RNN, GRU-RNN, BiLSTM RNN, BiGRU-RNN. As for traditional classifiers, they applied KNN, Gaussian Naive Bayes, LR, Adaboost, Random Forest and SVC on *tf-idf* as the base vectors. In addition, they also reported a character based three layer CNN. Nonetheless, the classifiers' performance on various levels of *n-grams* were assessed to benchmark the difficulty in the datasets for text classification task. Note that Collins et al. [12] did not consider the multinomial classification task over the economic news snippet dataset, and to the best of our knowledge, no other previous study considered this dataset for text classification and sentiment analysis purposes.

[1]FigureEight is a human-in-the-loop machine learning and artificial intelligence company based in San Francisco, Los Angeles USA. Crowedflower has turned into FigureEight: https://appen.com/resources/datasets/

3 METHODOLOGY

3.1 Dataset

We perform our analysis with publicly available economic news relevance dataset [15]. The content in this dataset may be described as contributors reading snippets of news articles and then noting if the article has relevance to United States economy. Dataset contains these judgments as well as the dates, source titles, and text. This raw data set consists of 8,000 news articles from 1951 to 2014. Each observation, if found relevant to economy, is further judged on tone on a 9-point scale, where 9 represents the most positive relevance and 1 portrays the most negative one. In this work, the following description is adopted for convenience and references into coding: feature termed as "relevance" is used in text classification work while feature termed as "positivity" from the judgement on tone is identified as the 'sentiment' which is employed in our sentiment analysis study.

Figure 1 describes the details of the distribution of "relevance" and "positivity". As the first pie chart (a) shows, the data set is imbalanced where 82.1% does not have any relevance to economy amounting to a total of 6571 cases while 1420 cases are identified as having relevance to economy making it to be only 17.8%. There are only 9 cases of ambivalent "not sure" response. Collins et al. [12] proposed a quantitative measure of class imbalance as

$$imbalance = \sum_{1}^{C} \left| \frac{1}{C} - \frac{n_c}{T_{data}} \right| \tag{1}$$

where C is the total number of classes, n_c is the count of instances in class c, T_{data} is the total count of instances in the dataset. By this class imbalance statistics, zero signifies that all the classes have precisely equal number of instances i.e. perfectly balanced data and the upper bound value of 1 means that only one class has all the instances, i.e. 100% imbalanced data. In our dataset, if we consider the ambivalent "not sure" entries forming a class, then, for the 3 classes, the value of *imbalance* turns out to be 0.974 signifying a very highly imbalanced scenario which is unmanageable even by standard class balancing methodologies. As this class does not have enough statistical significance and is very small to consider, we preferred to remove it in subsequent work. Thereby, the text classification set is keeping only two classes thereafter as "yes" and "no" in terms of relevance to economy. Even on that consideration, *imbalance* turns out to be 0.644, which is a high measure in imbalance statistics. Such imbalance in class is obviously not quite appreciated for a reasonable predictive classification of unknown new inputs as test dataset.

Figure 1(c) and 1(d) shows the distribution histograms of the characters and words respectively in the original data set. The theoretical normal distribution (in black) overlay exhibits that these distributions can be considered as approximately normal with little deviation of the mean from standard normal. Among the 1420 relevant or "yes" cases, Figure 1(g) shows the sentiment labels on the scale of 2 to 9 where the actual number of corresponding records are imprinted therein. According to this pie chart of distribution within the "positive class", there is no news graded as 1, i.e. none are counted as the most negative one. Thereby, this part of the data which labels the tone as 'positivity' is quite diverse among eight different classes which are treated as the sentiments. Shannon

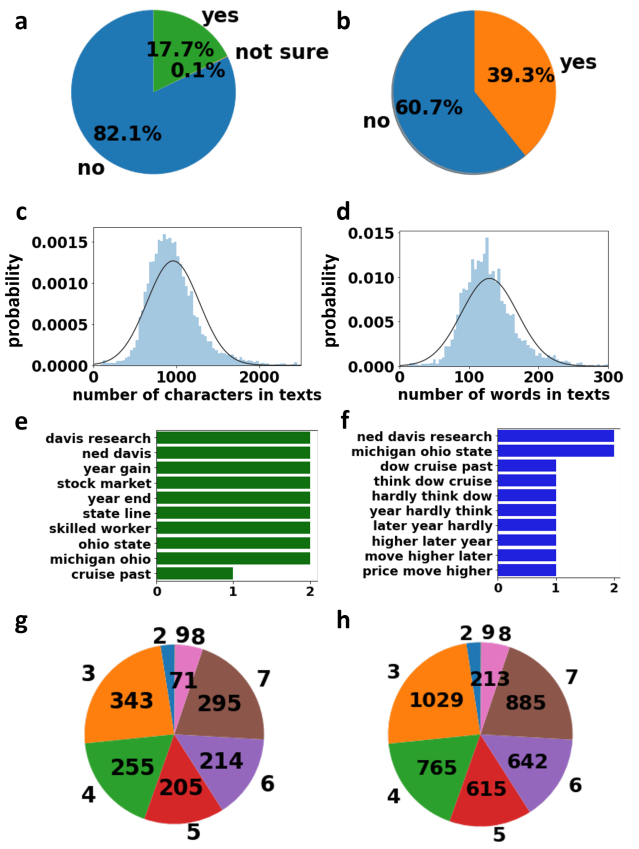


Figure 1: Class distribution in original and augmented data sets: (a) original financial news articles data [12, 15]; (b) Oversampling of the minority class to balance the classes for classification (described in text); (c and d) Distribution of characters and words in original data where the black line shows the theoretical normal curve; (e) The 10 most common bigrams and (f) 10 most common trigrams in the data set; (g) The sentiment Pie of the positive class in (a), including the number of samples in each class from 2 through 9; (h) Sample Pie with maximum augmentation of (g) while maintaining the class diversity intact.

[50] devised a statistical measure of class diversity in datasets in terms of entropy or the degree of randomness known as "Shannon diversity index" as follows:

$$H = -\sum_{i=1}^{R} p_i \ln p_i \qquad \text{and} \qquad E_H = \frac{H}{\ln R} \tag{2}$$

where R defines the "richness" of the data that corresponds to the number of classes, p_i is the probability of the ith class, H is the entropy as the measure of disorder or randomness in the data and E_H is the diversity index. The relationship indicates that the higher the chaos (or randomness) in the data values, higher is the diversity index. For the "yes" class data considered as the base for sentiment classification task in this work, the index turns out to be 0.86, which indicates that the sentiment set is highly diverse, implying that the dataset is highly challenging for the prediction task [12].

Table 1 shows a sample of the data used in text classification. The relevance to economy are indicated by various related terms such as 'certificate', 'deposits', 'money markets' etc. coupled to the names of different places in United States. The relevance is however indicated here only by the headlines avoiding the actual lengthy texts. Complete data set can be found online [2] along with the implementations. The negative relevance are indicated by the verbs such as 'fall' and 'decline'. As discussed earlier, these examples imply the inherent complexity in text classification problem that might turn out nontrivial with respect to differentiating by real meaning just from the common interpretation of each word. Table 2 gives a sample in terms of headlines of the sentiment within the relevant class of the classification dataset which, as mentioned earlier, has been labelled in a point scale of 2 to 9. The semantic analysis on how the grading has been done, is however beyond the scope of this work as the data is obtained as already labelled.

Table 1: Classification data sample

Relevance	Headline
yes	A Peek at Trucking Data, and Then the Stock Surged; Glimpses of Key Figures Can Aid Investors in Truck Stocks, Soybeans, Bed Makers and Others
no	In Europe, Job Protections for Older Generation Are Barriers for Younger Workers; Earnings Gap Looms for Younger Generation Dependent on Short-Term Contracts

There is no missing entry in the dataset for any feature. In addition to removing the ambivalent "not sure" class instances, following data cleaning steps are carried out: 1) *url* and *html* tags, special characters that carries no meaning, and the stop words are removed, 2) all the words are transformed to lower cases and 3) lemmatization is performed [22, 36, 38]. Stemming has also been performed, however it is observed that it reduces the percentage of words that can be explained by GloVe embedding, and therefore is not included under the pretext that GloVe embedding is already pre-stemmed. The vocabulary and text coverage are subsequently tested using two different pre-trained word vectors, namely, GloVe [41] and Fasttext [24]. The results show embedding coverage by both vectors to an extent of 98% texts, wherein GloVe vectors covered over 86% of unique words in the dataset which is almost 15% higher than what Fasttext could cover. Therefore, it is decided to use GloVe embedding for the purpose of model training, unlike Collins et al. [12], who used Fasttext in their work.

3.2 Models

In our analysis, we mainly focus on neural network-based methods for both text and sentiment classification tasks. As the baseline techniques however, SVC, XGBoost and Logistic regression [9, 14, 28, 40]

[2] https://github.com/jianning1

Table 2: Sentiment data sample

Positivity	Headline
2	How To Play Rising Rates; The Bond Boom Has Left Many Investors Vulnerable to a Surprise Jump in Interest Rates. Here's How to Protect Yourself–and Profit
3	Currency Trading: Dollar Remains in Tight Ranges Amid Wait for U.S. Jobs Data
4	U.S. Dollar Falls Against Most Currencies; Decline Is Softened as Bond Rally Stalls
5	A Season of Doubt Grips Wall Street; With Economy Soft, Insiders See No Set Direction This Fall
6	Fed's Greenspan Refuses to Accept Blame for Recession; Upturn's Pace Is 'Glacial'
7	Tech Sector in Hiring Drive; Google, Intel Add Workers as Profits Snap Back; Start-Ups Also Fight for Talent
8	New home sales rose 7.1% in May; Personal income and spending up 0.3%
9	Wanted: Employees in Michigan and Ohio — States Compete to Lure Skilled Workers Amid Shortage
note:	**1: most negative; 9: most positive**

are considered as well. In addition to these traditional methods, LSTM, BiLSTM and an ensemble of 1D CNN with LSTM/BiLSTM has been employed wherein, the text classification is binary and the sentiment classification is multinomial.

Basic neural networks such as feed forward neural networks (e.g. multilayer perceptron (MLP)) consists of input and output layers organized by interconnected nodes. Every node in output layers formulates outputs using inputs from nodes in the previous layers, together with some new inputs. The weights are optimized at output layers for the purpose of obtaining the best results. Besides input and output layers, neural networks consisting of multiple hidden layers are often referred as deep neural networks [37, 47]. Two of the most well acclaimed examples in deep neural networks are CNNs and RNNs. Their main difference is that CNNs share weights within the same layer, while RNNs use the same weights between layers, resulting in faster training speeds [37, 51]. The speciality in RNNs is the use of memory cells which receive the output from the previous layer as an input, thus causing effective reduction in loss of information. However, as the transitive memory is diluted after every iteration, RNN is efficient to learn from words that are within close proximity. For the purpose of having a semantic understanding of the whole sentence, the system often requires information from much distant words (i.e. earlier layers) in which RNNs are not efficient. A special network built upon RNN is LSTM which are constructed to resolve such issues. It retards the dilution of important information under the impact of forget gate-layers [37, 56] as opposed to regular RNN unit which has single layer. Significant improvements are accomplished through LSTM in the predictive analytic of various types of textual data in which word sequence is important. However, LSTM has the drawback that it does not consider the existence of backward dependencies [37]. BiLSTM networks architecture has been developed to include both forward and backward dependencies [48]. The architectural distinction between LSTM and BiLSTM is shown in schematics in Figure 2.

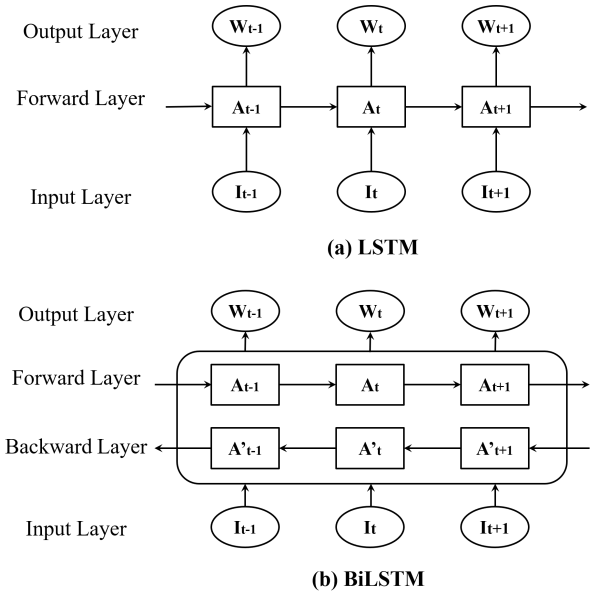


Figure 2: Schematic of LSTM and Bi-SLTM layer structures [31].

In this work, we explore both LSTMs and BiLSTMs as well as using these methods in tandem with 1D-CNN to explore the effect of sophistication in algorithmic architecture in text and sentiment classifications. Figure 3 shows the neural network architecture of 1D CNN concatenated with LSTMs.

3.3 Performance evaluation

In our experiments, we used 10-fold cross validation, as it is one of the most commonly used techniques that allows for computing predictions in an accurate way and helps in reducing the biased estimations [18]. Afterward, the performance of the multinomial and binary classifiers has been measured using recall and precision as follows [40]:

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \tag{3}$$

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \tag{4}$$

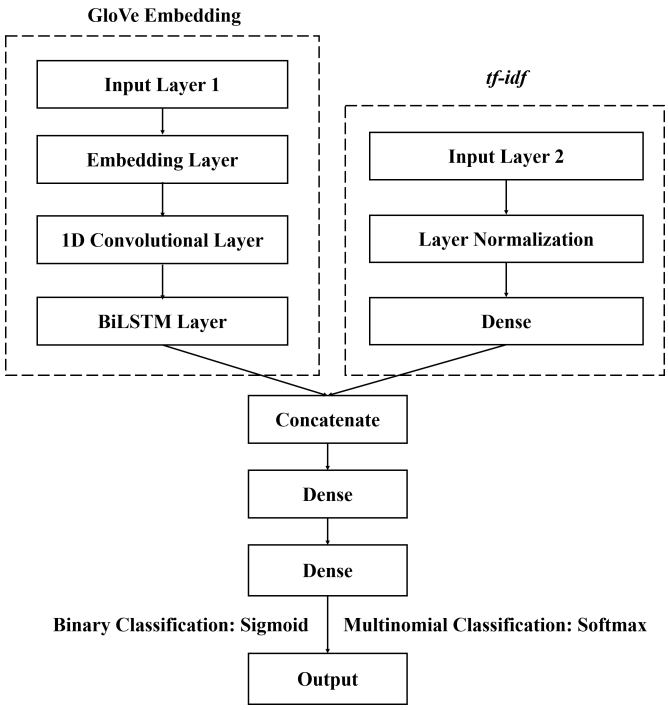


Figure 3: The (1D CNN + BiLSTM) neural network scheme concatenated to *tf-idf*

where True Positives are the instances that have been correctly retrieved by the employed classifiers, False Positives are the instances that have been incorrectly classified which in reality are the negatives, and False Negatives are the actual positives that the classifier has mistakenly predicted as the negatives. In order to have a sum-up and represent the precision and recall in terms of one value, we also present the harmonic mean of recall and precision as the $F1$ score given by [40]:

$$F1 = \frac{2}{\frac{1}{Recall} + \frac{1}{Precision}} \tag{5}$$

In addition, the area under curve (AUC) from typical Receiving Operator Characteristics (ROC) as defined by the ratio of true positive rate (TPR) to false positive rate (FPR) is also reported. The Cohen Kappa score [34, 40] defining the strength of agreement is calculated in binary text classification as well.

4 RESULTS

We performed a detailed numerical study to evaluate the performance of various machine learning models for the text and sentiment classification tasks. We then explored the impact of data augmentation on model performances. The analysis with standard supervised learning methods and deep learning methods were conducted using scikit-learn and tensorflow packages in python, respectively, and the GPU support in Google Colab platform is utilized for the deep learning models [3, 7, 10, 39].

4.1 Exploratory analysis and augmentation

Total texts have a minimum of 83 to a maximum of 3379 unique characters as rendered by a minimum of 10 to a maximum of 432 unique words. Most of the articles have 500 to 1500 characters against 60 to 200 words. Figure 1(e) and 1(f) exhibit the 10 most common bigram and trigram (n-gram) in the data set. Note that dataset is highly class imbalanced having an imbalance statistic of 0.644 with two classes (see equation (1)). To deal with this issue, we decided to oversample the minority "yes" class by the use of synonyms importing "wordnet" from Natural Language ToolKit (NLTK) library [5]. Such oversampling by two times results in a reasonably balanced class distribution as shown in Figure 1(b) where 39.3% of the dataset represents 4260 positive (or 'yes') class records. By doing this data augmentation, imbalance statistic given by *imbalance* turns out to be 0.355 which is a reasonable figure to perform the text classification. The pie chart in Figure 1(b) shows the minority oversampled balanced two class data that is subsequently used in this task.

The pie chart in Figure 1(g) shows the sample proportion of sentiment by 'positivity' ranking on the point scale of 2 to 9 within the original 17.8% "yes" class shown in Figure 1(a). This constitutes the baseline data for the sentiment classification study. However, here the total number of instances 1420 diversified over eight different sentiment labels, and accordingly, it is very small to train a well performing classification model. Therefore, we adopt the same strategy of data augmentation by using synonyms similarly but keeping the "diversity index" unaltered as measured by equation (2). This is because this index (0.86) is a measure of the difficulty level in the sentiment data which is treated as a pre-factor to judge the performance in terms of sophistication of the algorithms applied. Thereby, the starting dataset in the sentiment classification is 20% augmented over what is shown in Figure 1(g). We then exercise successive augmentation from 20% to a maximum of 200% in eight steps in order to study the critical limit in such augmentation, measured in terms of performance scores. The pie chart in Figure 1(h) on that score shows the 200% (maximum) such oversampled data over what is shown in Figure 1(g), with the proportion of data instances inscribed therein as distributed in eight different classes. In addition, Figure 1(h) represents the 'positivity' instance divisions of the 39.3% "yes" data shown in Figure 1(b). It might quite plausibly be asked on the legitimacy of such introduction of artificiality. Nevertheless, for text data, as long as the contextual semantic of the specific text is not disturbed, such proportional augmentation does not overrule the objective of the task as far as the lexicon boundaries of English language is concerned.

4.2 Text classification

SVC [14], XGBoost [9] and Logistic regression [28] are chosen as the baseline models in text classification study with *tf-idf* vectors where relevance to economy is the class; 0 stands for non relevant and 1 for relevant. The minority (class) oversampled data presented in Figure 1(b) is the input to all text classification experiments. All calculations are done using 10 fold cross validation. The performance of the base models measured in terms of four metrics, namely, accuracy, F1, AUC, and Cohen Kappa, are given as bar graph in Figure 4 which shows approximately competitive scores

wherein logistic regression slightly edges over the others at 87% accuracy in prediction.

Having these baseline performances, to further improve the text classification, we apply LSTM and BiLSTM. GloVe embedding is used in all LSTM and BiLSTM layers, initially. Following that, (1) *tf-idf* is concatenated to LSTM and then (2) to BiLSTM. Word embedding weights are adopted from GloVe embedding vector [41]. The final network layering structure is as follows: LSTM and BiLSTM layers are first employed on top of GloVe embedding layer with one 1D Global Average Pooling layer, followed by one dense layer with rectified linear unit (ReLU) as the activation function. In the dense output layer, Sigmoid activation function is applied for text classification as we consider a binary classification task [20].

Figure 4 shows that, when compared to the performance of the baseline models, LSTM and LSTM+TF_IDF do not provide a significant performance improvement. Concatenating the LSTM/BiLSTM layer on GloVe embedding with normalized *tf-idf* vectors shows advantage in the case of BiLSTM model where F1 score is improved by a margin of 3%. It is then explored to determine any advantage by using an ensemble of convolution net (CNN) with LSTM and BiLSTM. Specifically, 1D CNN layer is added preceding the LSTM/BiLSTM structure. The results showed that this approach indeed enhances the overall performance score raising F1 by another margin of 0.5%.

The hybrid approach with LSTM is in fact considered as providing the best performance in text classification of which the characteristic training and validation curves are shown in Figure 5. The learning rate has been optimized through a wide range from 10^{-8} to 10^{-3} as shown in Figure 5(i). The optimum learning rate in training here is $5x10^{-5}$. The training has been undertaken employing early stopping method to ward off overfitting issues which is rather prevalent in these calculations when the data size is relatively small. Accuracy and validation accuracy over the early stopped epochs are shown in Figure 5(ii) while the levelling off of validation loss and mean absolute error (MAE) are shown in Figure 5(iii). Even though, the numeric validation scores of 1D CNN with BiLSTM is about 2-3% more than 1D CNN with LSTM (see Figure 4), there is training overfitting in the former as compared to that in the latter. Thereby, in this binary classification, it is rather considered that counting on backward dependency by employing BiLSTM does not truly provide substantial improvements over LSTM network which could be a disadvantage in having a relatively smaller data set in neural network applications.

4.3 Sentiment classification

Due to significant data imbalance, the sentiment classification task requires data augmentation. The starting data for sentiment analysis is the 20% augmented data over what is shown in Figure 1(g). All nine algorithms as in text classification are then applied at each of the eight steps of data augmentation to answer our research question of determining the critical limit in such augmentation by measuring in terms of performance scores. Such detailed measurements over the exhaustive repetitive runs are performed after similar learning rate optimization as before and the optimized learning rate of training in this case is $2x10^{-4}$, as shown in Figure 6(1). The basic layering structure here remains the same, however Softmax is used as the activation function in output dense layer because it is a multinomial classification task [20]. The same approach of using normalized *tf-idf* concatenated to LSTM/BiLSTM shows here an improvement in F1 score by a margin of 13% when used in BiLSTM (i.e. BiLSTM + *tf-idf*) as compared to when used in LSTM.

The performance scores measured in terms of accuracy, F1 and AUC are shown with respect to eight stages of percentage augmentation in Figure 7. These results show that, for this multiclass classification task, BiLSTM renders a substantial edge over LSTM and the best performance here is given by the ensemble of 1D CNN with BiLSTM which is winning over a margin of 30% in both accuracy and F1 scores than 1D CNN with LSTM. The AUC coverage is also the highest at 82%. The architecture of layering of the ensemble method is the same as shown in the common Figure 3 with the use of Softmax as the activation function for the output layer (labelled on the right side). Clearly, the backward dependency taken into account in BiLSTM provides quite substantial advantage even for a relatively smaller dataset when it is a more complex task of classification over the dataset with higher level of difficulty as attributed by the Shannon diversity index [50]. The results also reveal on the other hand that, there is indeed a critical limit in data augmentation. We stretched augmentation to an extreme of 200%, however 180% seems to be the maximum limit whereupon the performance starts decreasing suggesting the critical limit for this data set. It is obvious that any better performance in such task even with texts certainly depends on having more actual amount of data in the corpus. Nonetheless, this exercise tries to portray one side of the reality with certain obvious shortcomings in terms of marginal overfitting issues as maybe observed in Figure 6(2) and (3) in accuracy and MAE of the validation set. An optimum seven fold cross validation is done each time across all algorithms in this case and the scores shown in Figure 7 are the averages over all cross validations carried out using the optimized learning rate. Further tuning of the hyperparameters actually did not render any better promise in performance that a richer data volume could have. It might be projecting that the exercise in text and sentiment predictive classification in this dataset has resulted the scores to its eventual limits.

5 THREATS TO VALIDITY

Internal Validity. We believe the measures in our study are acceptable as we employed well acclaimed Python libraries, Tensorflow and Keras as available on free Anaconda distributions as well as Google Colab (colab.research.google.com).

External Validity. The dataset used in this study is freely available as open source at the website of *data.world* resourced from *Crowdflower.com* under 'economic-news-article-tone'.

Construct Validity. The methodologies used are not developed in this study. Their wide acclaimed use in machine learning and statistical studies by many researchers reduce the subjective bias on judgment. Data augmentation follows the standard practice adopted in machine learning research on NLP.

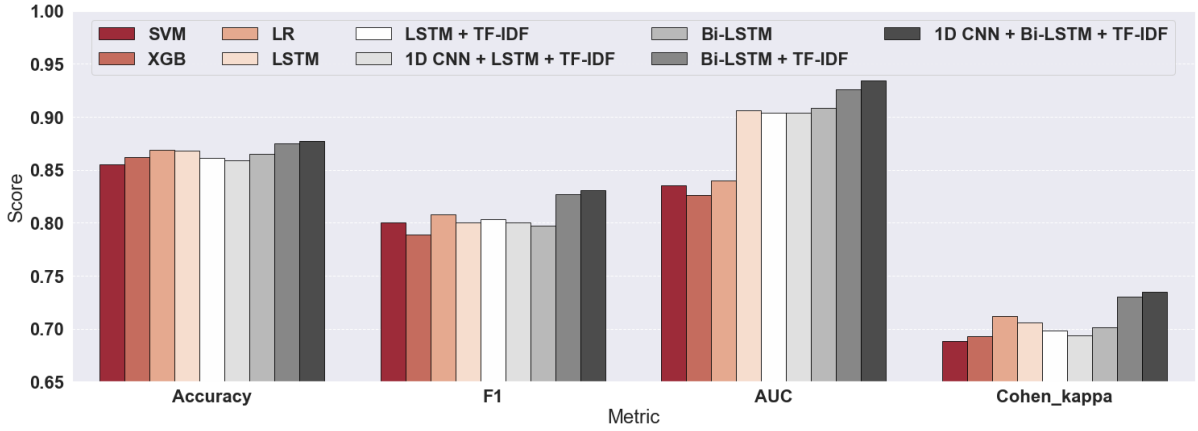


Figure 4: Four different performance measures across all models for text classification. The best performance is obtained by the ensemble method of (1D CNN + LSTM) with *tf-idf*

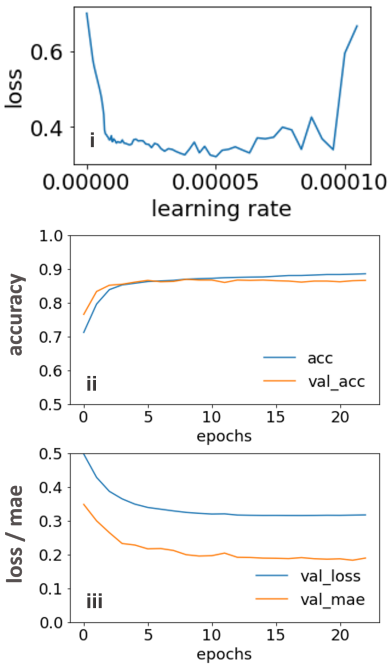


Figure 5: Hyperparameter tuned best model for text classification. (i) Learning rate optimization with Loss function; (ii) Accuracy and validation accuracy best performance by the ensemble method of (1D CNN + LSTM) with *tf-idf*; (iii) Validation loss and Mean absolute error (MAE) for the same model.

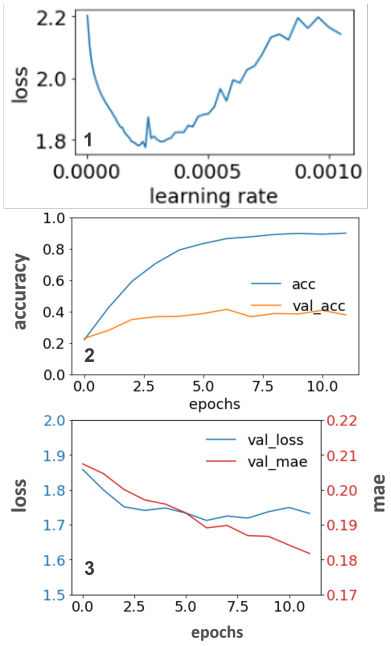


Figure 6: Hyperparameter tuned best model for multinomial sentiment classification. (1) Learning rate optimization with Loss function; (2) Accuracy and validation accuracy best performance by the ensemble method of (1D CNN + BiLSTM) concatenated to *tf-idf*; (3) Validation loss and Mean absolute error (MAE) for the same model.

6 CONCLUSION

We commenced this machine learning work in text classification and sentiment analysis with the objectives:

- Predicting the relevance of news snippets towards economy by a binary text classification and,
- Predicting the sentiment within the economy relevant news snippets labelled on a "positivity" scale over eight different classes based on their tones.

In the course of these studies, we also asked a question in view of the standard practice of data augmentation in textual data: is there a critical limit of augmentation as measured by statistical

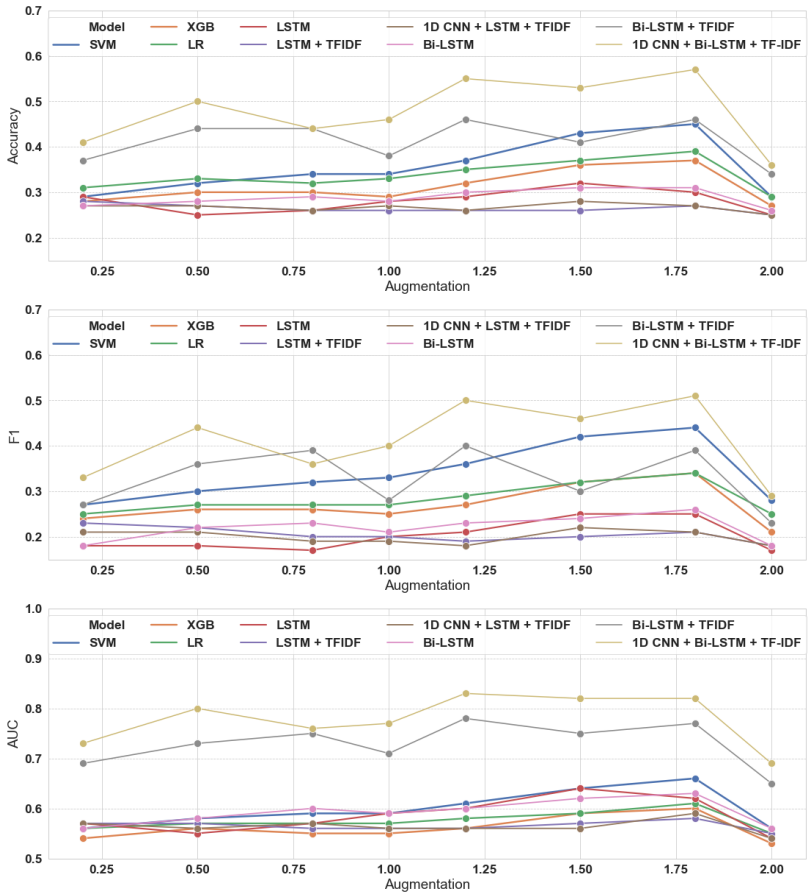


Figure 7: Detailed performance measures (Accuracy, F1 score and AUC) for multinomial sentiment classification across all models. The best performance is obtained by the ensemble method of (1D CNN + BiLSTM) concatenated to *tf-idf*.

performances of the classifier? In addition, on a secondary objective, we also tried to reflect on the role of sophistication of the algorithm in text classification in data with higher level of difficulty.

We applied nine different classifiers where three are the traditional machine learning and rest six are neural network based algorithms. As this open source dataset is relatively small in size and dimension, we adopted data augmentation by oversampling with synonyms from well acclaimed NLTK library and then studied the effect of such augmentation in terms of various statistical performances. With respect to this dataset, it would be pertinent to summarize our findings as follows:

(1) The ensemble neural network method of 1D CNN with LSTM/BiLSTM provides the best performances. In binary text classification, it is 1D CNN with LSTM concatenated to *tf-idf* and in multinomial sentiment classification, 1D CNN with BiLSTM concatenated to *tf-idf* turns out to be the best for this dataset.
(2) Even though data augmentation is quite legitimate in practice in textual data, there maybe a limit to the extent in such augmentation which might depend on the particular dataset and the task it is employed to. In our case, the critical limit turns out to be 180% whereupon, performance starts decreasing.
(3) The higher level of sophistication in BiLSTM architecture which counts on backward dependency, does provide an edge over LSTM when the dataset has higher level of difficulty (as measured by Shannon diversity index) implying that the task is more complex.

There are several future research directions from this study. We plan to test out the presented methods using other economic news related datasets. In addition, other recent deep learning architectures and word embeddings such as BERT can be explored for text classification and sentiment analysis over economic news datasets [17].

REFERENCES

[1] [n.d.]. Introduction to news analytics. https://www.eventstudytools.com/introduction-news-analytics. Accessed: 2020-06-22.
[2] [n.d.]. News feeds, analytics, and indices. https://www.refinitiv.com/en/products/world-news-data. Accessed: 2020-06-22.
[3] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, and et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. http://tensorflow.org/ Software available from tensorflow.org.
[4] Xuemei Bai. 2018. Text classification based on LSTM and attention. In *2018 Thirteenth International Conference on Digital Information Management (ICDIM).*

IEEE, 29–32.
[5] Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.".
[6] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv* (2016), preprint arXiv:1607.04606.
[7] Tiago Carneiro, Raul Victor Medeiros Da Nóbrega, Thiago Nepomuceno, Gui-Bin Bian, Victor Hugo C De Albuquerque, and Pedro Pedrosa Reboucas Filho. 2018. Performance analysis of google colaboratory as a tool for accelerating deep learning applications. *IEEE Access* 6 (2018), 61677–61685.
[8] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.
[9] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.
[10] François Chollet et al. 2015. Keras. https://keras.io.
[11] Poonam Choudhari and S Veena Dhari. 2017. Sentiment Analysis and Machine Learning Based Sentiment Classification: A Review. *International Journal of Advanced Research in Computer Science* 8, 3 (2017).
[12] Edward Collins, Nikolai Rozanov, and Bingbing Zhang. 2018. Evolutionary Data Measures: Understanding the Difficulty of Text Classification Tasks. *arXiv preprint arXiv:1811.01910* (2018).
[13] Alexis Conneau, Holger Schwenk, Loıc Barrault, and Yann Lecun. 2017. Very deep convolutional networks for text classification. *In Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics* 1 (2017), 1107–1116.
[14] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning* 20, 3 (1995), 273–297.
[15] CrowdFlower. 2015. Economic News Article Tone, https://data.world/crowdflower/economic-news-article-tone. (Dec 2015). Dec 2015.
[16] Sanjiv R Das. 2011. News analytics: Framework, techniques and metrics. *The Handbook of News Analytics in Finance* 2 (2011).
[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:cs.CL/1810.04805
[18] Tadayoshi Fushiki. 2011. Estimation of prediction error by using K-fold cross-validation. *Statistics and Computing* 21, 2 (2011), 137–146.
[19] Maya R Gupta, Samy Bengio, and Jason Weston. 2014. Training highly multiclass classifiers. *The Journal of Machine Learning Research* 15, 1 (2014), 1461–1492.
[20] Geoffrey E Hinton and Russ R Salakhutdinov. 2009. Replicated softmax: an undirected topic model. In *Advances in neural information processing systems*. 1607–1614.
[21] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
[22] Matthew Honnibal and Ines Montani. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. (2017). To appear.
[23] K Sparck Jones. 2001. Natural language processing: a historical review. *University of Cambridge* (2001), 2–10.
[24] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of Tricks for Efficient Text Classification. *arXiv preprint arXiv:1607.01759* (2016).
[25] Tarek Kanan, Odai Sadaqa, Amal Aldajeh, Hanadi Alshwabka, Shadi AlZu'bi, Mohammed Elbes, Bilal Hawashin, Mohammad A Alia, et al. 2019. A review of natural language processing and machine learning tools used to analyze arabic social media. In *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*. IEEE, 622–628.
[26] Ayman E Khedr, Nagwa Yaseen, et al. 2017. Predicting stock market behavior using data mining technique and news sentiment analysis. *International Journal of Intelligent Systems and Applications* 9, 7 (2017), 22.
[27] Max R Kimbrough, Steven O Kimbrough, and Priscilla Murphy. 2011. On using text analytics for event studies. In *Proceedings of the 13th International Conference on Artificial Intelligence and Law*. 209–218.
[28] David G Kleinbaum, K Dietz, M Gail, Mitchel Klein, and Mitchell Klein. 2002. *Logistic regression*. Springer.
[29] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.
[30] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural Computation* 1, 4 (1989), 541–551.
[31] Gang Liu and Jiabao Guo. 2019. Bidirectional LSTM with attention mechanism and convolutional layer for text classification. *Neurocomputing* 337 (2019), 325–338.
[32] Yifei Lu, Yanghui Rao, Jun Yang, and Jian Yin. 2018. Incorporating Lexicons into LSTM for sentiment classification. In *2018 International joint conference on neural networks (IJCNN)*. IEEE, 1–7.
[33] G Harry Mc Laughlin. 1969. Smog grading-a new readability formula. *Journal of reading* 12 (1969), 639–646.
[34] Mary L McHugh. 2012. Interrater reliability: the kappa statistic. *Biochemia medica: Biochemia medica* 22, 3 (2012), 276–282.
[35] R Monika, S Deivalakshmi, and B Janet. 2019. Sentiment Analysis of US Airlines Tweets Using LSTM/RNN. In *2019 IEEE 9th International Conference on Advanced Computing (IACC)*. IEEE, 92–95.
[36] Travis E Oliphant. 2006. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA.
[37] Daniel W Otter, Julian R Medina, and Jugal K Kalita. 2020. A Survey of the Usages of Deep Learning for Natural Language Processing. *IEEE Transactions on Neural Networks and Learning Systems* (2020).
[38] The pandas development team. 2020. *pandas-dev/pandas: Pandas*. https://doi.org/10.5281/zenodo.3509134
[39] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research* 12 (2011), 2825–2830.
[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[41] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. 1532–1543. http://www.aclweb.org/anthology/D14-1162
[42] David Martin Powers. 2011. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *Journal of Machine Learning Technologies* 2, 1 (2011), 37–63.
[43] Nicolas Pröllochs, Stefan Feuerriegel, and Dirk Neumann. 2015. Enhancing sentiment analysis of financial news by detecting negation scopes. In *2015 48th Hawaii International Conference on System Sciences*. IEEE, 959–968.
[44] Juan Ramos et al. 2003. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, Vol. 242. Piscataway, NJ, 133–142.
[45] Gabriele Ranco, Darko Aleksovski, Guido Caldarelli, Miha Grčar, and Igor Mozetič. 2015. The effects of Twitter sentiment on stock price returns. *PloS one* 10, 9 (2015).
[46] Gabriele Ranco, Ilaria Bordino, Giacomo Bormetti, Guido Caldarelli, Fabrizio Lillo, and Michele Treccani. 2016. Coupling news sentiment with web browsing data improves prediction of intra-day price dynamics. *PLoS one* 11, 1 (2016).
[47] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural networks* 61 (2015), 85–117.
[48] Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing* 45, 11 (1997), 2673–2681.
[49] Achira Jeewaka Shamal, Rankothge Gishan Hiranya Pemathilake, Sachith Paramie Karunathilake, and Gamage Upeksha Ganegoda. 2018. Sentiment Analysis using Token2Vec and LSTMs: User Review Analyzing Module. In *2018 18th International Conference on Advances in ICT for Emerging Regions (ICTer)*. IEEE, 48–53.
[50] Claude Elwood Shannon. 2001. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review* 5 (2001), 3–55.
[51] Richard Socher, Eric H Huang, Jeffrey Pennin, Christopher D Manning, and Andrew Y Ng. 2011. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in neural information processing systems*. 801–809.
[52] Sahar Sohangir, Dingding Wang, Anna Pomeranets, and Taghi M Khoshgoftaar. 2018. Big Data: Deep Learning for financial sentiment analysis. *Journal of Big Data* 5, 1 (2018), 3.
[53] Peter D Turney. 2002. Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews. In *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics, 417–424.
[54] Shuo Xu. 2018. Bayesian Naïve Bayes classifiers to text classification. *Journal of Information Science* 44, 1 (2018), 48–59.
[55] Dani Yogatama, Chris Dyer, Wang Ling, and Phil Blunsom. 2017. Generative and discriminative text classification with recurrent neural networks. *arXiv* (2017), preprint arXiv:1703.01898.
[56] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. 2018. Recent trends in deep learning based natural language processing. *IEEE Computational intelligenCe magazine* 13, 3 (2018), 55–75.
[57] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. *Advances in neural information processing systems* (2015), pages 649—-657.

Voting for Authorship Attribution Applied to Dark Web Data

Britta Sennewald
britta.sennewald@unb.ca
University of New Brunswick
Fredericton, Canada

Rainer Herpers
rainer.herpers@h-brs.de
University of Applied Sciences Bonn-Rhein-Sieg
Sankt Augustin, Germany

Marco Hülsmann
marco.huelsmann@h-brs.de
University of Applied Sciences Bonn-Rhein-Sieg
Sankt Augustin, Germany

Kenneth B. Kent
ken@unb.ca
University of New Brunswick
Fredericton, Canada

ABSTRACT

This research is about authorship attribution (AA) within multiple Dark Web forums and the question of whether AA is possible beyond the boundaries of a single forum. AA can become a curse for users that try to protect their anonymity and simultaneously become a blessing for law enforcement groups that try to track users. In this paper, we explore AA within multiple Dark Web forums to determine whether AA is possible beyond the boundaries of a single forum. The analysis revealed that analyzing all features together with a single classifier does not achieve as good results as when they are classified separately and the final result is computed by a voting mechanism. The latter achieves an F1-Score that is up to 44% higher than in the former case. On top of that, the analyses show that the author of a post is at least 94% within the top three most likely candidates. This shows that AA can threaten the anonymity of Dark Web users across the boundaries of different forums.

CCS CONCEPTS

• **Security and privacy**; • **Computing methodologies** → **Machine learning**; **Natural language processing**;

KEYWORDS

Authorship Attribution, Dark Web, Machine Learning, Natural Language Processing, Voting

ACM Reference Format:
Britta Sennewald, Rainer Herpers, Marco Hülsmann, and Kenneth B. Kent. 2020. Voting for Authorship Attribution Applied to Dark Web Data. In *Proceedings of 30th Annual International Conference on Computer Science and Software Engineering (CASCON'20).* IBM Corp., Riverton, NJ, USA, 10 pages.

1 INTRODUCTION

Authorship attribution (AA) focuses on assigning documents to their corresponding authors. This is very successful when a sufficient amount of text is available. If the length of a text and/or the number of texts per author is small, it becomes increasingly challenging to attribute them correctly to an author [23]. Nonetheless, several researchers e.g., S. R. Pillay et al. [18] or R. Layton et al. [11] focused especially on this typical problem of online texts. It applies to texts within the openly accessible internet (Surface Web) as well as for the Dark Web. The latter is accessible via special technologies such as The Onion Router (TOR), which are becoming increasingly popular [1]. It offers the opportunity to access the internet anonymously, which is interesting for users who want to protect their privacy or circumvent censorship on the internet, but also for those who do not want to be identified when performing illegal activities [21]. Therefore, AA transferred to the Dark Web implies, that posts within Dark Web forums could be assigned to their authors, and thus it means that AA can be a danger to their anonymity.

A good AA algorithm is of interest to law enforcement agencies who, e.g., want to track or find users engaged in illegal activities shown by scientists like M. Yang et al. [25] or M. Sultana et al. [22]. However, the same algorithm can be used to identify users who are dependent on the anonymity of the Dark Web to be able to express their opinions freely and thus avoid the suppression of regimes. Therefore AA can be both a way to track criminals, as well as a danger to privacy on the Dark Web. For these two reasons, it is interesting to investigate how precise posts can be assigned to their authors on the Dark Web, especially between multiple forums.

In this research, four different Dark Web forums are crawled/-scraped to apply AA to posts published by authors that are active in two of these Dark Web forums. For this purpose, different techniques like Natural Language Processing (NLP) and Machine Learning (ML) are used. By doing so, it is possible to determine to what extent posts can be attributed to their correct author, regardless which of the two forums the posts were published within or which username was used. A very important role in this context will also be played by the application of a voting classifier, which will be used in addition to the normal classifiers.

Since usernames within forums can be chosen freely by everybody they are not a reliable way to link user profiles. Hence, within this research, another way that provides more reliability was chosen as a ground truth: Pretty Good Privacy (PGP)- keys that are often used for confidentiality reasons within Dark Web forums and marketplaces. Not all users use the same PGP-key within two or more forums, however, users that do so are linkable by this feature.

CASCON'20, November 10–13, 2020, Toronto, Canada

2 RELATED WORK

Authorship attribution is a difficult task in an environment with hundreds or even thousands of different authors and texts that differ greatly in size. Many researchers already focused on authorship attribution and also on the Dark Web, but only a few researchers have concentrated on a combination of both. Two of them are Ho and Ng, which analyzed stylometric features of texts posted in different Dark Web forums [9]. They tried to connect ten authors within Dark Web forums by extracting stylometric-based features and special *fingerprints* like typical words or typos of an author. Within their analysis, Ho and Ng focused on Support Vector Machine (SVM) classifiers only.

Another research project regarding Dark Web and authorship attribution was undertaken by Spitters et al. [21]. They achieved their results by using a combination of time-based and stylometric features as well as character trigrams. For the classification, they also used Support Vector Machines just as Ho and Ng did [9]. By using all three features together, the actual author was ranked first with a probability of 88% and within the top five most highly ranked candidates with a probability of 97%.

Forum posts also provide another interesting feature for AA, the time when a post was posted. Even if the research of La Morgia et al. [15] is not in the context of authorship attribution, it shows that only the activity of a user can reveal his/her location. The authors analyzed the activity of users of five different dark web forums. In the case of two of them, they already knew where most of the users were coming from. Hence, their goal was to figure out where the users of the other three forums were coming from. To obtain a ground truth regarding how the activity of users appears around the world, they used a Twitter data set with the known origin of all users. After that, they were able to distinguish groups of dark web users from different regions around the globe within a crowd just by analyzing the timestamps of posts. Therefore it can be concluded, that the activity of users that are living in a different timezone is different regarding the time. Hence, this feature might be especially useful when analyzing forums with a global community.

Ashcroft et al. tried to match multiple aliases of the same user within a data set of an Irish web forum and within a Twitter dataset [2]. This research did not focus on the Dark Web environment, but tried to match users within different domains by using AA techniques. The authors used stylometric and time-based features, but they also added so-called emotion-based or Twitter-specific features to their analysis. Within their analysis, the authors used three different machine learning classifiers AdaBoost, Support Vector Machine, and Naive Bayes (NB).

Linking users within different Dark Web forums is one of the main tasks in this research and is quite challenging. Fortunately, although not in the context of authorship attribution, Me, Spagnoletti and Pesticcio focused on the relationship between different TOR marketplace users by concentrating on their PGP-keys [12]. Within Dark Web marketplaces, PGP-Keys are often used to provide confidentiality within transactions. Hence, the public key of vendors and buyers can often be found in their user profile. Even if the original research problem of this paper does not belong to AA, it points out that Dark Web users are using PGP keys for their transactions and can be linked by these keys.

Pillay and Solorio also worked on AA of Web Forum posts [18]. They used stylometry features (lexical and syntactical), statistical language models, clustering, and different machine learning algorithms in their work. Clustering was applied to use the output as meta-features for identifying the authors. The results show, that their approach is able to classify posts of five authors with a probability of around 90% correctly by using BayesNet. However, by an increasing number of authors, the C4.5 algorithm to create a decision tree seems to be the better choice than BayesNet. Besides that, the authors observed that when the number of authors increased, those classifiers that incorporate a cluster identifier worked best.

Swain, Mishra, and Sindhu give an overview of recent approaches to AA techniques [23]. They list multiple research projects focusing on this area, especially the category, language, domain, features, and techniques that have been used. This survey shows that Naive Bayes and Support Vector Machine are the most commonly used classifiers, English the most frequently analyzed language and lexical and syntactic features the most popular features.

3 DATASET

One of the most important parts of this research was to create a suitable dataset for AA analysis. Thus the following section focuses on how this dataset was created, which Dark Web forums were used, and which features were extracted.

3.1 Creation of the Dataset

Developing a crawler for the Dark Web is not as straightforward as for the Surface Web. Therefore, in this section, the most important key aspects for crawling/scraping Dark Web forums that were essential for creating this dataset are briefly described. Some of them are strongly influenced by Gwern Branwen's experiences when creating his Dark Web dataset between 2012 and 2015 [6].

First of all, connecting to a Dark Web website is completely different than connecting to a Surface Website. The TOR network is accessible over the TOR browser or a TOR proxy that can be used by a crawler. To increase the speed of the crawling process multiple TOR proxies were set up to allow multiple TOR sessions in parallel.

However, Dark Web websites tend to protect themselves from being crawled or attacked, e.g., by a Distributed Denial of Service (DDoS) attack [14], more rigorously than websites on the Surface Web. Therefore, the timing of requests and thus the speed of the crawler is an extremely challenging task within the Dark Web. Too many requests per minute might lead to a detection of the crawler, whereas too few requests will result in a slow crawl. Thus, a trade-off between both factors needs to be found, which unfortunately is a trial and error process.

In addition, most Dark Web forums do not allow users to access the content of the forum without being registered. Fortunately, having an account on a Dark Web forum is often linked with the possibility to adjust the settings for the forum's outward appearance. For example, choosing the highest possible number of posts per page will result in a faster crawl as there are fewer pages to crawl. Furthermore, an approach that contains blacklisting as well as whitelisting needs to be set up, e.g., to avoid accidentally being logged out during the crawling process. The last important aspect was to separate automatic and manual processes as all forums

Table 1: Statistics of all four crawled Dark Web forums, showing an overview of the dataset used in this thesis. The number of PGP-Key owners within TMG refers to only those that could be matched.

Forum name	*Number of posts*	*Number of users*	*Number of users with PGP Key*	*Total number of files crawled*
DNMA	75,165	10,489	277	20,645
TH	225,135	26,502	1,900	55,106
TMG	201,538	5,121	193+	15,678
Dread	385,839	63,299	2,943	163,943

require solving a captcha when signing in, or logging in. Since the captchas were not easy to solve automatically, this had to be done manually, whereas the final crawl was completely automatic.

3.2 Dark Web Forums Used

The number of active users in the dark web forums found between October and December 2019 within the context of this research ranged either between a few hundred or between a thousand and more. Since the probability of finding users who are active in two or more forums is expected to be higher when concentrating on those forums that seem to be the most popular, only forums with more than 1000 active users were selected. However, in future work, this threshold could be lowered to also include smaller forums with only a few hundred users to increase the size of the data set [1].

At the end of 2019 there were fewer than 10 Dark Web forums found with a large community (around 1000 active authors or more). Unfortunately, the number of those forums that allow users to publish their PGP keys in their user profiles, was even smaller. In the end, only four forums fulfilled the requirements for this analysis, which are presented in Table 1.

3.2.1 DNM Avengers. This is the smallest Dark Web Forum that was crawled within this research. It focuses mainly on drugs, but there are also some threads about more general topics, politics, security, cryptocurrency or Dark Web marketplaces.

3.2.2 The Majestic Garden (TMG). This onion service is a mixture of forum and marketplace. It has some threads dealing with general topics but the main focus is on drugs as well. TMG has over 40,000 users in total (end of 2019) but only around 5,000 that are active (wrote at least one post) in the forum. Compared to the three other forums, it is the only one where users are not able to access the user profiles of others. Therefore, users post their public PGP keys within multiple threads to share them with others. Unfortunately, it's very time-consuming to check all posted PGP keys manually, whether they are complete or unusable or whether they have been posted twice or even more times. Hence, due to simplicity, only those users that have a PGP key and are linkable to one of the three other forums are manually checked and counted as users that own a key in Table 1. Hence, 193+ indicates that there are more users with a PGP key within this forum, but their exact number was not calculated.

[1]Due to privacy concerns, the data set created for this project is available by request only on GitHub [7].

3.2.3 The Hub (TH). TH is the sister-forum of The Majestic Garden, containing approximately 225,100 posts and 26,500 active users. As its name already indicates, TH is a kind of central point with many threads with discussions about other Dark Web sites, mainly marketplaces. Additionally, it contains many threads regarding security and vendor reviews.

3.2.4 Dread. Dread is the biggest forum crawled within this research with over 63,000 active users. In contrast to the others, it does not have a specific topic. It is more like a platform for everyone that wants to ask questions or talk about various topics. Besides that, the design of this website strongly resembles Reddit, which many people already know from the Surface web. Dread recently faced significant DDoS attacks and thus has extremely strict DDoS protection, which makes it difficult to crawl.

3.3 Features

Features that are extracted from the given data are the basis for an AA analysis. The feature categories used in this research, are influenced by related work or are established based on scientific interest (language model). Another category called social-based features (e.g., the usage of quotations of other user comments) was used in previous work but did not contribute well to the final results. Therefore, this feature category was excluded from this research. However, there might be other features (e.g., transforming text to an image) that are worthy of interest but that are not considered here. These could be analyzed in future work.

The features used in this project are listed in Table 2. More detailed information can also be found in [20]. The four feature-categories used in this research, as well as their corresponding subfeatures, are explained in the following.

3.3.1 Lexical-based Features. Term frequency, also known as Bag of Words (BOW), or Term Frequency Inverse Document Frequency (TF-IDF) features are called lexical-based features in this research. They are often used in the context of AA [18], [11]. The main focus of these features is not on the topic but rather on the frequency of words within a text. However, the TF-IDF approach tries to overcome a typical problem of the BOW approach that rarely used words (that might be the most interesting ones) are shadowed by more frequently used words.

3.3.2 Stylometric-based Features. Stylometric features are frequently used for authorship attribution tasks [21],[2], [9]. The focus of this category is not on *what* an author is writing about, but rather *how* he writes a text. This includes grammar mistakes, typos, emojis, part-of-speech (POS) tags, as well as statistical measurements of an author's writing style, e.g., the number of sentences, words, characters per word, etc.

3.3.3 Time-based Features. This feature category is inspired by La Morgia et al. [15], and Spitters et al. [21]. It contains six subfeatures: the time (hour and minute) when a user is typically active within a Dark Web forum, the date (year, month, and day) that a post was posted, and the day of the week on which a post was written, which might be very important to see whether some users tend to be more active during weekends and others more during weekdays.

Table 2: Features used within this research. For those features that are annotated with a * the sum, mean, median, and standard deviation are computed with regard to every post.

Category	*Feature*	*Extraction Tool*
Lexical	Count Vectoriser	scikit-learn [16]
Lexical	TF-IDF	scikit-learn [16]
Language Model	Word2Vector, GloVe, FastText	Gensim [19]
Language Model	Sentiment (pos./neg./neu./comp)	vaderSentiment [10]
Language Model	LDA and NMF	scikit-learn [16]
Stylometric	11 Emoji categories	RE (Python)
Stylometric	Grammar mistakes	LanguageTool API [8]
Stylometric	Typos	LanguageTool API [8]
Stylometric	35 POS tags	nltk [3]
Stylometric	Number of characters per word*	Python
Stylometric	Number of capital letters*	RE (Python)
Stylometric	Number of small letters*	RE (Python)
Stylometric	Number of punctuation marks*	RE (Python)
Stylometric	Number of abbreviations*	nltk [3]
Stylometric	Number of lowercase-words*	RE (Python)
Stylometric	Number of uppercase-words*	RE (Python)
Stylometric	Number of words with both cases*	RE (Python)
Stylometric	Number of numbers	Python
Stylometric	Number of words	Python
Stylometric	Number of spaces	Python
Stylometric	Number of sentences starting with a capital letter*	RE (Python)
Stylometric	Lexical richness	nltk [3]
Stylometric	Number of Sentences	nltk [3]
Stylometric	Number of Lines	nltk [3]
Stylometric	Number of invisible Characters*	RE (Python)
Time	Minute, hour, day, month, year, day of the week	Python

3.3.4 Language model-based Features. This feature category contains features based on a sentiment analysis, on two topic-modeling algorithms, Latent Dirichlet Allocation (LDA) [4] and Non-Negative Matrix Factorization (NMF), as well as on three language-modeling algorithms, Word2Vec [13], GloVe [17], and FastText [5]. These algorithms are not typically used within AA analyses. However, they were chosen to be part of this research to experiment with different ways to analyze the topic, word embeddings, semantic, and sentiment an author typically uses within a post.

4 METHODOLOGY

AA in the Dark Web is technically hardly different from AA in the Surface Web, but it is very different in terms of the conceptual view and the underlying conditions. In the Surface Web, there are large data sets (e.g., datasets based on Twitter) that can be used for AA. In many cases, they contain the full name of the authors which can be used as ground truth for a supervised ML approach. The Dark Web, on the other hand, contains relatively few data sources with posts from users who want to hide their identity (ground truth). The latter makes it nearly impossible to combine training data from the Surface Web and test data from the Dark Web for a supervised ML approach and therefore to overcome the problem of a limited amount of data in the Dark Web.

The authorship attribution (AA) analysis within this research is based on ML tools provided by scikit-learn version 0.22 [16] and is very extensive, which is why the following sections are very important to understand and interpret the results described in Section 5. Due to the immense computational costs of the analyses, hardware provided by the Platform of Scientific Computing at Bonn-Rhein-Sieg University of Applied Sciences was used for this research [24].

Table 3: Remaining forum combinations and number of corresponding authors after filtering out all authors with less than 50 posts in both forums.

Name	*Forum combination*	*Number of authors*
FC-1	DNMA & TH	2
FC-2	DNMA & TMG	2
FC-3	DNMA & Dread	7
FC-4	TMG & Dread	10
FC-5	TH & Dread	17
FC-6	TMG & TH	20

Table 4: The number of words per author. All values are averaged over all authors within the respective forum combinations.

FC	*Median*	*Mean*	*Standard Deviation*	*Min*	*Max*	*Sum*
FC-1	484	626	734	2	5697	223659
FC-2	121	238	467	4	4732	108605
FC-3	158	366	482	8	3963	111012
FC-4	168	243	390	4	2775	74720
FC-5	132	307	530	2	4016	111028
FC-6	145	268	560	2	4992	131828

4.1 Preparations

Before the analysis could start it was crucial to select suitable authors from the dataset, as well as to decide which preprocessing tools and which classification algorithms should be used. Detailed information about all preprocessing steps and classifiers used within this research can be found in [20].

4.1.1 Selection of Authors. One of the first steps to be able to start the analysis was to find appropriate authors. In this case, the term appropriate refers to authors that can be linked via a PGP-key. On top of that, a trade-off needed to be found between the number of posts that have been written by an author within a forum (the more the better) and the total number of authors that remains for the analysis (the more the better). Therefore in this research, only authors that have written at least 50 posts in two forums were considered to be appropriate. The remaining forum combinations with more than one candidate author, as well as the number of remaining authors, are listed in Table 3. In addition, some text statistics are available in Table 4.

4.1.2 Preprocessing Tools. Before data is fit into a classifier, it is often beneficial to preprocess it first to either fulfill the requirements of a classifier or just to improve the final results. In this research, three different kinds of preprocessing are tested: standardization, normalization or no preprocessing at all. Standardization is used because some estimators are sensitive to the distribution of the data they are fitted with. Normalizing is especially useful when the similarity between a pair of samples should be computed and is

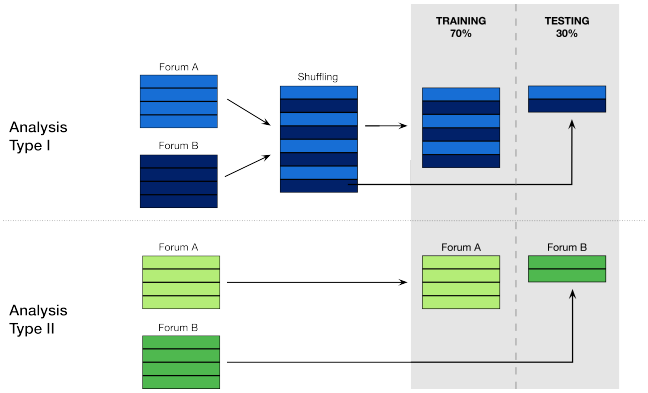


Figure 1: Visualisation of analysis type I and analysis type II

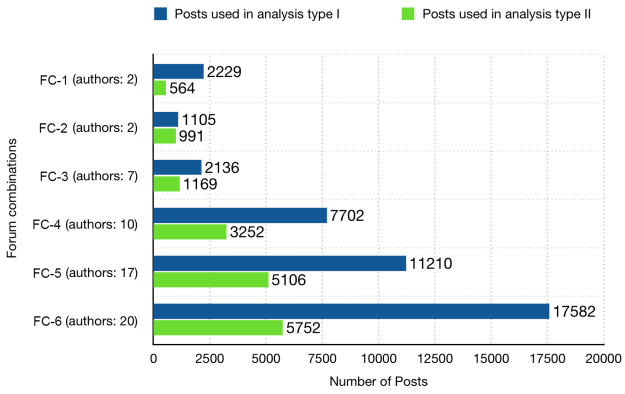


Figure 2: Each bar shows the sum of all posts from all authors within the corresponding forum combination and type of analysis.

often used for text classification, e.g., in the TfidfTransformer from scikit-learn [16].

4.1.3 Classifiers. There are several different classifiers available in scikit-learn. Some of them can be linked to the three classifier categories SVM, Naive Bayes, and Decision Trees, which are in the main focus within this research. However, other classifiers like K-nearest neighbors (KNN) or a Multilayer Perceptron (MLP) are used in this analysis.

4.2 Final Setup of the Analysis

The analysis of this research is extensive because of the intention to analyze the data in as many ways as possible to find the most appropriate one. First of all, each of the six forum combinations listed in Table 3 is analyzed in two different ways (see Section 4.2.1 and 4.2.2). In both types, each forum combination is once analyzed with an unbalanced dataset (with the full amount of data) and once with a balanced dataset (see Section 4.2.3). Furthermore, within each analysis type, each forum combination is analyzed with three versions of the dataset. Each of these three versions contains only those authors that wrote a specific minimum of posts (see Section 4.2.3).

4.2.1 Analysis Type I: Combined Analysis. The most common technique for AA is to extract a text corpus that includes all texts from all authors and split this corpus into a training set and testing set. This is done in the first part (type I) of the analysis by combining each forum pairing as listed in Table 3 into a single dataset. After that, the posts are mixed and split up into 70% training data and 30% testing data. The advantage of this type of analysis is that all posts from all linked authors can be used. However, a major disadvantage is that the proportion of posts from forum A and B vary significantly from author to author. Therefore, the main focus of this analysis is on the feasibility of AA, based on posts from different Dark Web sources. The results of this analysis type will always be visualized in blueish colors in all the figures in this paper.

4.2.2 Analysis Type II: Separate Analysis. The second part of the analysis is based on training sets that contain only posts from forum A of a given forum combination and testing sets that contain only posts from forum B of the same forum combination. The differences between analysis type I and II are visualized in Figure 1. Within analysis type II, the forum that has more posts per author on average is used for the training set and the other for the testing set respectively. Furthermore, the proportion between the two sets remains the same as in type I (70%/30%), which unfortunately leads to a high loss of data. The extent of this problem is illustrated in Figure 2. However, when the number of posts of an author has to be reduced to maintain the ratio, then only the longest posts were chosen for the corresponding data set.

This type of analysis can reveal which features can achieve good results even when the author might have changed some of their typical behaviors between the training and testing forum. Therefore the focus of this analysis is on the suitability of the extracted features. The results of this analysis type will always be visualized in greenish colors in all figures in this paper.

4.2.3 Sub-analyses. Both analyses (type I and II) are further divided into several sub-analyses. In general, the more text from an author that exists, the better is the probability of a successful AA analysis. Thus, there are three different sub-analyses for each analysis type. In the case of analysis type I, this is an analysis with all authors, one with only those authors that wrote more than 500 posts, and the last with only those authors with more than 1000 posts. As the total number of posts per author is lower in analysis type II, one sub-analysis is based on all authors, the second on all authors that wrote more than 200 posts (summed over both forums), and the last on authors that wrote more than 400 posts.

A major problem of both analysis types, as well as the previously mentioned sub-analyses, is that some authors wrote a huge number of posts whereas others just wrote only a few hundred or fewer. Figure 3 visualizes the situation in analysis type I only, but it is similar to that in analysis type II. To analyze the effect of this problem on an AA analysis, all datasets are analyzed twice; one time unbalanced and the other time balanced. Balanced means that the number of posts of all authors is limited to the number of posts written by the author with the fewest posts within the dataset.

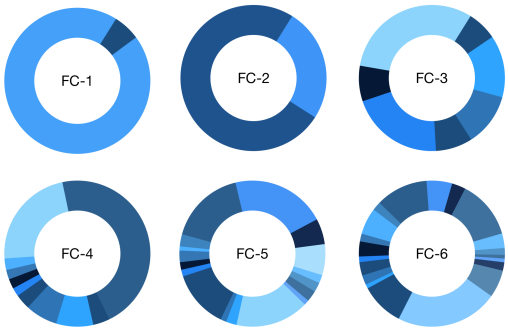


Figure 3: The proportion of the number of posts for each author within the six datasets of analysis type I, where each color represents a different author.

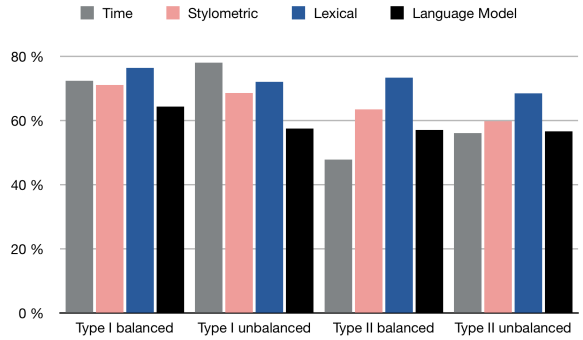


Figure 4: The average F1 scores achieved by the four feature categories calculated over the results obtained from the analyses of all six forum combinations.

Similar to the procedure in analysis type II, only the largest posts are kept when shrinking the number of posts of an author. In the case of analysis type I, also the proportion of posts from forum A and B was balanced as much as possible to focus on the main research question, which was AA of posts written in two different Dark Web forums.

5 RESULTS

The results described in this section are selected results for each forum combination and each feature category within each analysis mentioned in Section 4. In this research, the best results of an analysis and/or classifier are considered to be those with the highest F1-score and the highest accuracy. This is based on the fact that a high recall, as well as a high precision, are essential for AA in the Dark Web to assign a post to its correct author as reliably as possible. Since the F1 score is a kind of average between precision and recall, only this score is chosen as an evaluation criterion.

When focusing on Figures 5 and 6 it can be observed, that AA within this analysis becomes more difficult with more candidate authors. However, there is a significant difference between the results of analysis type I and analysis type II, which is most significant for datasets with more than two authors (FC-3 to FC-6). The results of analysis type II (Figure 6) are, in general, 10% to 20% worse than those achieved within the analysis type I (Figure 5). This tendency can also be found in the remaining analyses with a focus on authors that have written comparatively many posts.

The results achieved by the different feature-categories differ significantly among each other and between the different types of analyses. Therefore, in the following subsections strengths and weaknesses of the different categories that can be concluded from the given results of FC-5 and FC-6 are described. These two forum combinations are chosen for this more detailed analysis because they represent the challenges and difficulties for a successful AA analysis. However, a brief overview of the *average* results (F1-score) of the feature categories within the analyses of *all* forum combinations is shown in Figure 4.

5.1 Time-based Features

Time-based features belong to the most accurate features of all within both analysis types of FC-1 and FC-2 but unfortunately not for the other four forum combinations with a higher number of authors (see Figures 5 and 6). However, a detailed analysis of the results revealed that the correct author of a post is determined 88% (FC-6) and 91% (FC-5) within the three most likely candidates when considering time-based features only and all authors within type I. Except that, in 40 out of all 52 analyses, the time-based features were classified best by tree-based classifiers like the ExtraTrees classifier and RandomForest classifier.

Within FC-5, one main reason for good performance of the time-based features is the number of posts per author. However, the results of FC-6 show that it does not necessarily mean that an author is unidentifiable when they have written only a few posts. This indicates two facts: in general, the more candidate authors there are, the higher the number of posts per author is needed to identify an author using time-based features. On the other side, some authors do not need a high number of posts because their daily rhythm seems to be so atypical that they stand out easily. When taking a look at the balanced analysis that considers the longest posts of all authors, the F1-score of those authors that wrote many posts drops significantly. This is because the number of posts is reduced to a minimum number of posts written by an author within the dataset. On top of that, when balancing the dataset by choosing posts at random, the same tendency occurs. This leads to the conclusion that there is, in general, no connection between the length of a post and the time when a post is written. As visible in Figure 6 the results achieved by analysis type II are significantly lower than within analysis type I. The most obvious reason for the poor results within type II would be that there are simply not enough posts for each author to be able to find all daily rhythms. As the results within analysis type I are comparatively high, it seems that authors tend (at least) to be active to a similar time within their favorite forum in that they have written the most posts.

5.2 Stylometric-based Features

Compared to the time-based features, the results of the stylometric-based features are worse within analysis type I whereas they are,

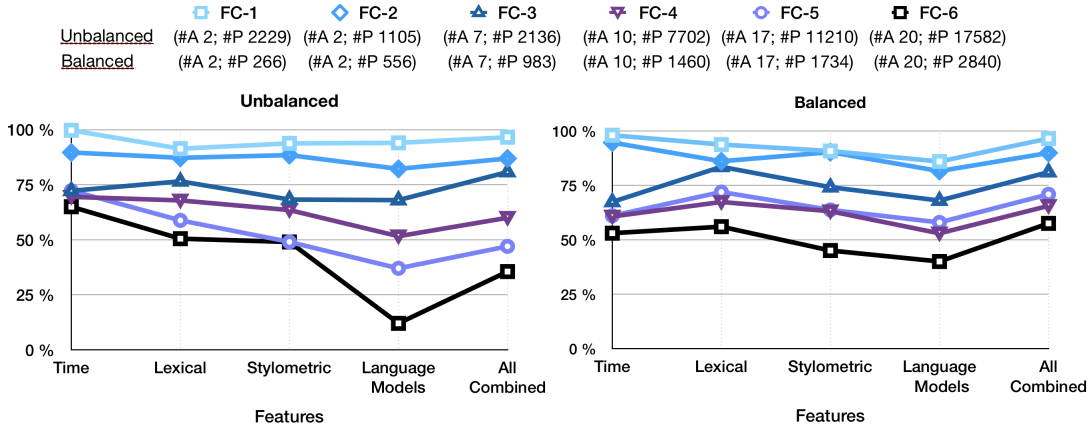


Figure 5: Overview of the F1-Score achieved by analyzing the complete datasets of all forum combinations for each feature category with analysis type I by using either an unbalanced dataset or a balanced dataset. #A denotes the number of authors within a forum combination and #P the number of posts.

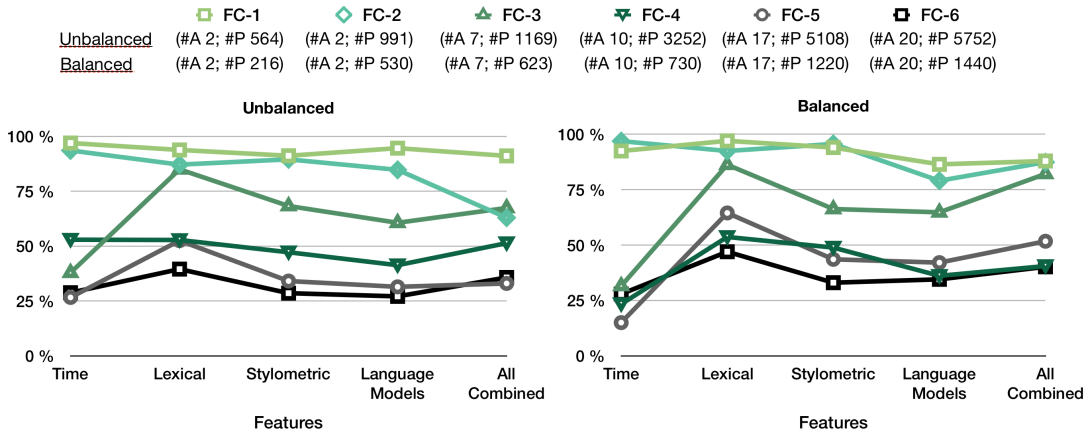


Figure 6: Overview of the F1-Score achieved by analyzing the complete datasets of all forum combinations for each feature category with analysis type II by using either an unbalanced dataset or a balanced dataset.

in general, better within analysis type II (see Figures 5 and 6). In addition to the results shown in these two charts, the probability that the correct author can be found within the top 3 most likely candidates within FC-5 and FC-6 is around 50% for type II and ranges between 70% and 83% for type I. However, the classifiers that achieved the highest results by analyzing this feature category, were the LinearSVC (26/52), the ExtraTreesClassifier (11/52), and the MLP Classifier (9/52).

One of the most important factors for a high F1-score when analyzing stylometric-based features seems to be the number of posts. This is based on the observation that the F1-score of authors that could be identified best is more negatively affected when the dataset is balanced (no matter if only the longest posts are chosen or not). Therefore, the stylistic pattern that makes these authors expose most, can only be found when analyzing a great number of posts and thus a mixture of long and short posts. The remaining analyses revealed, that the more posts per author on average are included in the dataset, the more effective it is to focus on the writing style in longer posts. Furthermore, the results achieved by poorly detectable authors show that balancing the dataset and focusing on only the longest posts leads to an improvement of their score up to 44%. Therefore, a tradeoff needs to be found that keeps as many posts as possible for each author from the well-identifiable authors within the dataset, but at the same time, reduce the number of posts as much as possible so that the others also have a chance to stand out.

However, results of analysis type II are around 20% lower than those within analysis type I. The main reason for that seems to be the comparatively low number of posts within the dataset as well as a tendency, to write posts of a different length within the two forums. Therefore, the overall conclusion when considering the results of analysis type I and II is that authors seem to write more passionately in either one or the other forum, which results either in a different style of their posts or in a different number

of posts per forum that are available for the analysis. Both cases are a problem for the stylometric-based features especially within analysis type II.

5.3 Lexical-based Features

The results shown in Figures 5 and 6 prove that the lexical-based features belong to the best features within this analysis. When an unbalanced dataset is used, then the probability that the correct author of a post is within the top three candidates is at least very close to or above 60% in nearly all cases (analysis type I and type II). On top of that, when the analysis is based on a balanced dataset, the probability of finding the correct author within the top three rises to at least 66% and up to 86%. When taking a closer look at the best classifiers, then it becomes apparent that Naive Bayes classifiers seem by far the most suitable type when analyzing the lexical-based features (in 37 of 52 analyses).

When focusing on the results of each author within the balanced and unbalanced analyses (those that consider all authors) it becomes obvious that an author that wrote long posts stands out from the crowd more easily when considering only a few posts than an author that usually writes many short posts. This means that the length of the posts is the most important influencing factor for the lexical-based features. Furthermore, when comparing the results achieved by an analysis of all authors to those achieved by an analysis that considers only authors with 500 or 1000 posts, an interesting fact can be observed. Focusing on a few longer posts per author by balancing the dataset is, in general, more or at least equally effective than focusing on only those authors that wrote comparatively many posts.

In general, the lexical-based features are suited comparatively well for authorship attribution within analysis type II. As the overall score of the lexical-based features within analysis type II is better than that of the stylometric-based features, it seems like authors rather tend to write about the same things or at least use similar words within two different forums than to use the same writing style when posting a post.

5.4 Language Model-based Features

The language model-based features, combined together into a single dataset, are not suitable for AA within the Dark Web (see Figures 5 and 6). Datasets with a large number of authors (FC-6) seem to be especially problematic; in the best case the correct author of only 62% of all posts is listed within the top 3 most likely candidates. FC-5 has three fewer authors, which seems to lead to a slightly better probability of finding the correct author within the top 3 candidates (up to 75%). When examining at the classifiers, the LinearSVC (25/52) and the PassiveAgressiveClassifier(13/52) seem to achieve the best results of all classifiers when analyzing the language model-based features only.

There are a few authors that can be recognized better when focusing on language model-based features only but there is no single common reason for all authors. Some have many posts and others do not, the same applies to the length of the posts, some have long posts, others not. However, two small tendencies can be observed. First, it is more likely that an author that has written 500 or more posts can be identified comparatively well. Second, when an author tends to write small posts, then it is more likely that a classifier cannot classify them by using only this feature category.

5.5 Voting

When putting all features together and analyzing them combined, one might expect that the results must increase because all information is now merged. However, real-life experience shows, that this is not the case within most of the analyses in this research. The probability that a post can be correctly attributed to its author when focusing on all features combined within the unbalanced datasets of FC-5 and FC-6 including all authors is comparatively low (40% or less). However, there is a clear tendency that this probability increases when balancing the dataset. In the best case, it rises within analysis type I to 80% and even higher (to 86%) when the top 3 most highly ranked candidates are included.

The question is, where did the potential of the individual features get lost in the joint analysis? The answer is surprisingly simple: each feature category can be classified best by different classifiers. Thus, when putting all features together and classifying them with only one classifier, the results decrease. Therefore, another approach to analyzing all features combined was tested. In contrast to the previous one, the features are not put together in one single dataset. Instead, they are classified as stand-alone by the same classifier with the same classifier parameters. However, this time, the result of each classifier is fed into a final voting classifier. Thus, this final classifier receives four results from four different classifiers for each sample that is used for testing. Out of these results, the voting classifier computes the most likely candidate author for each post. Since it is known from the previous analyses when each feature category works well, weights can be added to the computation so that those classifiers that are more reliable in a given situation than others, have more influence on the final result. E.g., when focusing on analysis type I with an unbalanced dataset, it is known that the time-based features work very well, the stylometric-based and lexical-based features are not as suitable as the time-based features, but still work well, whereas the language model-based features achieved the worst results. This knowledge produces a tendency for which features should be weighted more or less. However, the final weights still need to be determined by experimentation.

Mathematically, the computations made by the voting classifier can be described as in Equations (1) and (2) where $\mathbf{B} \in \mathbb{R}^{n \times j \times k}$ denotes a three-dimensional matrix that contains the probability estimates of all classifiers (the probability of the posts for each author in each model) and $\mathbf{A} \in \mathbb{R}^{j \times k}$ denotes a matrix that contains the weighted averages $a_{m,l}$ of the probability estimates.

$$a_{m,l} = \frac{\sum_{i=0}^{n} b_{i,m,l} \cdot w_i}{\sum_{i=0}^{n} w_i} \; \forall \; a_{m,l} \in [0,1] \tag{1}$$

for all $m \in \{0, ..., j\}$, where j denotes the total number of samples, for $l \in \{0, ..., k\}$, where k denotes the total number of authors, for $b_{i,m,l} \in \mathbf{B}_{n \times j \times k}$, where n denotes the total number of classifiers, and where the vector w contains the weights for each classifier. The final output of the voting classifier can be mathematically described as shown in (2):

$$v_m = \arg \max_{l \in \{0,\ldots,k\}} a_{m,l} \tag{2}$$

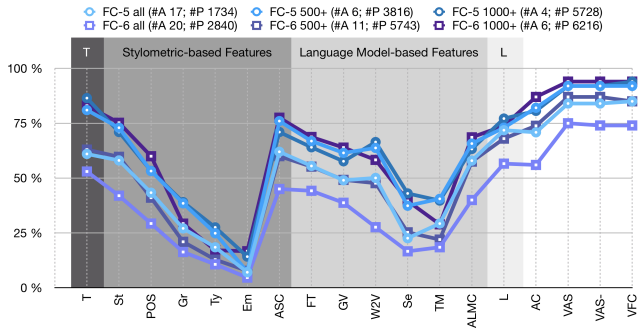


Figure 7: Detailed overview of all results achieved by analysis type I of FC-5 and FC-6 (balanced). #A denotes the total number of authors and #P denotes the total number of posts. Acronyms are explained in Table 5.

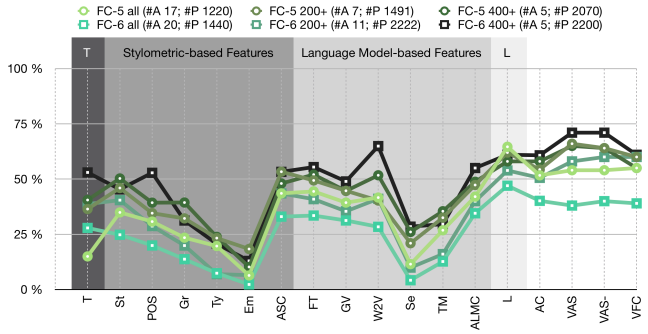


Figure 8: Detailed overview of all results achieved by analysis type II of FC-5 and FC-6 (balanced). #A denotes the total number of authors and #P denotes the total number of posts. Acronyms are explained in Table 5.

Table 5: Acronyms used within Fig. 7 and Fig. 8.

ASC: All Stylometric-based Features Combined	**Em:** Emojis
ALMC: All Language model Features Combined	**FT:** FastText
AC: All Features Combined	**GV:** GloVe
L: Lexical-based Features	**Gr:** Grammar
T: Time-based Features	**POS:** POS-tags
TM: Topic Modeling	**Se:** Sentiment
VAS: Voting all Features Separately	**Ty:** Typos
VAS-: Voting all Features Separately - Emojis etc.	**St:** Style
VFC: Voting Combined Feature Categories	**W2V:** Word2Vec

where *argmax* computes the column index of the author/column with the highest probability and thus the most likely author within each row (sample) of **A**.

Figure 9 shows a significant increase of the results achieved by voting all feature categories instead of putting them all together into one single dataset and analyzing them by a single classifier (Figures 5 and 6). Especially within analysis type I, the results of the previously unsuitable unbalanced datasets increases to 80% in the case of FC-6 which is the largest forum combination with 20 authors. A similar trend can be seen for FC-5, which achieves an F1-Score of 87% when voting the results of all feature categories. Interestingly, the results of analysis type II, as well as those of the balanced analyses, do not increase that much. There might be several reasons for that. All datasets of analysis type II as well as the balanced datasets of analysis type I contain significantly fewer posts. This leads to an increase of the F1-score of the lexical-based features but to a decrease of the score for the time-based features. As the latter ones had the most influence on the voting results of the unbalanced datasets, it is not surprising that the results of the balanced datasets do not increase in the same way. In the case of analysis type II, there were only a few feature categories that passed the 50% limit at all. Thus, the voting classifier is not able to improve the results of this final analysis when most of the feature categories are not able to pass the 50% limit stand alone.

In Figures 7 and 8 a more detailed overview of the results achieved by all features and different types of voting analyses is shown. Voting all Features Separately (**VAS**) denotes a voting analysis of all subfeatures without the results of the corresponding feature category where all subfeatures are combined and analyzed with a single classifier. In Voting all Features Separately without Emojis etc. (**VAS-**) only those subfeatures with the best results are voted. Therefore this analysis does not include the results of the analyses of the emoji, sentiment, topic modeling, grammar, and typo features. As the weights for these features were already quite small within VAS it is not surprising that the voting results of an analysis without them, do not change significantly. The results that are labelled Voting Combined Feature Categories (VFC) are those that were discussed in the previous paragraph. Thus in this case, only four results (one for each feature category) are voted. This version of voting seems to work best for large datasets. However, when the datasets are small, like in the balanced analyses, there is either no difference with the other two voting methods or the results are a little worse. Besides that, Figure 7 shows that in most cases subfeatures that are analyzed separately do not achieve better results than when analyzed combined.

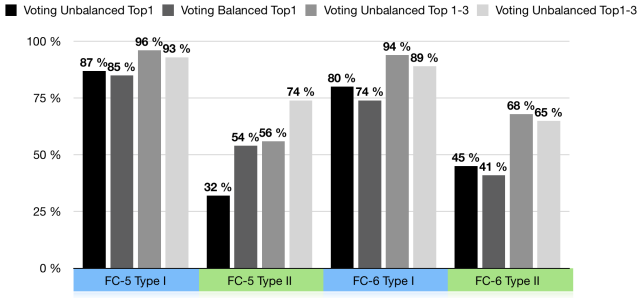


Figure 9: Comparison of the probability that a post is classified to its correct author (top 1) or that the correct author is within the three most likely candidates (top 1-3) when considering all authors and vote the classification results of all features.

6 CONCLUSIONS

The focus of this research was on authorship attribution within multiple Dark Web forums. That AA is feasible within one forum was already shown by e.g., M. Spitters et al. [21]. Thus the main research question was whether it would also be possible beyond the boundaries of a single forum. Therefore, a crawler/scraper was developed that can extract posts from four different Dark Web forums: DNM Avengers, The Hub, The Majestic Garden, and Dread. By comparing public PGP-keys, users were linked between these four forums. The results show that it is (in general) a good idea to reduce the number of posts from authors that wrote significantly more posts than the average. By this, it is more likely to better classify authors with only a few posts. However, even under the challenging conditions with posts originating from different sources, there is still a probability of at least 94% that the correct author of a post can be found within the three most likely authors. This result is achieved by voting the results of four different classifiers that classify four different feature categories. However, it shows that AA is indeed a danger to the anonymity of Dark Web users across the boundaries of different forums. Therefore, all users that want to avoid getting linked via AA should keep the following aspects in mind: a post can most likely be assigned to its author if they tend to have an abnormal or very typical daily rhythm that is reflected in their online behavior, if they tend to write many long or short posts with the same structure, and also even when they write only a few but comparatively long posts with a large number of similar words.

6.1 Future Work

Several aspects could not be realized within this research and should be optimized or extended in future work. Unfortunately, only a few authors could be linked at all between these forums and those that could be linked rarely wrote more than 50 posts in both forums. Thus, it would be desirable to find more popular Dark Web forums that can be used as an additional data source for further validating the research presented. It would also be interesting to analyze whether the improvment in the results within the analyses with a focus on authors with a greater number of posts is rather related to the comparatively high number of posts or to the reduced number of authors. Apart from this aspect, the analysis of ensembling methods like stacking, boosting, or bagging would also bear fruitful results.

ACKNOWLEDGMENTS

The authors would like to acknowledge the financial support of both the Natural Sciences and Engineering Research Council (NSERC) as well as the New Brunswick Innovation Foundation (NBIF) for their support of the research. Thanks also to Stephen MacKay for editing grammar and style and Andreas Priesnitz for the great support on the University of Applied Sciences Bonn-Rhein-Sieg (BRSU) side to help this research move forward. The authors would also like to thank the Plattform of Scientific Computing at BRSU for providing the hardware needed for the analyses.

REFERENCES

[1] Johanna Amann and Robin Sommer. 2016. Exploring Tor's Activity Through Long-Term Passive TLS Traffic Measurement. In *Passive and Active Measurement*, Thomas Karagiannis and Xenofontas Dimitropoulos (Eds.). Springer International Publishing, Cham, 3–15.

[2] M. Ashcroft, F. Johansson, L. Kaati, and A. Shrestha. 2016. Multi-domain Alias Matching Using Machine Learning. In *2016 Third European Network Intelligence Conference (ENIC)*. 77–84.

[3] Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python* (1st ed.). O'Reilly Media, Inc.

[4] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet Allocation. *J. Mach. Learn. Res.* 3, null (March 2003), 993–1022.

[5] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching Word Vectors with Subword Information. *CoRR* abs/1607.04606 (2016). http://arxiv.org/abs/1607.04606

[6] Gwern Branwen, Nicolas Christin, David Décary-Hétu, Rasmus Munksgaard Andersen, StExo, El Presidente, Anonymous, Daryl Lau, Sohhlz, Delyan Kratunov, Vince Cakic, Van Buskirk, Whom, Michael McKenna, and Sigi Goode. 2015. Dark Net Market archives, 2011-2015. www.gwern.net/DNM-archives. Accessed: 22-05-2019.

[7] CAS-Atlantic. [n. d.]. Dark Web forum dataset 2019 (DWF-CAS-IVC-2019). https://github.com/CAS-Atlantic/Dark-Web-forum-dataset-2019-DWF-CAS-IVC-2019. Accessed: 21-08-2020.

[8] LanguageTooler GmbH. [n. d.]. LanguageTool - Proofreading Software. https://languagetool.org. Accessed: 09-08-2020.

[9] Thanh Nghia Ho and Wee Keong Ng. 2016. Application of Stylometry to DarkWeb Forum User Identification. In *Information and Communications Security*, Kwok-Yan Lam, Chi-Hung Chi, and Sihan Qing (Eds.). Springer International Publishing, Cham, 173–183.

[10] C.J. Hutto and E.E. Gilbert. 2014. VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. In *Eighth International Conference on Weblogs and Social Media (ICWSM-14)*. Ann Arbor, MI.

[11] R. Layton, P. Watters, and R. Dazeley. 2010. Authorship Attribution for Twitter in 140 Characters or Less. In *2010 Second Cybercrime and Trustworthy Computing Workshop*. 1–8.

[12] G. Me, L. Pesticcio, and P. Spagnoletti. 2017. Discovering Hidden Relations Between Tor Marketplaces Users. In *2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*. 494–501.

[13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. 1–12. https://arxiv.org/abs/1301.3781

[14] Jelena Mirkovic and Peter Reiher. 2004. A Taxonomy of DDoS Attack and DDoS Defense Mechanisms. *SIGCOMM Comput. Commun. Rev.* 34, 2 (apr 2004), 39–53.

[15] M. La Morgia, A. Mei, S. Raponi, and J. Stefa. 2018. Time-Zone Geolocation of Crowds in the Dark Web. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. 445–455.

[16] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-Learn: Machine Learning in Python. *J. Mach. Learn. Res.* 12, null (Nov. 2011), 2825–2830.

[17] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. 1532–1543. http://www.aclweb.org/anthology/D14-1162

[18] S. R. Pillay and T. Solorio. 2010. Authorship attribution of web forum posts. In *2010 eCrime Researchers Summit*. 1–7.

[19] Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. ELRA, Valletta, Malta, 45–50.

[20] Britta Sennewald. 2020. *Authorship Attribution in the Dark Web*. Master's thesis. University of New Brunswick, Fredericton, NB, Canada.

[21] M. Spitters, F. Klaver, G. Koot, and M. v. Staalduinen. 2015. Authorship Analysis on Dark Marketplace Forums. In *2015 European Intelligence and Security Informatics Conference*. 1–8.

[22] M. Sultana, P. Polash, and M. Gavrilova. 2017. Authorship recognition of tweets: A comparison between social behavior and linguistic profiles. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 471–476.

[23] S. Swain, G. Mishra, and C. Sindhu. 2017. Recent approaches on authorship attribution techniques — An overview. In *2017 International conference of Electronics, Communication and Aerospace Technology (ICECA)*, Vol. 1. 557–566.

[24] Bonn-Rhein-Sieg University. [n. d.]. Platform for Scientific Computing at Bonn-Rhein-Sieg University. https://wr0.wr.inf.h-brs.de. Accessed: 08-05-2020.

[25] Min Yang and Kam-Pui Chow. 2014. Authorship Attribution for Forensic Investigation with Thousands of Authors. In *ICT Systems Security and Privacy Protection*, Nora Cuppens-Boulahia, Frédéric Cuppens, Sushil Jajodia, Anas Abou El Kalam, and Thierry Sans (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 339–350.

Dynamic Reconfiguration of Consensus Protocol for IoT Data Registry on Blockchain

Mohammadreza Rasolroveicy
mohammadreza.rasolroveicy@polymtl.ca
Department of Computer and Software Engineering
Polytechnique Montreal
Montreal, Canada

Marios Fokaefs
marios.fokaefs@polymtl.ca
Department of Computer and Software Engineering
Polytechnique Montreal
Montreal, Canada

ABSTRACT

Blockchain technology has recently gained momentum among practitioners to increase security in shared distributed platforms. One of the main characteristics of Blockchain is consensus that prevents double-spending attacks on the network and lowers the chance of Distributed Denial-of-service (DDoS) attacks. However, each consensus algorithm has its strengths and its limitations. The purpose of this paper is to design a self-adaptive mechanism that can dynamically choose the appropriate consensus algorithm for the network. This framework provides an ability to easily manage and change the consensus algorithm for IoT data registries based on Blockchain in an effort to manage cost, performance, and security of the network. Moreover, it can dynamically reconfigure properties of the consensus protocol including number of validation nodes, validation time and others. The goal is to provide an effective and cost-efficient consensus protocol while ensuring quality of service. Our experiments show that different consensus algorithms behave differently as the workload changes and we demonstrate how our MAPE-k architecture can allow additional flexibility under dynamic conditions.

KEYWORDS

Internet of Things, Software Performance, Adaptive Systems, Blockchain

ACM Reference Format:
Mohammadreza Rasolroveicy and Marios Fokaefs. 2020. Dynamic Reconfiguration of Consensus Protocol for IoT Data Registry on Blockchain. IBM Corp., Riverton, NJ, USA, 10 pages.

1 INTRODUCTION

Internet of Things (IoT) applications are rapidly gaining popularity and by 2030 the number of IoT devices that will be connected to the internet is projected to surpass 125 billion [27]. IoT has been applied on several domains such as smart homes, smart agriculture, and smart hospitals[11]. Nevertheless, IoT is not without challenges. Due to improper configuration and limited computation resources of IoT devices, they are vulnerable and a main target of hackers. In October 2016 [9], an attack, known as the Mirai botnet attack, exploited the poor security configuration of thousands of IoT devices, connected to the internet using IP addresses, such as security cameras and wireless printers. The attackers created several DDoS (Distributed Denial of Service) attacks and targeted the DNS (Domain Name System) provider. Subsequently tens of thousands of IoT devices were used to produce several DNS lookup requests which resulted in downtime of the internet for several users in the USA, Canada and Europe [5].

IoT applications usually include a variety of sensors, as well as controllers and network gateways. These distributed IoT peers constantly store their generated data to the database and they need to access the database for reading and making appropriate decisions. The massive amount of generated data by IoT applications will make it difficult to control and monitor the system properly [17].

Distributed Ledger Technology (or more commonly known as Blockchain) has been proposed as a secure and immutable network for IoT applications which can addresss three main concerns: availability, confidentiality, and integrity [16]. Blockchain principle is built around a consensus protocol [34] that secures the network against double spending attacks [21] and DDoS attacks [29] in IoT applications. This means that when we store IoT data in the network, we can ensure that our data is almost immutable and tractable, so that we can easily trace the anomalies inside the system.

Taking advantage of Blockchain has proven [5] to be a valid solution to mitigate DDOS attacks by introducing a distributed database that is based on peer-to-peer (P2P) networks. In Blockchain systems, each data block that is submitted to the network will be verified by the peers of the network, and then it will be broadcast to all members. Moreover, the leaders in Blockchain only have permission to append the block to the network and these leaders are chosen by a consensus mechanism [14]. Blockchain, unlike the traditional distributed databases, only provides read/write access of the data and when data has been stored, it can neither be altered nor erased. This technique can be effective in preventing attacks, like the Mirai botnet attack, in IoT applications because of non-repudiation; malicious activities are stored, they cannot be erased or tampered with and they can be traced [5].

Despite the ability of Blockchain to guarantee immutability of data through consensus, it is also true that this consensus protocol can be a performance and resource bottleneck, which is contradicted for real-time IoT applications. A number of different consensus protocols have been proposed for Blockchain including Proof of Work (PoW) [15], Proof of Elapsed Time (PoET) [20], Practical Byzantine Fault Tolerance (PBFT) [19], Raft [38] and Devmode [1]. These algorithms have different mechanics and properties, which also include different degrees of performance degradation. One of

CASCON'20, November 10-13 2020, Toronto, Canada

the objectives of this work is to study and quantify this impact on a variety of IoT application scenarios. To achieve this, we selected Hyperledger Sawtooth [36], a Blockchain platform that supports at least four of these consensus algorithms [3], namely Raft, PoET, PBFT and Devmode. To evaluate the performance of each consensus algorithms, we subjected the platform to varying IoT data loads and measured the performance in terms of CPU and response time.

It is expected that the different algorithms will have both advantages and limitations and that these would manifest under different scenarios. Consequently, it is also expected that the stakeholders would want to choose the algorithm that fits best their requirements and business goals. Given that in a real-time IoT application, these requirements may be dynamic, so should the decision about the consensus algorithm. In order to implement such a dynamic behavior, we invest in the design of a self-adaptive management system that would automatically and dynamically switch between consensus algorithms and deployment configurations according to the requirements of the applications and based on changes in the data load. The proposed self-adaptive system is based on the MAPE-k architecture [25]

This work makes two contributions:

- A comparative study on the performance overhead of the available consensus algorithms in Hyperledger Sawtooth including PBFT, PoET and RAFT.
- A self-adaptive system to dynamically and automatically choose and deploy the right consensus algorithm at runtime.

The rest of the paper is organized as follows. In Sections 2 and 3 we outline the related literature and provide the background of the concepts and technologies used in our study. In Section 4, we present the methodology and preparation for our study, while in Section 5, we present our novel adaptive and dynamic consensus protocol. In Section 6 we discuss the results of our experimental study and the evaluation of our self-adaptive mechanism. In Section 7, we discuss the threats to validity of our study and finally, Section 8 concludes our work.

2 RELATED WORK

Our work studies the performance of Blockchain systems, focusing on the consensus protocols, when they act as data registries for IoT applications. In addition, it proposes a self-adaptive system to automatically manage the deployment infrastructure of Blockchain and dynamically switch between consensus algorithms. As such, the present work is related to the integration of Blockchain with IoT and self-adaptive systems.

2.1 Integration of IoT and Blockchain

Dorri et al. [23] proposed a Blockchain-based architecture for resource constrained IoT network. They have removed the Proof of Work (PoW) consensus algorithm to increase efficiency and overheads for real-time IoT applications. They argue that using Blockchain for IoT security and privacy is vital as we can store our produced data in a private immutable ledger that performs similarly to Blockchain and it is also manageable. They have evaluated their proposed method for smart home applications using the Cooja simulator [39]. The results of their study demonstrated that the proposed solution reduces the processing and packet overhead comparably to the Blockchain implementation which was used in traditional Blockchain PoW-based algorithms in Bitcoin.

Aung and Tantidham [12] reviewed different possible approaches that can be applied to an Ethereum private Blockchain for a Smart Home System (SHS). In their work, they have considered SHS as a case study which is an integration of home appliances together with sensors to get automatic operations of heating, lighting, air conditioning, home security, health care systems, and others. The authors presented an approach for a private Blockchain implementation for SHS to deal with privacy and security issues. The Ethereum Blockchain packages for SHS, according to its smart contract features for carrying out access control policy, data storage, and data flow management, have been reviewed. As compared with the work of Dorri et al. [23] the architecture of the two systems is different. The architecture proposed by Aung and Tantidham consists of a smart home Validator nodes (SH-Validators), a private Blockchain and a local storage, which connects to SH sensor and actuator devices. SH-Validators carries out a private Blockchain. The purpose of the private Blockchain is to store policies for data flow or transaction management. However, they argue that because of the low transaction time of Ethereum Blockchain, it may be inappropriate for time-sensitive conditions.

Liang et al. [32] proposed a secure Hyperledger Fabric Blockchain-based dynamic secret sharing mechanism to secure data transmission for Industrial Internet of Things (IIoT). The security authentication in Hyperledger Fabric power, a Blockchain-based network, can guarantee secure communication between the power node and the system, as well as between systems. Their experiments showed that the optimized Fabric power data storage and transmission show high security and reliability. The proposed technique can improve the transmission rate and packet receiving rate by 12% and 13%, respectively. According to the authors, it is important to evaluate how efficient their model is in terms of resource and energy consumption.

Alaslani et al. [6] have studied the effect of emerging IoT applications on the end-to-end delay encountered by Byzantine-based Blockchain systems. The effects of a broad range of IoT traffic characteristics including packet arrival rate, payload size, device density, and surface area were studied. The results illustrated the importance of incorporating the unique IoT characteristics in designing Byzantine-based Blockchain systems for IoT networks. They suggested enhancing the semi-distributed BFT (Byzantine Fault Tolerance) algorithms by partitioning the large-scale IoT network and reducing the communication complexity to facilitate and enable the wide adoption of Blockchain in the IoT context. However, sending large-scale data to the validator nodes (consensus protocol) will introduce another network and bandwidth bottleneck.

Ahmed et al. [5] have proposed a Blockchain-based solution to the problem of mitigating Mirai botnet attacks on IoT devices. The solution depends on dividing the network into AS (Autonomous Systems) which communicate through the Blockchain network to share malicious node information. Blockchain platforms are used to store and share a list of IP addresses of different hosts connected to an AS, to show which of these have been identified as malicious. The proposed approach is simulated on a custom simulator, which is adapted to use an appropriate value for the malicious threshold.

The authors are interested in finding the delay incurred in propagating the information of malicious hosts between the AS over an Ethereum Blockchain. To this end, they have simulated a scenario in which the AS is very large and thousands of malicious hosts are connected to the AS. The simulation indicates that the proposed approach, when appropriately tuned, can yield a true detection rate of 95%.

Lingering Zhao and Jiangshan Yu [44], have studied an alternative of Blockchain technology for integration of lightweight IoT devices. They have summarized the central features of IoTA by analyzing the functionality of this Directed Acyclic Graph (DAG)-based model. DAG-based examples are expected to provide greater scalability and fast economical benefits that most of the Blockchain-based platforms fail to accomplish. They have shown that, IOTA has a transaction rate under 20 tps according to live transaction monitor provided by the Tangle[40] A peak transaction rate more than 400 tps has been seen in IOTA before and it has been handled well with a confirmation ratio over 99%. Although, IoTA [22] is much more faster than Blockchain. However because there is neither validator nodes nor blocks to participate in consensus and integrity of the data, it is shown that IoTA is still vulnerable to double-spending attacks [31].

2.2 Integration of Blockchain and Self-Adaptive Systems

Alzahrani and Bulusu [8], proposed "Block-Supply", a decentralized anti-counterfeiting supply chain that exploits NFC (Near-field communication) and Blockchain technologies. This Block-Supply chain can detect modifications, cloning, and tag reapplication attacks, in addition to tracking products without a centralized managing server. The authors developed a new truly decentralized consensus protocol that does not need PoW and dynamically uses a set of validators of varying size each time a new block is proposed based on random algorithms. The proposed protocol employs a game-theoretical model to analyze the risk likelihood of the block's proposing nodes. Likewise, the proposed protocol uses a novel, decentralized, dynamic mapping between the nodes that participate in the consensus process. The protocol protects against several real attacks mounted by powerful adversaries. The simulation results depict that the new protocol is scalable for large networks using a relatively small number of validators. Moreover, it maintains a satisfactory level of security. However, concerning the simulation, they have used OMNET++ [43], which is not considered to be an ideal real-time simulation for IoT applications.

Liaskos et al. [33] proposed a simple model for quantitative reasoning about the parameters that impact and are influenced by the consensus process of public open-access Blockchain networks and attempted one of the first formulations of such a model as an adaptive system. To motivate the approach, the authors experimented with an idealized controller, only to subsequently analyze it and reveal the challenges in designing real-world controllers. The authors identified variables that influence its sustainability, including those imposed by the environment and those that can be directly configured by the network's governance. Next, they proposed a model to show how these variables relate and affect each other. With that model underway, they worked out the problem as a standard control engineering problem in which a controller (Blockchain governance) optimizes the parameter choices so that the output of the controlled system (Blockchain network) meets certain objectives.

Casado-Vara et al. [18] proposed a new adaptive strategy for Blockchain-based system by exploiting game theory and prediction of accuracy of future states to reduce the tracking error and enhance the effectiveness of the algorithm aiming at ensuring the efficiency of an adaptive temperature control algorithm. The authors have designed a Blockchain-based platform to enhance the operation of the monitoring and control of the IoT networks to improve energy efficiency. This control system optimizes the temperature of a smart building uses state prediction module, which uses Markov chains to predict the accuracy states of IoT nodes in future time. The experiment results indicate that the predicted temperature signal is surrounded by a small interval close to the collected temperature data.

Hovland and Kucera [26] designed and implemented a self-adaptive simulation model for the Proof-of-Work consensus algorithm in Blockchain. The model has been validated by a statistical analysis of miner solutions to PoW challenges with varying difficulty levels from a sample of more than 500,000 solutions. The controller in their architecture consists of a nonlinear inverse model, which estimates the next difficulty level from the desired block time, the current average block time, and the previous difficulty level. By using the simulation model, controller designs can be tested and analyzed in a few seconds compared to typically several days for implementation on a test network. The purpose of the study is to develop a Blockchain difficulty adjustment controller and to study the stability of the closed-loop system from a control engineering perspective. Their results show that their model has fast response and high sability in all cases.

3 BACKGROUND

3.1 Consensus in Blockchain

Consensus in Blockchain is introduced to address two problems, namely the Byzantine Generals Problem [30] and Double Spending. Double Spending means that we can not reuse the digital currency in two separate transactions at the same time. Blockchain can address this issue by verifying all the transactions by several distributed nodes in the decentralized network, and not by a single authority. This is more robust especially when the single authority can also be corrupted. The Byzantine Generals problem is a common issue in distributed environments. The data in the network will be delivered between several nodes through peer-to-peer communications. However, some of the nodes in the distributed network could act maliciously which can corrupt data. Healthy nodes need to able to distinguish information delivered to them that has been altered legitimately and obtain consistent results with other healthy nodes. Byzantine Fault Tolerance implies that the system has enough nodes, so that it is resilient in the presence of some malicious nodes. The consensus protocol in Blockchain has been designed to provide data verification (by consensus) and fault tolerance [34].

Proof of Work (PoW) is the first consensus algorithm that has been used in Blockchain. Its core idea is to give rewards based

on hashing power competitions among peers in the Blockchain network. Each node calculates and solves a specific mathematical problem. The first node which is successful to solve the mathematical puzzle will be allowed o create the next block and will be rewarded a certain amount of currency, which could be either Bitcoin or the Ether currency in the Ethereum network [35]. PoW is computationally expensive and has a time overhead which is undesirable for real-time IoT applications [23].

Next, we will discuss three alternative consensus protocols that are introduced in Hyperledger Sawtooth.

3.1.1 Proof of Elapsed Time . Proof of Elapsed Time (PoET) is the main consensus protocol that was introduced by Intel in Hyperledger Sawtooth. The algorithm randomly chooses the next leader to finalize the block. This consensus protocol employs this method to deal with malicious peers in an open-ended Blockchain network. PoET utilizes the Trusted Execution Environment (TEE) which is called Enclave inside of Intel processors to prevent cheating and to provide confidentiality, integrity, and blacklisting malicious behaviors based on asymmetric key cryptography and an additional set of election policies[2, 13].

3.1.2 Practical Byzantine Fault Tolerance . The PBFT consensus algorithm is based on the Byzantine Fault Tolerance principles, which can tolerate that less than one third faulty nodes of total nodes in the $n = 3f + 1$ total nodes in the network. PBFT extends this principle to the concept of consensus: at least $2f + 1$ nodes need to agree to reach the consensus and confirm transactions [42].

3.1.3 Raft Algorithm . The Raft [37] algorithm can be used in private Blockchains, where an administrator of the network can add or remove nodes. This algorithm is proposed to solve the inefficiencies of PBFT. It is a lead-based algorithm which means that the peers of the network will choose the leader in an election process. This means that only the leader node is allowed to publish blocks which are validated by other peers. This algorithm cannot tolerate any malicious nodes but it can tolerate up to 50% crash faulty nodes. Since in a private Blockchain all the peers are verified by the administrator, this algorithms aims at resolving crash faults other than Byzantine faults [28].

4 STUDY METHODOLOGY AND EXPERIMENTAL SETUP

With respect to our first objective, our intention is to evaluate the performance and scalability of the various consensus algorithms in Hyperledger Sawtooth to guide the right decisions and to motivate the design for our self-adaptive system. As we mentioned before, Hyperledger Sawtooth has four main consensus protocols, namely Raft, PoET, Devmode, and PBFT. As Devmode is only considered for testing purposes, but not for the production network, we decided to experiment only with the other three protocols. The three algorithms were compared based on their resource consumption and latency in the context of a simulated IoT application.

To evaluate the performance of the system, we created a custom workload generator that simulates data generated by sensors in a smart home environment. In our simulated environment, we assume that the smart home is equipped with several IoT devices, including surveillance cameras, thermostats, door sensors, GPS

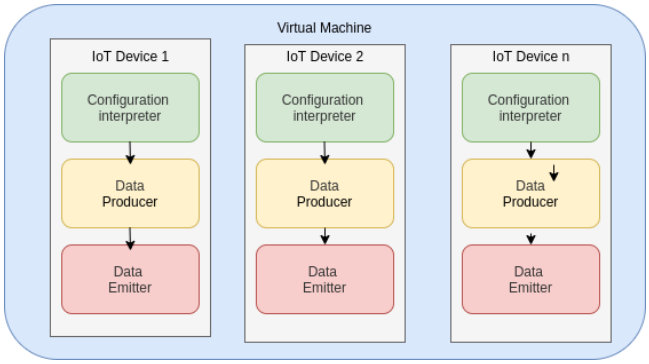


Figure 1: Simulated IoT environment

sensors and sound sensors (capturing random noise). The devices differ with respect to the transmission rate and the amount of data transmitted. For example, GPS sensors send a position report every 2 seconds, thermostats report the temperature every 1 second and door sensors report the state of a door (Open / Closed) every 5 seconds. The size of data is ranging from few bytes to few kilobytes based on the type of sensor. These devices would send their data to a gateway and the gateway pushes the data to a Blockchain database which in our case is a Hyperledger Sawtooth network. Figure 1 shows the architecture of our simulated IoT lab, which uses python scripts and Docker containers to simulate each device, designed according to instructions set forth by Ramprasad et al.[41]. We specify the number of IoT devices, type of devices (camera, thermostat, GPS, door sensor device and sound device), time-scheduling in the Configuration Interpreter, the Data Produces component produces the data and Data Emitter transfers the data to the Blockchain for storing the produces data. To implement the virtual IoT environment, every sensor is implemented as a python script, with a message that corresponds to the appropriate size of data and transmission rate, deployed on a Flask web server hosted by a Docker container. In addition, we use Nginx as the web server that implements the gateway and accepts data from the sensors and HAProxy is a load balancer deployed in front of the Blockchain. The devices first generate and transmit their data to the gateway, as presented in Figure 2, and then the gateway sends it to the Sawtooth network to be stored securely.

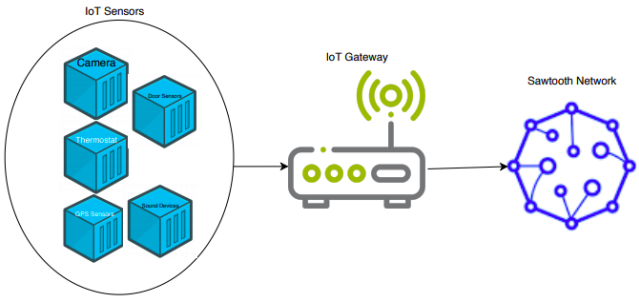


Figure 2: Architecture of experimental setup

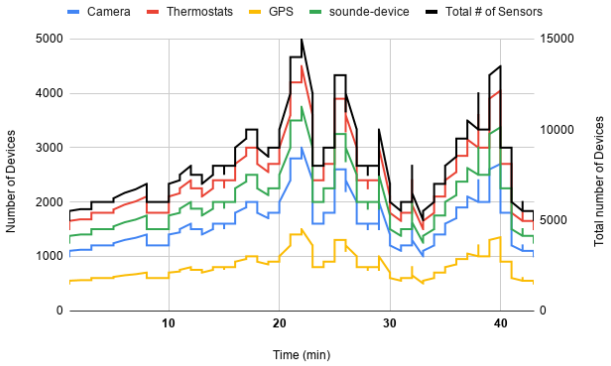


Figure 3: The distribution of devices over time

In our experiments, the number of sensors varies over time but the distribution of sensors and devices is kept relatively stable: 20% cameras, 30% thermostats, 10% GPS, 15% door sensor devices and 25% sound devices. Figure 3 shows that our experimental study takes about 43 minutes per experiment. During this time, the number of IoT devices continuously increases and decreases to periodically stress and saturate the network. The change in load is that which will trigger the adaptation and will stress the Blockchain system. Besides that, the configuration of the IoT network is kept constant throughout the experiment, since our goal is to examine the impact of load on the consensus algorithm and by extent on the performance of the Blockchain, so we try to eliminate the effect of different configurations.

The data we are storing in our scenario includes: Device ID, Time Stamp, Message, Message Size, Message Hash and Device Type. It is important to mention that the purpose of this IoT simulation is only to saturate the network with random data as much as possible to evaluate the performance of the consensus protocols in the same scenario. However, in a real case, in order to reduce the size of the network, we could store data off-chain and record only the hash on Blockchain for immutability, aggregate the data and store only relevant portions on Blockchain. Since we need to verify all the data by all validators, it will not be suitable for large volumes of data on a permanent Blockchain store, especially those that may by relevant only for a short time. In our experiments, our goal is to cause as much saturation as possible, so we send and store all data, under a "worst case scenario".

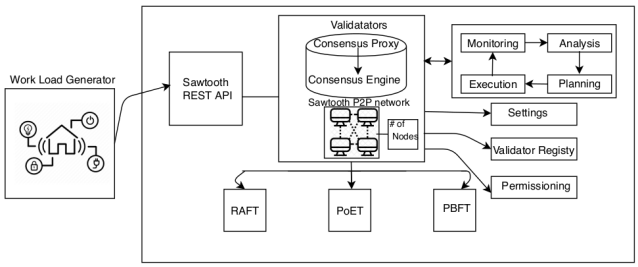


Figure 4: Our Sawtooth Network

Figure 4 demonstrates the Hyperledger Sawtooth infrastructure and its components as it was used in our experiments. We used multiple containers to deploy Sawtooth as the default framework consists of multiple processes, including among others the supply Rest API, validator nodes, the consensus engine, a Postgres database, and others. By default, the consensus protocol is designed to be "Devmode", which is a simplified random-leader consensus algorithm mostly for testing purposes. For our experiments, we have made some small modifications to Sawtooth to enable dynamic configuration of the consensus protocol [4]. This allowed us to evaluate the different consensus algorithm under our simulated algorithm, but it also enabled the implementation of our self-adaptive system. We have applied the default configuration for each consensus algorithm based on Sawtooth documentation [4] guidelines.

The infrastructure of our experiment was deployed on the Amazon EC2 cloud. One virtual machine with 8 VCPUs and 15 GB RAM (rc4.2XLARGE) was used to deploy the Hyperledger Sawtooth. Docker and Docker-compose were installed in this machine to enable the use of containers. The containers that corresponded to validators, which run the consensus protocol, and were hosted by this VM were limited to use only 25% of the total CPU of the host VM. It has been shown that when left unrestricted containers can take up all the resources of the host VM, rendering scaling ineffective [24]. Another VM with 2 VCPUs and 8 GB RAM (T2.Large) was used for the simulation of the IoT workload. Both machines ran Ubuntu 18.03. As mentioned earlier, we evaluated performance in terms of CPU consumption and response time. For CPU, we used the Docker Remote API [10] to collect CPU consumption for the Sawtooth network. For response time, the python script for data emission by every sensor was configured to receive back a response from the Sawtooth network and record the response time.

5 SELF-ADAPTIVE SYSTEM DESIGN

In order to implement our self-adaptive mechanism to enable the dynamic configuration of the consensus protocol, we followed the MAPE-k architecture [25] with four componets: a) a monitoring component which gathers CPU data from Docker and response time using our custom script, b) an analysis component that checks if the CPU has crossed the predefined thresholds, c) a planning component that will decide to add or remove a validator, based on CPU consumption, and to dynamically change the consensus algorithm, and d) an execution component that connects to Docker to scale the validators or to Sawtooth to change the consensus algorithm. We have integrated our MAPE-k framework to the Sawtooth network as illustrated in Fig. 4.

The analysis component checks the average CPU consumption of the Sawtooth validator containers and if that surpasses 70%, our MAPE-k system automatically adds a new validator node in the network. Preliminary tests have shown that when there is an increasing number of peers in the network (sensors in our case), Sawtooth can handle the traffic to validators and it crashes. Increased CPU consumption can be an indicator for impending crashes, so scaling of the validators is an appropriate action. Conversely, when average CPU drops below 30%, the analysis component signals the removal

of a validator in order to reduce costs, but also to speed up the validation process as less nodes are necessary for a reduced traffic.

The planning component has to make two decisions. The first concerns the scaling of the validator cluster as described above. The second decision concerns the choice of a consensus algorithm. Preliminary results have shown that the size of the network and the number of validators impacts the performance of consensus differently for each algorithm. According to Sawtooth documentation, each consensus protocol has different communication complexity. As was mentioned earlier, the number of nodes in the network has an impact on the performance of the system and we are interested to experimentally measure the threshold when the performance of the system decreases and at the same time to see which consensus protocol outperforms when adding more nodes. The planning phase for scaling and for the choice of consensus algorithm is described in Algorithm 1. The thresholds of validator nodes to trigger a change in the consenus algorithm were determined empirically, based on the results of our study, as presented in Section 6.

It is worth noting that while performance is a crucial aspect of distributed systems, and IoT systems in particular, the main motivation behind using Blockchain is security. We have studied relevant studies and corresponding documentation to assess the security capabilities of each algorithm and design our planning module accordingly. PoET is deemed the most vulnerable algorithm since an adversary can jeopardized the integrity of the network by compromising $\theta(\frac{loglogn}{logn})$ of the total participating nodes in the network [20]. This implies that this algorithm can be more secure when the number of nodes is relatively large. However, based on our experimental study (see Section 6), PoET is the most performant of the three in terms of resource consumption and average response time even when the network scales up to a large number of nodes. The Raft algorithm can tolerate 50% crashed faulty nodes and PBFT can tolerate 1/3 of faulty nodes [7].

Based on our performance results, Raft algorithm performance (in terms of both resp one time and resource consumption) reduces when it scales to 9 nodes thereby, PBFT can perform better than RAFT even with more nodes. For this reason in our self-adaptive mechanism, we start with Raft algorithm, when it reaches to 9 nodes we switch to PBFT and when PBFT reaches to 15 nodes as both resource consumption and response time start worsening we will switch to the PoET as it can assure the security with a large number of nodes.

Algorithm 1 Self-Adaptive framework for Hyperledger Sawtooth for choosing the right consensus algorithm

1: **Input:** *Nodes — the number of Validator nodes in the Sawtooth network*
2: **Input:** *Consensus-Algorithm - Retrieves the current Consensus Algorithm of the Sawtooth Protocol*
3: **Input:** *utilization — the average CPU utilization of containers in a VM*
4: **Input:** *lower threshold and upper threshold — the limits of the desired range for the CPU consumption of Validator containers in a VM*
5: **if** if utilization ≥ 70 **then**
6: Validator = Validator + 1;
7: **else if** utilization ≤ 30 **then**
8: Validator = Validator - 1;
9: Consensus-Algorithm = Raft
10: **if** Nodes < 9 **then**
11: Consensus-Algorithm = PBFT;
12: **else if** Nodes ≥ 9 **then**
13: Consensus-Algorithm = Raft;
14: **else if** Nodes > 14 **then**
15: Consensus-Algorithm = POET;
16: **else if** Nodes ≤ 14 **then**
17: Consensus-Algorithm = PBFT;
18: **else**
19: Consensus-Algorithm = Raft;

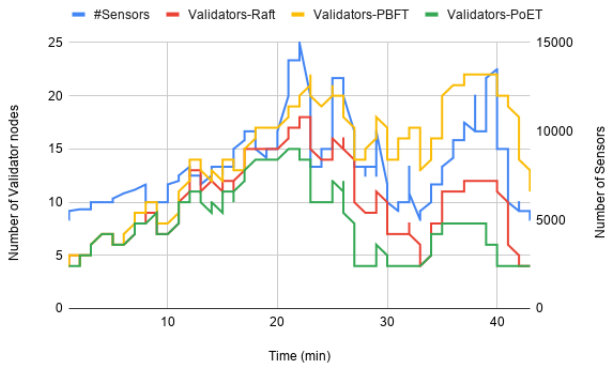


Figure 5: The distribution of number of sensors to all three validators for the studied consensus algorithms

6 EXPERIMENTAL RESULTS

6.1 Performance Evaluation of consensus protocols in Hyperledger Sawtooth

The first set of experiments concerns the study of how each consensus algorithm performs in terms of CPU consumption and response time under a varying number of peers in the Blockchain network. Since we know that none of the configurations automatically scales according to the size of the network and we also know that if the validators are not scaled, the network crashes, we enabled a simple scaler (lines 5-8 of Algorithm 1) to protect the system and be able to continue our experiments. Figure 5 shows the number of validators for Raft (red line), PBFT (yellow line) and PoET (green line) against the number of sensors (blue line) representing the size of the Blockchain network. As it is obvious from the graph, the number of validators increases and decreases according to the corresponding fluctuations to the size of the network. However, it is also obvious that PoET consistently need less validators compared to the other two algorithms for the same number of sensors, while we find PBFT at the opposite end of this comparison. In fact, as it is summarized in Table 1, PoET needs an average of 7.5 validators throughout the experiment, compared to 15.3 for PBFT and 10 for Raft. At the peak of the traffic (just after the 20th minute of the experiment), PoET requires a maximum of 15, while PBFT and Raft require 22 and

18 respectively. Already, this shows that PoET is a close optimal choice concerning cost-efficiency and resource utilization.

Table 1: Summary of results of comparison between Raft, PBFT and PoET

Algorithms	Raft	PBFT	PoET
Avg Validators	10	15.3	7.5
Max Validators	18	22	15
Avg CPU	51.26%	55.70%	45.72%
St.Dev. CPU	24.92	23.14	24.44
Avg Response time (ms)	874.12	853.04	759.14
St.Dev. Response time	100.96	122.64	106.62

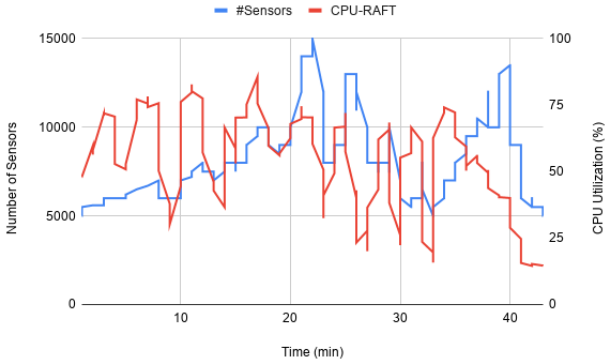


Figure 6: Raft CPU Utilization VS Number of Sensors

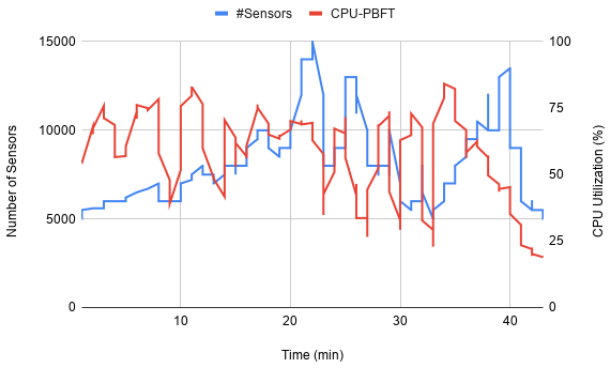


Figure 7: PBFT CPU Utilization VS Number of Sensors

When looking at CPU consumption, results do not differ much. PoET is by far the most efficient algorithm recording an average CPU consumption of 45.72%. At the other end, PBFT is still the worst of the three with 55.7%, while Raft is the middle solution with an average CPU consumption of 51.26%. Figures 7, 6, and 8 show graphically the progress of CPU consumption for PBFT, Raft and

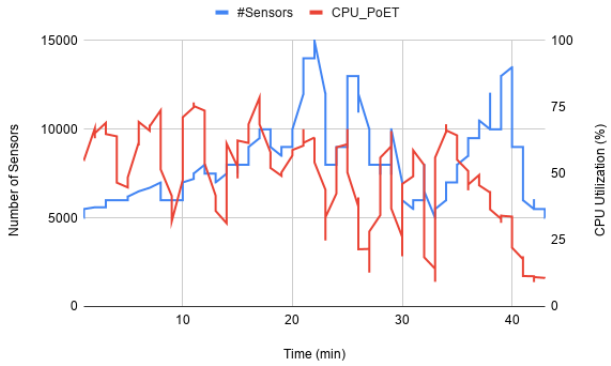


Figure 8: PoET CPU Utilization VS Number of Sensors

PoET respectively. One first observation is that CPU *oscillates* (drastically fluctuates), especially close to scaling action. The reason for this is that the impact of a scaling action is prominent to the output (in our case, CPU) especially right after the action and especially if the system is small (as is the case for the number of validators in our experiments). This condition is systemic throughout our experiments, in the sense that it is present in all three algorithms, so it does not affect our conclusions. To further minimize this impact, we should not that after taking any scaling or reconfiguration action, our analysis freezes for 10 seconds to allow the system to settle and avoid opposite scaling actions immediately after. Nevertheless, our monitoring module is not stopped, which is the reason why the oscillations are visibile in the graphs.

Finally, PoET is also the winner when considering response time with an average of about 760 ms. However, in this case we have a change for the second most performant algorithm, where PBFT had 853 ms average response time, while Raft had 874 ms. Nevertheless, PBFT had a more variance (122 standard deviation) compared to Raft, which implies that their difference may be neither significant nor consistent. The difference between PBFT and Raft is visible in Figures 10 and 11 respectively, where we see that in Raft response time stays higher for a longer period. Figure 9 shows clearly that PoET maintains low response time at all times.

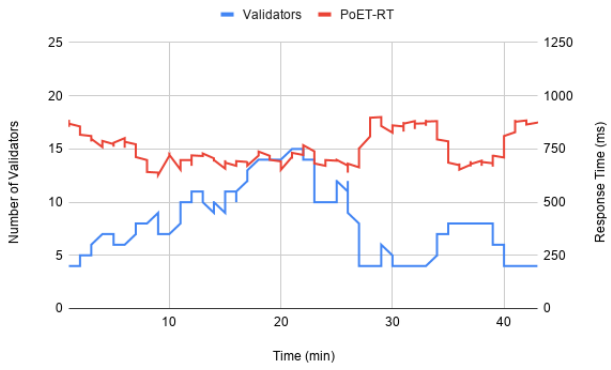


Figure 9: PoET Response Time VS Number of Validators

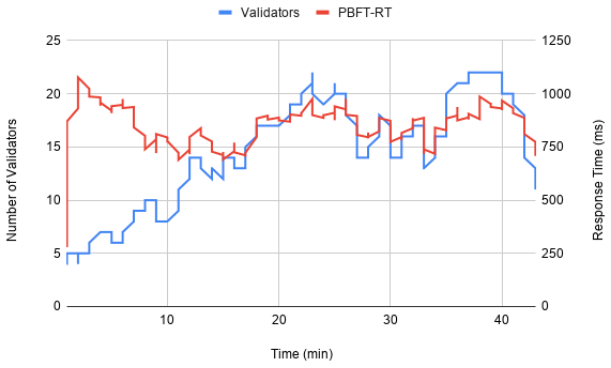


Figure 10: PBFT Response Time VS Number of Validators

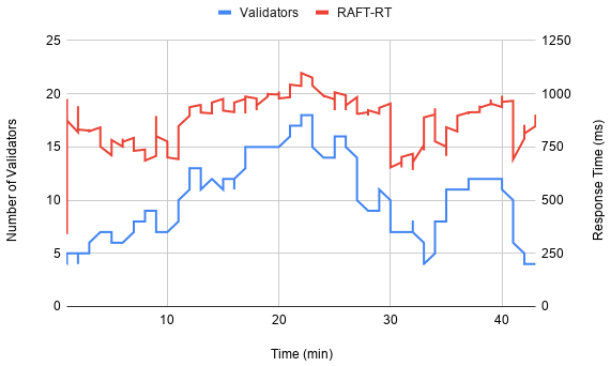


Figure 11: RAFT Response Time VS Number of Validators

Looking at Figures 9, 10, and 11 with the progress of response time for PoET, PBFT and Raft respectively, we can observe that scaling validators has a positive impact on response time as the size of the network grows, until we reach a certain point where there is a plateau and the response time remains at least constant even when we keep adding validators. The interesting observations is this point is different for each algorithm. We found that the plateau is reached for about 9 validators for PoET and Raft and for almost 14 validators for PBFT. This also roughly corresponds to the average number of validators necessary for each algorithm. We can see that from this perspective PoET is once again the most efficient of the three algorithms. However, as mentioned before, PoET is not as robust and secure with a small number of validators. Given its increased security thanks to higher number of validators and its ability to maintain good performance under this condition, we argue that PoET is the recommended consensus algorithm, when the size of the network increases considerably.

The findings of this study helped us to design our self-adaptive mechanism to dynamically reconfigure the consensus protocol of Hyperledger Sawtooth at run-time. When we have a small-sized network and a small number of validators (less than 9) is sufficient, our MAPE-k tool picks the robust PBFT algorithm. For medium-sized networks where less than 14 validators are necessary, MAPE-k picks the Raft algorithm. Finally, when the required validators exceed 14 and we need to maintain good performance, MAPE-k recommends the use of PoET (see Algorithm 1).

6.2 Evaluation of the Self-Adaptive mechanism

When testing our self-adaptive mechanism, we enable in the planning module the dynamic reconfiguration of the consensus protocol (lines 9-19 in Algorithm 1) on top of the simple scaler (lines 5-8 in Algorithm 1). By default our MAPE-k system starts with Raft as the consensus algorithm and changes based on the number of validator nodes deployed in the Sawtooth system. Figure 12 shows the number of validator nodes (blue for Raft, yellow for PBFT and green for PoET) deployed as the response to the fluctuation in the number of sensors (black line). As it can be seen, the line for valdiators is colored differently for each consensus algorithm and a change in color indicates a decision to switch algorithms. For example at time 8 minutes, the consensus has been changed from Raft (blue) to PBFT (yellow). In addition, Figures 13 and 14 show the CPU consumption and the response time of the system, when of our MAPE-k system is deployed. Finally, Table 2 summarizes the results of this experiment.

As it can be seen, our adaptive consensus protocol performs exceptionally well compared to Raft and PBFT, while its performance is comparable to that of PoET, 11 validators on average against the 10 of PoET, 50% average CPU consumption against 45% of PoET and 784 ms response time compared to 759 ms of PoET. The additional benefit, as it is obvious from the graphs, is that PoET is deployed at higher traffics with a large number of validators, exactly when it is the most powerful.

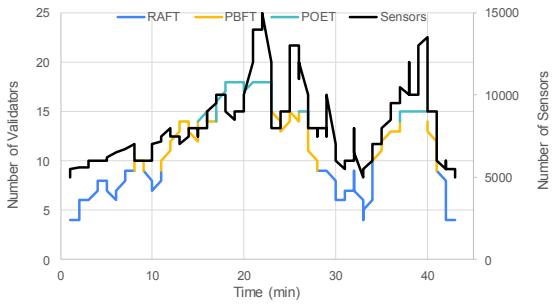


Figure 12: Number of validators VS number of sensors. The change in color indicates a switch of consensus algorithm by the self-adaptive mechanism.

7 THREADS TO VALIDITY

In this study, we have designed a self-adaptive mechanism that can efficiently choose and deploy a different consensus algorithm at run-time. To evaluate the system, we have simulated a virtual IoT lab environment that aims to store its generated data in the Blockchain database. For our future study, we plan to use real IoT devices that

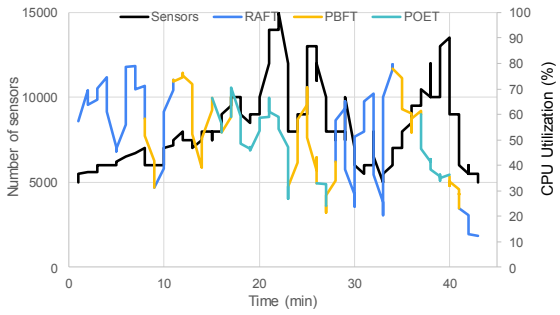


Figure 13: CPU Utilization VS Number of Sensors. The change in color indicates a switch of consensus algorithm by the self-adaptive mechanism.

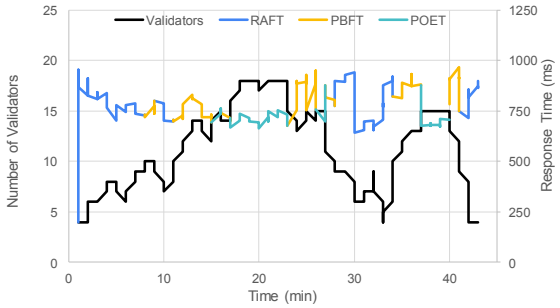


Figure 14: Response Time VS Number of Validators. The change in color indicates a switch of consensus algorithm by the self-adaptive mechanism.

Table 2: Summary of experimental results for self-adaptive system

Algorithm	SA
Avg Validators	10.98
Max Validators	18
Avg CPU (st.dev.)	49.96% (22.57)
Avg Response Time (ms) (st.dev.)	784.22 (97.49)

will produce and send their actual data to the Blockchain server to see how it impacts the network. In this study, we experimented with synthetic data to better control the environment. There can exist other parameters that can impact performance, such as such as messaging format (JSON vs. Protobug), communication protocols (such as GRPC), security, or any other unique workload characteristics of IoT devices. It is our intention to include these factors in other future studies.

Another threat to validity could be the number of considered consensus protocols. We specifically chose to focus on PBFT, PoET and Raft, because these are all supported by Hyperledger Sawtooth. By using a single Blockchain platform, thus a homogeneous hosting platform, we eliminated the impact of other factors and studied only the impact of the consensus protocol. Naturally, our intention is to study more algorithms in the future.

To conduct study, we have used the suggested default configuration of consensus protocols in the Hyperledger Sawtooth network. It is expected that special configurations may have render an algorithm much more efficient than we reported in our study. By keeping the same default configurations for all algorithms, we made sure that the comparisons were fair and under the same given conditions for all algorithms. The exploration of different configurations is another possible future step of this work.

Lastly, for our experiment , we have used a single VM, which implies the network I/O is negligible. For future studies, we plan to use different VMs for the nodes and see the effect of the network I/O on the performance.

8 CONCLUSION

Although Blockchain is an emerging technology among practitioners to improve security issues for distributed systems, it could also prove inefficient in terms of bandwidth and cost, especially for IoT systems. In this study, we first compared three popular consensus algorithms for Blockchain, namely PBFT, Raft and PoET, to see whether there is a single best with respect to time and computation overheads or if there are trade-offs between the three consensus protocols in Hyperledger Sawtooth. From the empirical analysis, we have observed that PoET is the most efficient consensus algorithm comparably to RAFT and PBFT, but probably not as robust and secure for small networks. Conversely, the performance of Raft and PBFT deteriorates for larger-sized networks.

Based on our performance evaluation and theoretical security analysis of the consensus protocols, we have designed a novel self-adaptive mechanism to dynamically and automatically choose and deploy the right consensus algorithm at run-time. Our results demonstrate that our self-adaptive mechanism has better CPU utilization comparably to PBFT and RAFT, and average Response Time is also better than both RAFT and PBFT. In theory, our adaptive consensus protocol addresses the security limitations of PoET for small networks while maintaining the same performance standards for larger networks. In the future, we plan to extend our study to also include these security concerns in a concrete manner.

In this study, we have focused on reducing the cost and assessing the security based on the number of nodes. For future research, we are planning to explore how the framework can be used if the optimization target changes regarding other security aspects (e.g., the ones related to Blockchain forks). Moreover, another interesting subject to explore for future studies is to analyze algorithms' performance concerning the number or frequency of the transactions and possibly the characteristics of these transactions.

ACKNOWLEDGMENT

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) Collaborative Research and Training Experience (CREATE) Dependable Internet of Things Applications (DITA) program.

REFERENCES

[1] 2020. DevMode Consensus, . https://sawtooth.hyperledger.org/docs/core/nightly/1-2/sysadmin_guide/about_dynamic_consensus.html#devmode-consensus-label. Accessed: 2020-05-20.

[2] 2020. PoET Consensus, . https://sawtooth.hyperledger.org/docs/core/releases/latest/architecture/poet.html. Accessed: 2020-05-20.

[3] 2020. Sawtooth Documentation for Supporting Different Consensus Protocols, . https://sawtooth.hyperledger.org/faq/consensus//. Accessed: 2020-05-20.

[4] 2020. Sawtooth Dynamic Consensus, . https://sawtooth.hyperledger.org/docs/core/releases/latest/sysadmin_guide/about_dynamic_consensus.html. Accessed: 2020-05-20.

[5] Zohaib Ahmed, Syed Muhammad Danish, Hassaan Khaliq Qureshi, and Marios Lestas. 2019. Protecting IoTs from Mirai Botnet Attacks Using Blockchains. In *2019 IEEE 24th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. IEEE, 1–6.

[6] Maha Alaslani, Faisal Nawab, and Basem Shihada. 2019. Blockchain in IoT Systems: End-to-End Delay Evaluation. *IEEE Internet of Things Journal* 6, 5 (2019), 8332–8344.

[7] Shikah J Alsunaidi and Fahd A Alhaidari. 2019. A Survey of Consensus Algorithms for Blockchain Technology. In *2019 International Conference on Computer and Information Sciences (ICCIS)*. IEEE, 1–6.

[8] Naif Alzahrani and Nirupama Bulusu. 2020. A new product anti-counterfeiting blockchain using a truly decentralized dynamic consensus protocol. *Concurrency and Computation: Practice and Experience* 32, 12 (2020), e5232.

[9] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. 2017. Understanding the mirai botnet. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*. 1093–1110.

[10] Docker Remote API. 2019. *Docker Remote API, 2019*. Accessed: 2019-11-02.

[11] Luigi Atzori, Antonio Iera, and Giacomo Morabito. 2010. The internet of things: A survey. *Computer networks* 54, 15 (2010), 2787–2805.

[12] Yu Nandar Aung and Thitinan Tantidham. 2017. Review of Ethereum: Smart home case study. In *2017 2nd International Conference on Information Technology (INCIT)*. IEEE, 1–4.

[13] Arati Baliga. 2017. Understanding blockchain consensus models. In *Persistent*.

[14] Shehar Bano, Mustafa Al-Bassam, and George Danezis. 2017. The road to scalable blockchain designs. *USENIX; login: magazine* (2017).

[15] Martijn Bastiaan. 2015. Preventing the 51%-attack: a stochastic analysis of two phase proof of work in bitcoin. In *Availab le at http://referaat. cs. utwente. nl/conference/22/paper/7473/preventingthe-51-attack-a-stochasticanalysis-oftwo-phase-proof-of-work-in-bitcoin. pdf*.

[16] Aymen Boudguiga, Nabil Bouzerna, Louis Granboulan, Alexis Olivereau, Flavien Quesnel, Anthony Roger, and Renaud Sirdey. 2017. Towards better availability and accountability for iot updates by means of a blockchain. In *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 50–58.

[17] Roberto Casado-Vara, Pablo Chamoso, Fernando De la Prieta, Javier Prieto, and Juan M Corchado. 2019. Non-linear adaptive closed-loop control system for improved efficiency in IoT-blockchain management. *Information Fusion* 49 (2019), 227–239.

[18] Roberto Casado-Vara, Fernando De la Prieta, Sara Rodriguez, Ines Sitton, Jose L Calvo-Rolle, G Kumar Venayagamoorthy, Pastora Vega, and Javier Prieto. 2019. Adaptive Fault-Tolerant Tracking Control Algorithm for IoT Systems: Smart Building Case Study. In *International Workshop on Soft Computing Models in Industrial and Environmental Applications*. Springer, 481–490.

[19] Miguel Castro, Barbara Liskov, et al. 1999. Practical Byzantine fault tolerance. In *OSDI*, Vol. 99. 173–186.

[20] Lin Chen, Lei Xu, Nolan Shah, Zhimin Gao, Yang Lu, and Weidong Shi. 2017. On security analysis of proof-of-elapsed-time (poet). In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*. Springer, 282–297.

[21] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. 2017. Blockbench: A framework for analyzing private blockchains. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 1085–1100.

[22] M Divya and Nagaveni B Biradar. 2018. IOTA-next generation block chain. *International Journal Of Engineering And Computer Science* 7, 04 (2018), 23823–23826.

[23] Ali Dorri, Salil S Kanhere, Raja Jurdak, and Praveen Gauravaram. 2017. Blockchain for IoT security and privacy: The case study of a smart home. In *2017 IEEE international conference on pervasive computing and communications workshops (PerCom workshops)*. IEEE, 618–623.

[24] Marios Fokaefs, Cornel Barna, Rodrigo Veleda, Marin Litoiu, Joe Wigglesworth, and Radu Mateescu. 2016. Enabling devops for containerized data-intensive applications: an exploratory study. In *Proceedings of the 26th Annual International Conference on Computer Science and Software Engineering*. 138–148.

[25] P Horn. 2001. Autonomic computing: IBM's perspective on the state of information technology, IBM Corporation.

[26] Geir Hovland and Jan Kucera. 2017. Nonlinear feedback control and stability analysis of a proof-of-work blockchain. (2017).

[27] Jenalea Howell. 2017. Number of connected iot devices will surge to 125 billion by 2030. *IHS markit says* (2017).

[28] Dongyan Huang, Xiaoli Ma, and Shengli Zhang. 2019. Performance analysis of the raft consensus algorithm for private blockchains. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* (2019).

[29] Nir Kshetri. 2017. Can blockchain strengthen the internet of things? *IT professional* 19, 4 (2017), 68–72.

[30] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4, 3 (1982), 382–401.

[31] Yixin Li, Bin Cao, Mugen Peng, Long Zhang, Lei Zhang, Daquan Feng, and Jihong Yu. 2020. Direct Acyclic Graph-Based Ledger for Internet of Things: Performance and Security Analysis. *IEEE/ACM Transactions on Networking* (2020).

[32] Wei Liang, Mingdong Tang, Jing Long, Xin Peng, Jianlong Xu, and Kuan-Ching Li. 2019. A secure fabric blockchain-based data transmission technique for industrial Internet-of-Things. *IEEE Transactions on Industrial Informatics* 15, 6 (2019), 3582–3592.

[33] Sotirios Liaskos, Bo Wang, and Nahid Alimohammadi. 2019. Blockchain networks as adaptive systems. In *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 139–145.

[34] Du Mingxiao, Ma Xiaofeng, Zhang Zhe, Wang Xiangwei, and Chen Qijun. 2017. A review on consensus algorithm of blockchain. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2567–2572.

[35] Satoshi Nakamoto. 2008. Bitcoin whitepaper. *URL: https://bitcoin. org/bitcoin. pdf-(: 17.07. 2019)* (2008).

[36] Kelly Olson, Mic Bowman, James Mitchell, Shawn Amundson, Dan Middleton, and Cian Montgomery. 2018. Sawtooth: An Introduction. *The Linux Foundation, Jan* (2018).

[37] Diego Ongaro and John Ousterhout. 2014. In search of an understandable consensus algorithm. In *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*. 305–319.

[38] Diego Ongaro and John Ousterhout. 2015. Raft consensus algorithm. (2015).

[39] Fredrik Österlind, Adam Dunkels, Joakim Eriksson, Niclas Finne, and Thiemo Voigt. 2006. Cross-level sensor network simulation with cooja. In *First IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006)*.

[40] Serguei Popov. 2016. The tangle. *cit. on* (2016), 131.

[41] Brian Ramprasad, Marios Fokaefs, Joydeep Mukherjee, and Marin Litoiu. 2019. EMU-IoT-A Virtual Internet of Things Lab. In *2019 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 73–83.

[42] Harish Sukhwani, José M Martínez, Xiaolin Chang, Kishor S Trivedi, and Andy Rindos. 2017. Performance modeling of pbft consensus process for permissioned blockchain network (hyperledger fabric). In *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 253–255.

[43] Andras Varga. 2010. OMNeT++. In *Modeling and tools for network simulation*. Springer, 35–59.

[44] Liangrong Zhao and Jiangshan Yu. 2019. Evaluating DAG-Based Blockchains for IoT. In *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 507–513.

Parallel Windowed Method for Scalar Multiplication in Elliptic Curve Cryptography

Tanya Bouman
McMaster University
Hamilton, Ontario
boumante@mcmaster.ca

Yusra Irfan
McMaster University
Hamilton, Ontario
yirfan3@uwo.ca

James You
McMaster University
Hamilton, Ontario
youy2@mcmaster.ca

Christopher K. Anand
McMaster University
Hamilton, Ontario
anandc@mcmaster.ca

ABSTRACT

Commercial software applications, such as permissioned Blockchains, are increasingly dependent on Elliptic Curve Cryptography for digital signatures. Elliptic Curve Cryptography uses a group operation, point addition, in the set of points on an elliptic curve over a prime field. Scalar multiplication is the repeated addition of a fixed point P in the curve. A common optimization for scalar multiplication, the windowed method, decomposes the number of additions into nibbles or other digits, using a pre-computed table of values P, $2P$, $3P$, and so on to compute the final product. To avoid side-channel attacks, implementations must avoid conditional execution. This ensures constant-time and constant-power execution. This paper presents a theoretical 42% reduction in latency for the windowed method using two tables and three cores, versus a single-threaded computation.

CCS CONCEPTS

• **Security and privacy** → **Cryptography**; • **Computing methodologies** → **Parallel computing methodologies**; • **Theory of computation** → *Scheduling algorithms*;

KEYWORDS

cryptography elliptic curve cryptography window method blockchain parallel computing scheduling algorithms

ACM Reference Format:
Tanya Bouman, Yusra Irfan, James You, and Christopher K. Anand. 2020. Parallel Windowed Method for Scalar Multiplication in Elliptic Curve Cryptography. In *EVOKE CASCON 2020: Conference of the Centre for Advanced Studies on Collaborative Research.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

Scalar multiplication is a fundamental operation in the various schemes comprising elliptic curve cryptography such as Elliptic Curve Diffie-Hellman (ECDH) key exchange and the Elliptic Curve Digital Signature Algorithm (ECDSA). The Elliptic Curve Digital Signature Algorithm is the most frequently used signing algorithm for public blockchains [15].

The computation of a signature in ECDSA requires the generation of a private key, d, and a public key, $Q = d \times G$. The private key, d, is a bounded random integer, and the public key, Q, is computed by the scalar multiplication of d and G. Both parties agree in advance on G, which is one of the points in an elliptic curve. The public key is then combined with some hash[1] of the message in order to generate the signature.

There exist numerous methods [6] for computing the scalar multiplication product. Among these are the binary method, windowed method and the Montgomery ladder. Building on work of previous authors, we show that windowed method for scalar multiplication can be parallelized, first to generate the lookup table, and then to perform the rest of the computation. In particular, we work out a method which uses 3 cores and could achieve a speedup of 42% compared to a serial windowed method.

2 BACKGROUND

2.1 Elliptic Curve Cryptography

An equation of the form

$$y^2 = x^3 + ax + b$$

defines an elliptic curve on a plane. For the purposes of cryptographical schemes, this curve is defined over a prime field.

Given two points, P and Q, an addition operation to calculate a third point, R, can be defined by taking the line through P and Q and finding the third point $-R$ on the curve where the line intersects. Then $P + Q$ is defined as $-(-R)$ or R. In the case that $P = Q$, the tangent to the curve at the point is used. Figure 1 illustrates both cases. Thus the set of points on the curve forms a group, with the point at infinity, O, serving as the identity, i.e. $P + O = P = O + P$.

There are two representations for points on the curve used in computation. The usual representation of a point in (x, y) coordinates is called *affine coordinates*. This representation is compact, but cannot represent the identity point (infinity). Since not all points (x, y) are points on the curve, a point not on the curve could be

[1] A specific hashing method is not defined in the protocol.

CASCON'20, November 10-13 2020, Toronto, Canada

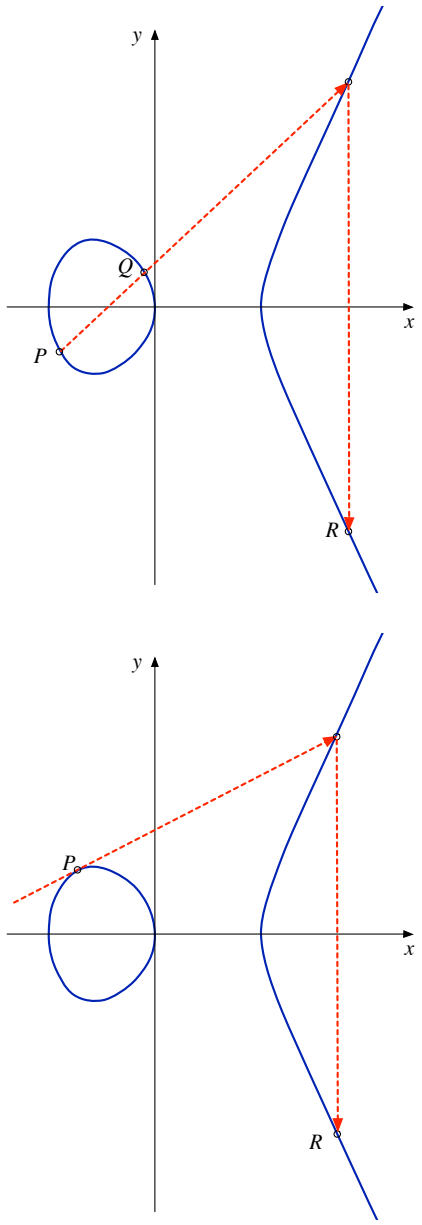


Figure 1: Addition and doubling of points on an elliptic curve [6]

used to represent the identity, but normal calculations would not work with this point.

An alternative way of representing the points on the curve is with projective coordinates, $(X : Y : Z)$. Unlike affine coordinates, projective coordinates are not unique representations of a point. *Jacobian coordinates* are one example of projective coordinates, and are useful for calculations on elliptic curves because they avoid expensive modular inverses that are necessary to compute in affine coordinates.

Since good performance is our goal, we use Jacobian coordinates in our calculations. We use existing algorithms for point addition and point doubling (see [4]). For performance estimates, we count the number of modular multiplies performed in each calculation, since other operations have negligible cost in comparison. See Table 1

Operation	Cost (modular multiplications)
2*J	8
J+A	12
J-A	12
J+J	16
J-J	16

Table 1: Cost of point operations on affine (A) and Jacobian (J) coordinates

Given these addition and doubling operations, it is possible to define a scalar multiplication, sometimes known as point multiplication, dP for an elliptic curve point P. In elliptic curve cryptography, the secret key d is the scalar in the multiplication and the product dP is the public key. Additionally, this scalar multiplication takes up a large amount of the execution time in elliptic curve cryptographic schemes [6].

The simplest method of calculating the scalar multiplication is to take the binary representation of the scalar and go through each bit, determining when to double and add points to get the answer. This is the same concept as methods used to calculate binary multiplication using addition and shifting [13], and similar methods can be used to improve performance. More sophisticated methods of doubling and adding attempt to use shorter addition chains than the one given by the binary representation of the scalar [5]. Since subtraction of an elliptic curve point is simply addition of the negative, and the negative of an elliptic curve point is formed by taking the negative of the second coordinate, subtraction costs about the same as addition and addition-subtraction chains further improve performance [11].

For any of these approaches, the value of the scalar affects the number of operations needed to compute the product, and thus the amount of computational time and energy spent. This makes it possible for an attacker to discover information about the secret key by monitoring either the time elapsed or the energy expended. In fact, within 200 signatures, an attacker may be able to reconstruct a secret key for a 256-bit curve through a side-channel, which would otherwise be computationally impossible [3].

To avoid this problem [8], the Montgomery ladder multiplication method [10] performs all of the possible doublings and additions needed for the scalar multiplication, but substitutes a no operation in place of unnecessary operations for the final product. The windowed method improves performance by calculating multiples of P in advance, i.e.,

$$1, P, 2P, \ldots, (2^w - 2)P, (2^w - 1)P,$$

where w is the width, in bits, of the window. To compute the scalar multiple dP, the integer is decomposed into "digits"

$$d = d_D d_{D-1} \cdots d_1 d_0$$

each with w bits, and containing a value between 0 and $2^w - 1$. In the first step, a table lookup finds the corresponding $d_D P$ to be

	Op.	Cost (mod. mults.)
$2 * nP$	2*P	8
$(2n-1)P + 1P$	P+1	12
$\forall i : 2 \leq i < n : (2n-i)P + iP$	P+Q	16

Table 2: Ways to calculate $(2n)P$, including costs in number of modular multiplications.

	Op.	Cost (mod. mults.)
$(2n)P + 1P$	P+1	12
$\forall i : 2 \leq i \leq n : (2n-i+1)P + iP$	P+Q	16

Table 3: Ways to calculate $(2n+1)P$, including costs in number of modular multiplications.

used as the accumulator. In subsequent steps, the accumulator is doubled w times before adding the next $d_i P$. So if the running total started as

$$\sum_{i=D}^{D} d_i 2^{(i-D)w} P,$$

after w doublings, it becomes

$$\sum_{i=D}^{D} d_i 2^{(i-D+1)w} P.$$

To this we add the value $d_{D-1}P$ from the table to obtain

$$\sum_{i=D-1}^{D} d_i 2^{(i-(D-1))w} P.$$

After $D-1$ steps, we obtain

$$\sum_{i=0}^{D} d_i 2^{iw} P = dP.$$

Other methods exist which improve upon the windowed method by skipping computation for bits which have a zero value, and using a non-adjacent form (NAF) [14] to increase the number of zero bits in the representation, but these do not perform the computation in constant time, and as a result are vulnerable to side-channel attacks [6].

Finally, there are other methods such as applying a Frobenius map [9] or other endomorphisms, but these only apply to specific types of curves.

3 SCHEDULING

This section discusses the scheduling methodology used.

3.1 Table Generation

To generate the table for a standard windowed method, we calculate all of the points between $2P$ and $2^w P - 1$, starting with $1P$ as the input value in affine coordinates. The number of operations necessary to calculate each point does not matter; only the total number of operations to calculate the entire table matters. The cost for a certain point is the number of operations used to calculate that point, given all the previously calculated points. This calculation will vary depending on the cost of the point doubling and point addition algorithm. The addition of a point in affine coordinates to a point in Jacobian coordinates costs less than the addition of two Jacobian points, and the only point that will be available in affine coordinates is P itself, so adding $1P$ is preferred over other additions.

Starting with $1P$, the only possible way to get $2P$ is by doubling $1P$, which costs 8 modular multiplies. Then, if all the points from $1P$ to $(2n-1)P$ are already calculated, there are several ways of calculating $(2n)P$, and then $(2n+1)P$. See Tables 2 and 3. For $(2n)P$, doubling nP is the cheapest option. This costs 8 modular multiplies. For $(2n+1)P$, the cheapest option is adding $1P$ to $(2n)P$, which costs 12 modular multiplies. Therefore, if all the table precomputation occurs serially, the best solution is that the even numbers be computed by point doubling and the odd numbers by adding P to the even number immediately preceding it, and this has a cost of $\frac{2^w-2}{2}(c_J + c_{JA})$, where c_J is the cost of doubling a Jacobian and c_{JA} is the cost of adding a Jacobian to an affine. These costs are 8 and 12 modular multiplies, respectively.

3.1.1 Greedy Scheduling for Infinitely Many Cores. However, when multiple cores can work in parallel to compute the table, it becomes possible that the best schedule performs different calculations than the optimized ones in the previous section. Table 4 shows a possible schedule for the initial calculations of a table. It starts out the same way as the table in serial. However, the order begins to differ at the computation of $8P$, as it could begin before the computation of $6P$ and $7P$. After a while, things get more complicated. Notice, for example, that at time 24, the computation $10P$ is not yet started, even though it could be calculated from $8P + 2P$. This is because at the next step we find out that it will actually be faster to compute $10P$ by doubling $5P$, if we just wait one more step until it is completed. So $10P$ was removed from that step and moved to a later step.

This scheduling is greedy in that it does as many computations as possible at the earliest time that they can possibly end.

Since there is no limit for simultaneous calculations, the completion time of each point can be minimized using the same methods as double-and-add calculations. The time to calculate the whole table is equivalent to the longest time required to calculate any single given point. In the case of the table of width 3, the point $7P$ is the last to finish, so that whole table requires 36 modular multiplies to calculate, compared to 60 modular multiplies if the table was calculated serially. However, the total amount of calculation increases to 64, because the calculation of $7P$ from the addition of $3P$ and $4P$ costs more than the addition of $1P$ to $6P$, and that adds two Jacobians together, rather than an affine point to a Jacobian point.

One of the relevant improvements to the double-and-add methods is that rather than limiting the chain of calculation to addition operations, we can include subtractions in the chain, and obtain a better result. The smallest point which benefits from a mixed addition subtraction chain is $15P$. Table 5 demonstrates the improvement, and the benefits continue for larger multiples of P.

3.1.2 Generate and Prune Scheduler for Finite Cores. The above algorithm schedules as many possible computations at a time. Realistically, we only have a limited number of cores available. When

Time					
0	$2P[2*P]$				
8	$3P[P+2P]$	$4P[2*2P]$			
16	$3P[P+2P]$	$5P[P+4P]$	$8P[2*4P]$		
20	$5P[P+4P]$	$6P[2*3P]$	$7P[3P+4P]$	$8P[2*4P]$	
24	$5P[P+4P]$	$6P[2*3P]$	$7P[3P+4P]$	$9P[8P+P]$	$11P[8P+3P]$ $16P[2*8P]$

Table 4: Given infinitely many cores, this is the greedy schedule.

Time	Without Subtraction		With Subtraction
0	$2P[2*P]$		$2P[2*P]$
8	$3P[P+2P]$	$4P[2*2P]$	$4P[2*2P]$
16	$3P[P+2P]$	$8P[2*4P]$	$8P[2*4P]$
20	$7P[3P+4P]$	$8P[2*4P]$	$8P[2*4P]$
24	$7P[3P+4P]$		$16P[2*8P]$
32	$7P[3P+4P]$		$15P[16P-1P]$
36	$15P[7P+8P]$		$15P[16P-1P]$
44	$15P[7P+8P]$		
52			

Table 5: Comparison of calculating $15P$ with and without subtraction.

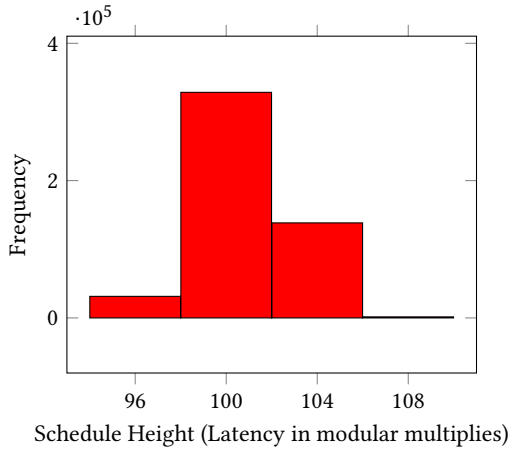


Figure 2: A histogram of schedule heights generated by depth-first search shows that most schedules have similar heights.

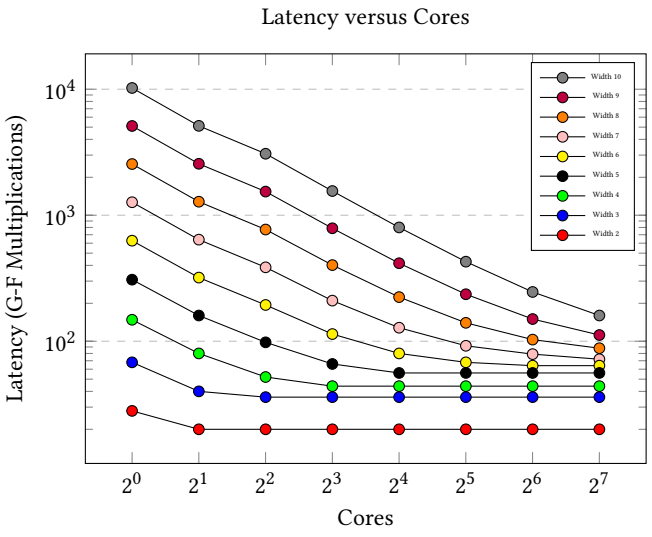


Figure 3: Cost of table computation for different numbers of cores and widths.

more computations are ready than cores available, we must decide which computations to allocate and which computations to delay for future allocation. Given all of the computations possible at a certain point, we come up with all of the possibilities for which computations to allocate, and which computations to delay until there is a core available. Since we have no obvious way to know which choice of delays will produce the shortest schedule, we try them all out. However, there might be schedules whose partial heights are longer than whatever schedules were already generated, so these are pruned away to avoid any further scheduling on them.

The results given here are an estimate based on the number of modular multiplications that an operation takes. This gives us a cost of 8 for point doubling, 12 for addition of the base point to another point and 16 for any other addition. While there are other operations involved, their cost is negligible in comparison, so for the purposes of this estimate, they are not considered.

The histogram in Figure 2 shows the distribution of schedule lengths produced from the brute force scheduling of a table with width 5 on 4 cores. Since the brute force scheduler produces many possible schedules, we select only first the 500 000 to show in the graph. While most of the schedules in the graph do not have the optimal height of 96 modular multiplications, none of them have a latency longer than 108, and the majority of the latencies are between 96 and 108.

Figure 3 show the latency of table generation for different bit widths versus the number of cores shows how much parallelism is available in the window computation. For example, at bit width 2 with a table size of 4, there is enough parallelism to reduce latency by 30% with two cores, but not enough parallelism to exploit more than two cores. At bit width 4, two cores brings a 40% reduction in latency, four cores a 48% reduction, and there are no further gains. Unsurprisingly, it is easy to exploit parallelism across a small number of cores. But for bit widths $w \leq 5$, there is no advantage to using more than 2^{w-1} cores.

Algorithm 1: Function for Brute Force Scheduling of Parallel Table Pre-Computation

Input: $width, cores, time, schedule, run, complete, delayed$
Satisfying: $time = 0, schedule = [(2, 0, (1, 1), doublePointCost)], run = [(2, doublePointCost)], complete = [1], delayed = []$
Output: A parallel schedule for a generating a table of the specified width

```
newlyCompleted ← remove completed points from run
complete ← complete + newlyCompleted
for all nc in newlyCompleted do
  for all c in complete do
    if nc == c then
      delayed += (newComplete+complete, (complete,newlyCompleted), doublePointCost)
    else if nc == 1 or c == 1 then
      delayed += (newComplete+complete, (complete,newlyCompleted), addJAPointCost)
                 + (newComplete-complete, (-complete,newlyCompleted), addJAPointCost)
    else
      delayed += (newComplete+complete, (complete,newlyCompleted), addJJPointCost)
                 + (newComplete-complete, (-complete,newlyCompleted),addJJPointCost)
    end if
  end for
end for
newRuns, newDelayeds
    ← take the updated delayed and find all the possibilities of what can currently be run and what needs to be delayed
for all r in run do
  r ← (r.0, r.1 − 1)
end for
for all rs, ds in newRuns, newDelayeds do
  newSchedule ← schedule
  for all r in rs do
    newSchedule ← newSchedule + r
  end for
  schedules ← scheduleFunction(width, cores, time + 1, newSchedule, rs, complete, ds)
end for
return minimum of schedules
```

Figure 4 shows the latency of table generation compared to the width. Plotting the latency of window computation as a function of the window width in bits shows that the computation scales exponentially with a single core, but has sub-exponential scaling with larger numbers of cores. For a single core, the cost depends on the width as $10(2^w - 2)$. For two cores, it is $5(2^w) - 4$, when $w > 2$.

3.2 Main Computation

Given the above method of pre-computing the table in parallel, we want the main part of the computation to also be parallelized. When looking at the method which requires the least amount of computation, it is not immediately obvious that any parallelism is possible:

$$\begin{aligned} dP &= d_0P + \sum_{i=1}^{D-1} 2^{w_s+(i-1)w} d_iP \\ &= d_0P + 2^{w_s}(\ldots(d_{D-2}P + 2^w(d_{D-1}P))\ldots) \end{aligned} \tag{1}$$

where D is $\lceil \frac{256}{w} \rceil$ and w_s (the width of the short digit at the end) is $256 - (D-1)w$. In the equation above, each operation must occur in order, leaving no room for parallelism. The cost to perform all of these operations in serial is

$$16(D-1) + 8(256 - w).$$

Every digit needs to be added to the total, so there are $\lceil \frac{256}{w} \rceil - 1$ or $D - 1$ additions. Then the repeated doubling in 2^w happens for all but one of the digits, for a total of $256 - w$ doubles.

In order to allow parallelism, Equation 1 must be modified to a different form that still calculates the same dP. These modifications may do redundant or duplicate operations in order to improve parallel performance. We present two methods of introducing parallelism, and combine them to produce the final speedup.

3.2.1 Negative table. Since we can very cheaply negate a point by negating its y-coordinate, calculating a table of $0 \ldots 2^wP$ costs almost exactly as much as calculating a table of $-2^wP \ldots 2^wP$. The advantage here is that the secret key d can be divided into larger digits, reducing the overall number of digits. This means that the width used in calculating the table is 1 less than the width of the digits, so we use w_d and w_t to distinguish them, where $w_d = w_t + 1$. For any digit d, we first subtract 2^{w_t}, then look it up in the table, finally adding $2^{w_t}P$. The cost of a single addition is 16 modular multiplications with the cost of the subtraction and table look-up considered negligible.

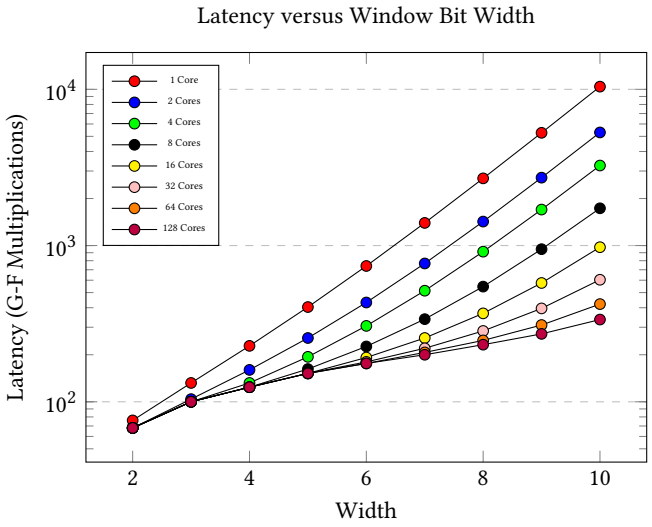


Figure 4: Cost of table computation for different widths and numbers of cores.

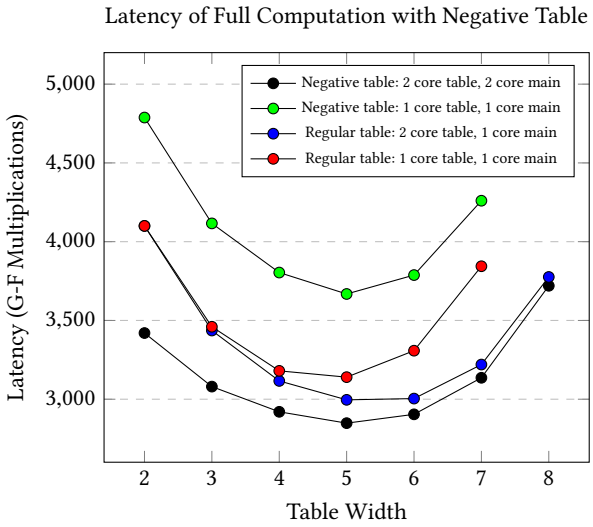


Figure 5: By expanding the table to include the negatives, we attain a slight performance improvement.

$$dP = (d_0 - 2^{w_t})P + 2^{w_t}P + \sum_{i=1}^{D-1} 2^{w_s+(i-1)w_d}((d_i - 2^{w_t})P + 2^{w_t}P) \quad (2)$$

In serial, the cost of this calculation is

$$16(2\lceil\frac{256}{w_d}\rceil - 1) + 8(256 - w_d),$$

and it needs to include another 8 modular multiplications, because the table needs to include the calculation of $2^w P$. The extra cost compared to the shortest serial version is

$$16(2\lceil\frac{256}{w_d}\rceil - 1) + 8(256 - w_d) + 8 - 16(\lceil\frac{256}{w_t}\rceil - 1) - 8(256 - w_t)$$

which simplifies to

$$16(2\lceil\frac{256}{w_d}\rceil - \lceil\frac{256}{w_t}\rceil).$$

Thus, this scheme requires more computation. However, the extra addition of $2^{w_t}P$ can be put in parallel with the rest of the computation, and the timing can be improved over the initial serial version. The amount of computation that can be put onto the second core is

$$16(\lceil\frac{256}{w_d}\rceil - 1).$$

So the total amount of time saved is

$$16(\lceil\frac{256}{w_d}\rceil - 1) - 16(2\lceil\frac{256}{w_d}\rceil - \lceil\frac{256}{w_t}\rceil),$$

simplified as

$$16(\lceil\frac{256}{w_t}\rceil - 1 - \lceil\frac{256}{w_d}\rceil).$$

Figure 5 compares the costs of a serial main calculation (Regular table: 2 core table, 1 core main; Regular table: 1 core table, 1 core main), with the extra cost of using the negative table (Negative table: 1 core table, 1 core main), and finally the parallel version with negative tables (Negative table: 2 core table, 2 core main).

For the best width, 5, the cost reduces from 3140 to 2848, an improvement of 10%. The first core is busy for the entire time of the main computation, while the second core is only busy part of the time. For width 5, this is 2688 and 672 modular multiplies, respectively. The balance is much better for the table, where the cost of 160 represents an almost even split with a gap of 8 at the beginning, and 4 at the end, so that the second core has 148 modular multiplies. In total, 29% of the runtime has 2 cores running simultaneously, while the other 71% is on one core only.

0			p_s				256
d_0	d_1	...	d_{D_1-1}	d_{D_1}	d_{D_1+1}	...	$d_{D_1+D_2-1}$
w_{s_1}	w_1	...	w_1	w_{s_2}	w_2	...	w_2

Table 6: Division of the scalar into digits

3.2.2 Multiple tables. To introduce more parallelism, we split the summation formula into 2 pieces. We split the secret key at some point p_s. There are p_s bits of the scalar on the lower side, and $256 - p_s$ on the upper side. The bits of the scalar are laid out into digits of sizes w_1,w_2, etc. as shown in Table 6.

$$\begin{aligned} dP = (d_0 P + \sum_{i=1}^{D_1-1} 2^{w_{s_1}+(i-1)w_1}(d_i P)) \\ + (d_{D_1} 2^{p_s} P + \sum_{i=1}^{D_2-1} 2^{w_{s_2}+(i-1)w_2}(d_{D_1+i} 2^{p_s} P)) \end{aligned} \quad (3)$$

where D_1 is $\lceil\frac{p_s}{w_1}\rceil$, D_2 is $\lceil\frac{256-p_s}{w_2}\rceil$, w_{s_1} (the width of the short digit at the end) is $p_s-(D_1-1)w_1$. w_{s_2} (the width of the short digit at the end)

cycle	Core 1	Core 2	Core 3
0	$2^{p_S}P$	Make Table 1	
152		idle	Use Table 1
1488	Make Table 2		
1584			
2192	Use Table 2		
2208	Addition	idle	idle
2224	Done		

Table 7: Work assignment in blocks to three cores.

is $(256 - p_s) - (D_2 - 1)w_2$. Rather than calculating $d_i 2^{p_s} P$ at every digit in the second half, we use a separate table to look up each digit. This table's values range from 0 to $(2^{w_2} - 1)(2^{p_s})P$ (or $-2^{w_2} 2^{p_s} P$ to $2^{w_2} 2^{p_s} P$ if it includes negatives.) The extra computation here is the calculation of $2^{p_s} P$ (which is p_s doubles or $8p_s$), plus the cost of the extra table calculation, which is $10(2^{w_2} - 2)$.

The total amount of computation done is

$$10(2^{w_1} - 2 + 2^{w_2} - 2) + 16(D_1 + D_2 - 1) + 8(256 + p_s - w_1 - w_2).$$

Put in parallel, the latency is

$$\begin{aligned} 16 + \max(&10(2^{w_1} - 2) + 16(D_1 - 1) + 8(p_s - w_1) \\ &, 10(2^{w_2} - 2) + 16(D_2 - 1) + 8(256 - w_2)). \end{aligned} \tag{4}$$

Figure 6 compares the costs of choosing different widths for the two tables and split points. The best performing schedule is 2412 modular multiplies, when both of the widths are 4 and the split point is 192. The total amount of computation is 4808 modular multiplications, and is split almost evenly across the two cores, with only the final point addition not happening in parallel.

Putting the method with two tables together with the negative table method uses a total of 4 cores. However, the 2 cores which add the extra $2^{w_d} P$ at each digit are not that busy, because there is only one addition that needs to happen in the same time that the an addition and w_d doubles occur. So these 2 cores can be merged together for a computation that runs on 3 cores. The latency of that method is

$$\begin{aligned} 16 + \max(&10(2^{w_{2_t}}) - 12 + 16(D_2 - 1) + 8(256 - w_{2_d}) \\ &, 5(2^{w_{1_t}}) + 16(D_1 - 1) + 8(p_s - w_{1_d})) \end{aligned} \tag{5}$$

where $w_{1_t} > 1$. The best case occurs with parameters $w_{1_d} = 6, w_{2_d} = 4, p_s = 204$, with 2308 modular multiplies.

Because of the amount of time spent on calculating $2^{p_s w_d} P$, there is also room on the 3 cores to split one of the main computations again, further improving the performance.

$$(\ldots d_0 + 2^{w_d}(d_1 P + 2^{w_d}(\ldots))) + 2^{p_s w_d}(\ldots d_{D-1} P + 2^{w_d}(d_D P)\ldots)$$

Table 7 shows how the calculations are laid out to achieve a performance of 2224.

3.3 Split Without Table

Similar to the multiple table method, we split the summation formula into 2 parallel pieces. We again split it at some point p_s, which is the split point. However, this does not require a second table, because 2^{p_s} happens once at the end of the second core portion of the calculation. We use the same digit layout as shown in Table 6, except that since there is only one table, there is only one width, $w_1 = w_2$.

$$\begin{aligned} dP = d_0 P &+ \sum_{i=1}^{D_1 - 1} 2^{w_{s_1} + (i-1)w}(d_i P) \\ &+ 2^{p_s}(d_{D_1} P + \sum_{i=1}^{D_2 - 1} 2^{w_{s_2} + (i-1)w}(d_{D_1 + i} P)) \end{aligned} \tag{6}$$

where D_1 is $\lceil \frac{p_s}{w} \rceil$, D_2 is $\lceil \frac{256 - p_s}{w} \rceil$, w_{s_1} (the width of the short digit at the end) is $p_s - (D_1 - 1)w$. w_{s_2} (the width of the short digit at the end) is $(256 - p_s) - (D_2 - 1)w$. Thus the total amount of calculation done is

$$10(2^w - 2) + 16(D_1 + D_2 - 1) + 8(256 + p_s - w_1 - w_2).$$

and the latency is

$$5(2^w) - 4 + \max(16(D_1 - 1) + 8(p_s - w_1), 16(D_2 - 1) + 8(256 - w_2)) + 16.$$

The table calculation happens on 2 cores, reducing its latency to $5(2^w) - 4$, for $w > 2$, and the two pieces of the main computation happen in parallel, with the latency being whichever piece takes longer.

Figure 7 compares the costs of choosing different widths and split points. From the graph, the best performance is 2348 and happens with a width of 4, and a split point 192. The total amount of computation is 4668 modular multiplications. The load balancing between the two cores is almost even, with a gap of 16 modular multiplies for the final sum at the end, and 12 modular multiplies in the table. This means that for 99% of the computation time, both cores are active. Compared to the best serial time, the performance improvement is 34%.

4 PERFORMANCE ESTIMATES AND RECOMMENDATIONS

When the windowed method is used without parallelization, the normal recommendation is to use a width of size 4, due to the trade-off between the amount of pre-computation and the main calculation itself. However, if the pre-computation portion can be performed in parallel, the latency is significantly reduced, meaning that larger size widths can be considered. When we calculate the total latency including both the window pre-computation and calculation using the window, we find a more modest expected speedup of 20% when comparing the best window size for single-core execution with the best window size for 32-core execution. This is because the window-using computation is still serial. It is interesting to note that even without parallelizing that part of the computation, the best window size increases with the number of cores. For a key (scalar) size of 256 bits, Figure 8 shows the total cost for the pre-computation and the main computation together, illustrating the trade-off that occurs for various different widths. Thus for 32 cores, it would be better to use a table of bit width 7 or 8, rather than 4.

One danger with a larger table is if the table is larger than the cache and some of the loads from the table miss the cache, it would be possible for an attacker to get side-channel information from those cache misses. For example, while the Montgomery ladder

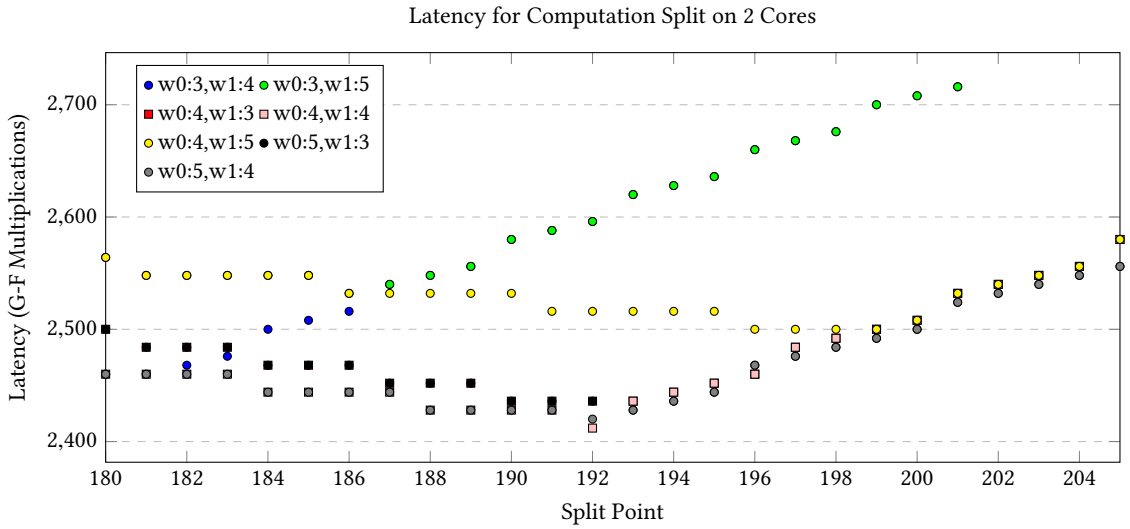


Figure 6: By splitting the main computation across two cores, the performance improves, depending on the split point and table widths.

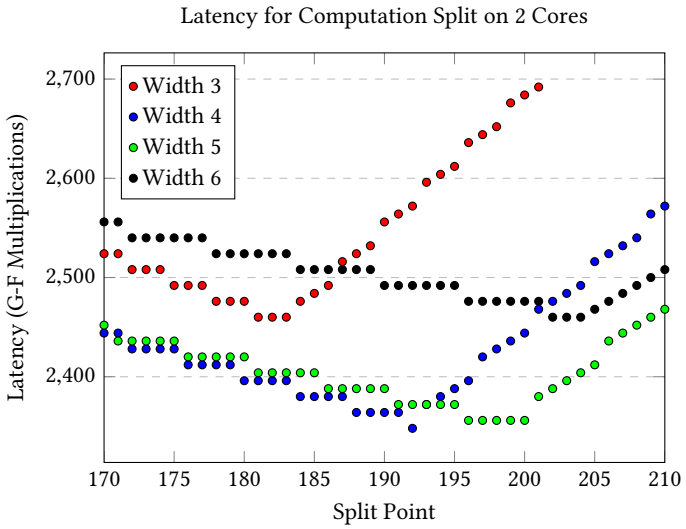


Figure 7: By splitting the main computation across two cores, the performance improves, depending on the split point and table width.

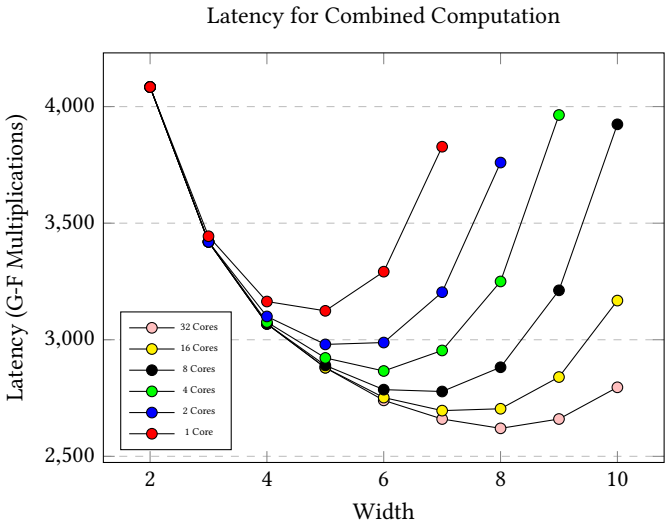


Figure 8: Cost of table computation for different widths and number of cores, and main computation on one core.

performs the same amount of computation regardless of the input, it is still possible on certain processors to perform a side-channel attack using information from the cache, since the memory lookups are not the same [16].

Once we include parallelization of the rest of the computation, the optimal width of the table reduces back down to 4 or 5, as the cost of computing the table starts to overtake the advantages which are now possible for the rest of the computation. There are a number of different parameters which change the performance, including the widths of the two tables, and the split points for breaking apart which terms go on separate cores.

Figure 9 shows how the performance is affected by the choice of split point 2. Split point 2, p_{s_2}, describes the number of bits calculated on one core while the rest are on another core. To find the best one, we try out different values.

For 3 cores, this full solution takes the time of 2224 modular multiplications. The total amount of computation done is 5220 modular multiplications, which is split across the cores as 2216, 2184, and 820, respectively. Comparing this to the fastest serial

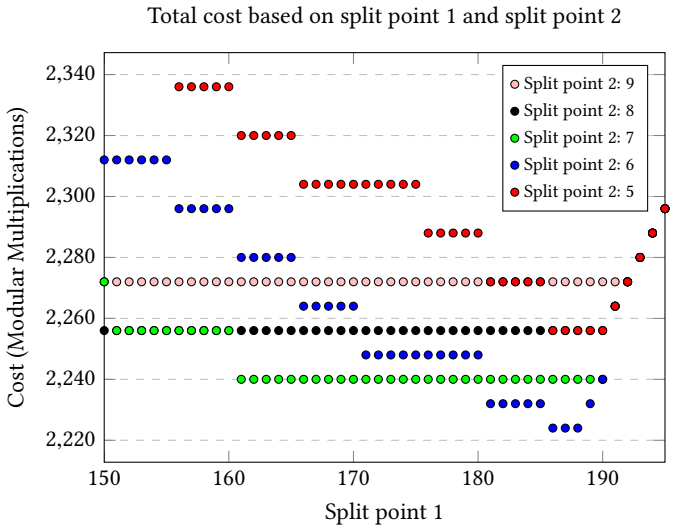


Figure 9: Finding the optimal split point 1 and 2 by trying different values.

computation, of 3164 modular multiplications, it has a speedup of 42%, by doing 1.6x more computation.

4.1 Simulated Implementation in Haskell

This project was developed in Haskell to take advantage of an interpreter for elliptic curve operations and for ease of implementing heuristic scheduling with greedy phases. This made it easy to test the table computations and the applications of the tables in the multi-window computation. This allowed us to test for correctness of the underlying algorithm before attempting to schedule it.

4.2 Parallel Implementation in Haskell

Given this model of the computation, we proceed to implement in parallel in Haskell, to do a preliminary test of the performance improvement. The improvement is in comparison to the standard, single core windowed method, also implemented in Haskell. Haskell provides `MVar` as the basic communication method among threads.

Given an implementation which performs a scalar multiplication on 3 threads, we check how the program actually uses the resources. ThreadScope is a program which analyzes parallel Haskell programs for their performance [1]. We used ThreadScope to help debug synchronization problems, and verify that computation was being distributed across cores. Unfortunately, it also shows a large amount of overhead on core 1, which is already scheduled to be the busiest core.

5 DISCUSSION

We have shown that there are many opportunities to reduce the latency of scalar multiplication in elliptic curves. We have used the number of multiplications in the Galois field as the primary unit of computation, ignoring other operations and overhead. Given that multiplication takes hundreds of cycles, including other computation would not change our recommendations. Overhead can significantly degrade performance, but in our case, we know the sequence of tasks required, which makes overhead easier to avoid.

There are a range of platforms which need to perform cryptographic operations, from high-throughput servers to desktops, laptops and mobile devices. On servers, we expect to have large numbers of cores and sophisticated thread scheduling in the operating system. In this case, using a thread pool and message queues with a fixed communication pattern would allow computation to be distributed across cores efficiently. Since the computation is not dependent on the data, we are reasonably confident that it would not be subject to a side-channel attack. If processor load varies significantly, the width of the algorithm could be tuned to the environment, from one computation to the next, trading latency off for efficiency as necessary. There is no need to dynamically schedule the computation. It would be relatively straightforward to implement a parallelization strategy on top of existing technology such as OpenMP, with either implicit or explicit synchronization, or Go using coroutines with communication via channels. On the smallest devices (which still have multiple threads), this support may be lacking, but the same structure as with coroutines could be implemented by using atomic memory accesses to communicate the availability of required inputs.

One side effect of using branch-free implementations to avoid side-channel attacks is that computation time is completely deterministic. Because multiplication in the Galois field dwarfs other computation at the same level of abstraction, the computation is also conveniently chunked into multiples of this time, which makes it easier to line up the computation to make spin locks efficient.

6 RELATED WORK

Using addition chains or addition-subtraction chains has applications in other contexts. Another example in cryptography is using addition chains to calculate powers for RSA. In the case of calculating powers, it is not helpful to include subtraction in the chains, since division is significantly more expensive than multiplication [5, 11]. Like many other methods, using chains of additions and subtractions in an efficient manner dependent on the data leads to non-constant time calculation, although there is a proposal by Oswald to get around this with randomization [12].

Izu and Takagi parallelize scalar multiplication by dividing the binary decomposition into high and low halves and doing the Montgomery ladder in parallel for both parts [7]. This modestly reduces the amount of computation exposes parallelism, in contrast to the window method which is designed to reduce computation. Basu also developed a parallel algorithm, with impressive theoretical speedup, but it is subject to side-channel attacks. Similar to our results, Basu notes that parallelism makes it feasible to use much larger pre-computed tables, up to a bit width of 10 [2].

7 CONCLUSION

Initially, we showed that it is possible to parallelize the table precomputation portion of the windowed method for elliptic curve scalar multiplication, and that it significantly improves the latency. Next, we considered block schedules, and found further expected efficiencies by scheduling computation using Table 1 in parallel

with the calculation and use of Table 2. In future work, we hope to more optimally minimize idle blocks in our current schedules.

In addition to estimating performance, we have implemented a synchronization scheme using MVars in Haskell. This allows us to use the existing interpreter for elliptic curve operations. Results from the interpreter show that the schedule produces the correct answer. However, it does not give reliable performance information, probably due to overhead. This gives us confidence to implement the computation in Go using a similar mechanism.

ACKNOWLEDGMENTS

The authors would like to thank NSERC and the IBM Centre for Advanced Studies for financial support.

REFERENCES

[1] [n. d.]. ThreadScope. https://wiki.haskell.org/ThreadScope. ([n. d.]). (Accessed on 05/04/2020).

[2] Saikat Basu. 2012. A new parallel window-based implementation of the elliptic curve point multiplication in multi-core architectures. *Group* 16, 4a3 (2012), 27b2.

[3] Naomi Benger, Joop van de Pol, Nigel P. Smart, and Yuval Yarom. 2014. "Ooh Aah... Just a Little Bit" : A Small Amount of Side Channel Can Go a Long Way. In *Cryptographic Hardware and Embedded Systems – CHES 2014*, Lejla Batina and Matthew Robshaw (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 75–92.

[4] Daniel J. Bernstein and Tanja Lange. [n. d.]. Explicit Formulas Database - Jacobian coordinates with a4=-3 for short Weierstrass curves. ([n. d.]). http://hyperelliptic.org/EFD/g1p/auto-shortw-jacobian-3.html Accessed 27 May 2020.

[5] Jurjen Bos and Matthijs Coster. 1990. Addition Chain Heuristics. In *Advances in Cryptology — CRYPTO' 89 Proceedings*, Gilles Brassard (Ed.). Springer New York, New York, NY, 400–407.

[6] Alfred Menezes Hankerson, Darrel and SA (Scott Alexander) Vanstone. 2004. *Guide to elliptic curve cryptography*. New York: Springer.

[7] Tetsuya Izu and Tsuyoshi Takagi. 2002. A Fast Parallel Elliptic Curve Multiplication Resistant against Side Channel Attacks. In *Public Key Cryptography*, David Naccache and Pascal Paillier (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 280–296.

[8] Paul C. Kocher. 1996. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Advances in Cryptology — CRYPTO '96*, Neal Koblitz (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 104–113.

[9] Willi Meier and Othmar Staffelbach. 1993. Efficient Multiplication on Certain Nonsupersingular Elliptic Curves. In *Advances in Cryptology — CRYPTO' 92*, Ernest F. Brickell (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 333–344.

[10] Peter L Montgomery. 1987. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of computation* 48, 177 (1987), 243–264.

[11] François Morain and Jorge Olivos. 1990. Speeding up the computations on an elliptic curve using addition-subtraction chains. *RAIRO-Theoretical Informatics and Applications* 24, 6 (1990), 531–543.

[12] Elisabeth Oswald and Manfred Aigner. 2001. Randomized Addition-Subtraction Chains as a Countermeasure against Power Attacks. In *Cryptographic Hardware and Embedded Systems — CHES 2001*, Çetin K. Koç, David Naccache, and Christof Paar (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 39–50.

[13] George W Reitwiesner. 1957. *SUMMARY DISCUSSION ON PERFORMING BINARY MULTIPLICATION WITH THE FEWEST POSSIBLE ADDITONS*. Technical Report. ARMY BALLISTIC RESEARCH LAB ABERDEEN PROVING GROUND MD.

[14] George W. Reitwiesner. 1960. Binary Arithmetic. In *Advances in Computers*, Franz L. Alt (Ed.). Vol. 1. Elsevier, 231 – 308. https://doi.org/10.1016/S0065-2458(08)60610-5

[15] Licheng Wang, Xiaoying Shen, Jing Li, Jun Shao, and Yixian Yang. 2019. Cryptographic primitives in blockchains. *Journal of Network and Computer Applications* 127 (2019), 43 – 58. https://doi.org/10.1016/j.jnca.2018.11.003

[16] Yuval Yarom and Naomi Benger. 2014. Recovering OpenSSL ECDSA Nonces Using the FLUSH+ RELOAD Cache Side-channel Attack. *IACR Cryptology ePrint Archive* 2014 (2014), 140.

Hybrid Quantum-Classical Problem Solving in the NISQ Era

Prashanti Priya Angara
pangara@uvic.ca
University of Victoria
Victoria, British Columbia, Canada

Ulrike Stege
ustege@uvic.ca
University of Victoria
Victoria, British Columbia, Canada

Hausi A. Müller
hausi@uvic.ca
University of Victoria
Victoria, British Columbia, Canada

Mehdi Bozzo-Rey
mbozzore@ca.ibm.com
IBM Canada
Markham, Ontario, Canada

ABSTRACT

Quantum computing has evolved from a field of scientific research to a quantum technology industry. Much progress is still needed to solve real-world problems with quantum technology and achieve quantum advantage. Industries, governments, and universities are experimenting with advanced quantum computing technologies to become quantum ready. One way forward is to combine quantum and classical approaches to form hybrid models, algorithms, and architectures to overcome the limitations of NISQ systems for near-term quantum computations. Many algorithm design, data management, and software engineering challenges have to be addressed for practical hybrid development including problem decomposition, variational circuit design, tool integration, and debugging. This paper presents algorithmic patterns and software infrastructures appearing in the literature for practical hybrid quantum problem solving and software development for selected application domains. Hybrid approaches provide significant opportunities for HPC centers and application developers to tackle hard problems that have been considered intractable using merely classical approaches. The most promising hybrid algorithms and architectures provide an excellent avenue for developers to scale quantum applications gradually and for educators to train the workforce incrementally.

CCS CONCEPTS

• **Computer systems organization → Quantum computing**.

KEYWORDS

Quantum computing, hybrid quantum-classical toolkits, variational algorithms and circuits, QPU, HPC

ACM Reference Format:
Prashanti Priya Angara, Ulrike Stege, Hausi A. Müller, and Mehdi Bozzo-Rey. 2020. Hybrid Quantum-Classical Problem Solving in the NISQ Era . In *Proceedings of ACM Conference (CASCON 2020).* IBM Corp., Riverton, NJ, USA.

1 INTRODUCTION

Many scientists and engineers consider quantum computing as one of the most interesting and challenging topics with enormous potential in many different application areas. The field of quantum computing is not new. Physicists argue it started in the early 1920s when the term quantum mechanics was coined. In 1981, Richard Feynman encouraged the physics community to build a quantum computer with his famous quote: "Nature isn't classical, and … if you want to make a simulation of nature, you'd better make it quantum mechanical, and by golly, it's a wonderful problem, because it doesn't look so easy." After delivering his famous lectures on *Simulating Physics with Computers* [27], the field began in earnest with quantum information theory, computing models, and algorithms [54]—culminating in seminal results, including Shor's 1994 factoring algorithm with exponential speedup [68], and Grover's search (1996) with quadratic speedup [30] over classical algorithms.

Over the last 15 years, companies such as IBM, D-Wave, Microsoft, Google, Honeywell, Fujitsu, Rigetti, Xanadu, and many startups have engineered real quantum computers and developed *quantum development kits (QDKs),* such as IBM Qiskit & Aqua, D-Wave Leap & Ocean, Microsoft Q# & QDK, Google Cirq & OpenFermion, Rigetti Forest & Quil, Xanadu's Strawberry Fields & PennyLane, as well as application platforms, such as 1QBit's OpenQEMIST, to program these innovative systems effectively. Thus, in recent years, the field of quantum computing has transitioned into a technology industry [40].

We are now in the noisy intermediate-scale quantum (NISQ) era where quantum computers have 50-200 qubits and their noise places serious limitations on their capabilities [59]. Researchers are investigating innovative ways to solve valuable problems using available NISQ systems and to achieve quantum advantage by demonstrating a significant performance advantage over today's classical computers [10, 61].

Over the last decade, scientists and engineers have developed effective hybrid algorithms and architectures by integrating classical and quantum processing units (QPUs) [14, 51]. Hybrid approaches can be applied at problem-solving and design time (e.g., problem decomposition and algorithm design techniques) as well as implementation and run-time (e.g., parameterizations and variational circuits). Researchers have begun realizing Feynman's dream by decomposing nature problems—simulating molecules using variational algorithms for drug design might be the early killer application of quantum computing [52, 58]. While hybrid algorithms and

platforms may just be the best first step, it is reasonable to assume that quantum applications will always be hybrid [42].

Large industry and government investments are pushing for breakthroughs in qubit counts and fidelities, with quantum advantage being a much-sought-after milestone [49]. Today, quantum computers and simulators are readily accessible and programmable over the internet in several *quantum clouds* [14, 40] and can now be explored by everybody. As a result, the demand for quantum computing education, training, and workforce development is increasing continuously [14].

The combination of these efforts has created significant opportunities for HPC centers and application developers alike, to tackle hard problems that are considered intractable using classical approaches. Moreover, hybrid approaches provide an excellent avenue for developers to grow and scale quantum applications gradually and for educators to train the quantum workforce incrementally.

Three problem types are at the core of many quantum applications: *simulation, machine learning,* and *optimization* [71]. Hybrid quantum-classical algorithms and full-stack quantum platforms are being developed for all three problem types. This paper characterizes the current *hybrid quantum toolkit* available to quantum scientists and engineers, including core building blocks, selected algorithm design patterns, and platform considerations for hybrid quantum-classical problem-solving in the NISQ era. Lessons we can learn in the short-term experimenting with hybrid quantum-classical computing will be eminently useful in the long-term because we expect that quantum computing will scale and remain hybrid.

Section 2 presents core building blocks for hybrid computing and describes effective techniques of the hybrid toolkit prominently featured in the literature. Section 3 outlines hybrid algorithmic approaches to optimization and decision problems. Section 4 highlights algorithms and architectural patterns in quantum machine learning problems. Section 5 discusses the notion of hybrid variational patterns, which are at the core of hybrid algorithms for quantum optimization, machine learning, and simulation problems. Finally, Section 6 concludes with a call for quantum software engineering skills.

2 THE HYBRID QUANTUM TOOLKIT

In Chapter 3 of the famous *Consensus Study Report* on Quantum Computing [32], the term *basic quantum building block* refers to quantum algorithms with exponentially better performance than classical algorithms. Here we expand the term to *hybrid quantum building blocks* (or *hybrid toolkit* for short) to include quantum algorithms that solve problems that lend themselves as sub-problems frequently used in a larger context, and provide an asymptotic speedup to its classical counterparts in general, as opposed to exponential speed-up only. A number of these hybrid quantum building blocks—core library routines—are useful for solving many (quantum) problems.

Building blocks are elements of the hybrid toolkit that also includes algorithmic and architectural design patterns. Examples of such powerful building blocks are *quantum phase estimation (QPE), Grover's search, variational quantum eigensolver (VQE), variational quantum factoring (VQF)* [4], and linear equation solvers [33, 46]. Famous techniques in the hybrid quantum toolkit include *quantum approximate optimization algorithms (QAOA)* [25], hybrid variants of classical algorithm-design techniques or design patterns (e.g., [3, 23]), *hybrid variational algorithms* (Section 5), and the *quantum support vector machine (QSVM)* [60]. Over the past decade, QPE, VQE, and QAOA have emerged as the most significant near-term building blocks and techniques for NISQ systems. [70].

Quantum Phase Estimation. QPE is the workhorse behind many quantum algorithms [55]. QPE, which is enabled by the *quantum Fourier transform (QFT)* [13], approximates the eigenvalue associated with a given eigenvector of a unitary operator. The long coherence times needed for QPE can be mitigated through a hybrid approach. The QPU is used as a state preparation and measurement routine while the CPU processes the measurement updates from the quantum circuit. QPE has numerous applications in the quantum problem-solving realm. For example, it is a key component of the order-finding and factoring algorithms. Shor's algorithm [68] is expected to be a serious threat to popular public-key systems that we rely on to secure information [11]. The VQF algorithm [4] is a promising near-term alternative to Shor's algorithm.

Grover's Search and Variants. One of the most famous building blocks is Grover's search [30]. Because searching is such a fundamental operation, the quantum search algorithm by Grover is used as a subroutine in many different contexts. Given n unstructured elements, it finds a specific item in $O(\sqrt{n})$ time as opposed to the classical $\Theta(n)$ algorithm, a quadratic speedup. The quantum algorithm is based on a black-box transformation and phase kick-back—both are concepts that are used in other foundational quantum algorithms including the Deutsch-Jozsa algorithm [19] and QPE. Important variants of Grover's search are the quantum algorithm for finding the minimum in an unstructured set [24], quantum partial search [31], and quantum random walk algorithms [67].

Hybrid Variational Quantum Eigensolver. VQE is a hybrid algorithm that computes the lowest eigenvalue of a Hermitian matrix or a Hamiltonian H [39, 58]. VQE algorithms are based on the variational method of quantum mechanics, which states that for a given H its expectation value must be greater than or equal to the lowest possible eigenvalue. VQE approximates solutions by decomposing an exponential-sized optimization problem and executing a polynomial number of quantum sub-problems. Hamiltonian H can be expressed as a set of terms that are realized as separate quantum circuits. The expectation values of its parts are then summed up classically to compute the expectation value of H. The initial states for these circuits are selected from a set of states based on an ansatz and are generated by a parameterized circuit. The parameters for the state preparation circuit are computed in a classical optimization loop that minimizes the expectation value of H. Compared to QPE circuits, the depths of the VQE circuits are considerably smaller, which is a big advantage in the NISQ era. Progress in computational quantum chemistry and optimization has been driven largely by VQE [39]. Many VQE applications are in molecule simulation [39, 50], where the lowest possible eigenvalue represents the ground state energy of the molecule. VQE also applies to hybrid machine learning and combinatorial optimization.

Hybrid Harrow Hassidim Lloyd. Another useful quantum algorithm is the *Harrow Hassidim Lloyd* algorithm (*HHL*) that solves a system of n linear equations in $O(\log n)$ steps [33]. Solving a system of linear equations is central to many optimization problems. Aaronson outlines ways of using the HHL algorithm in machine learning [2].

Lee et al. [46] propose a hybrid variational variant of the HHL algorithm to solve linear equations on NISQ systems by optimizing the quantum circuit. While the quantum circuit for the HHL algorithm consists of a quantum phase estimation part and an ancilla quantum encoding part, followed by an inverse quantum phase estimation, the proposed hybrid algorithm first repeatedly performs QPE to obtain k-bit classical information of eigenvalues. Then, it follows up with a classical routine that analyzes measurement outcomes from the first step. Based on the analyzed data; it then determines which simpler circuit implementation of the original ancilla quantum encoding part is applicable. Finally, the algorithm performs the original HHL algorithm with the reduced ancilla quantum encoding part instead of the original one.

Infrastructure for Hybrid Approaches. Basic linear algebra subprograms (BLAS) comprises a set of tools describing subroutines for basic linear algebra such as matrix multiplication, dot product, and cross-product. Biamonte et al. [8] refer to quantum algorithms for linear algebra tasks as *qBLAS* or quantum BLAS. These subroutines are provided by quantum software libraries, such as *IBM Qiskit & Aqua* [36] or *Microsoft Q# & QDK*, which include implementations of fundamental quantum algorithms such as QPE, Grover, VQE, and HHL, and support for techniques such as *QAOA* and *QSVM*. These are eminently useful in developing practical hybrid solutions. Xanadu supports photonic computing through *Strawberry Fields* and quantum machine learning through *PennyLane* [7]. Rigetti offers *quantum cloud services (QCS)* including parametric compilation, active qubit reset for improving performance, and co-located CPUs and QPUs, which are useful for optimizing hybrid and variational algorithms [40]. D-Wave offers access to hybrid quantum annealing through the software frameworks Ocean SDK & Leap allowing developers to work on quantum and classical systems in parallel. D-Wave also provides quantum variational autoencoders (QVAEs) to accelerate machine learning tasks [42]. Honeywell and IonQ provide access to their trapped-ion computers through Microsoft Azure Quantum and Amazon Braket.

Hybrid quantum-classical computations are increasingly supported by hybrid architectures. In many ways, the development of hybrid architectures with dedicated *quantum processing units* (QPUs) is similar to the evolution of graphics hardware accelerators such as *graphical processing units* (GPUs), which are tightly integrated into HPC platforms to tackle computationally data-intensive problems. The race is on to develop hardware-agnostic software models at different levels of abstraction to program hybrid solutions effectively. One of the most prominent frameworks is eXtreme-scale ACCelerator programming (XACC) [51], an open-source, hybrid programming model developed at Oak Ridge National Laboratories (ORNL), designed to enhance classical software workflows with near-term QPUs.

3 DESIGNING HYBRID ALGORITHMS

Since the 1980s (e.g. [17]) many computational optimization and decision problems have been studied with the goal to design quantum algorithms that exhibit a significant speedup over their best classical solutions. Today, in addition to quantum algorithms for computational problems, which produce exact solutions with high probability but are not practical to run on today's NISQ systems for even moderate input sizes, there are hybrid approaches that combine quantum and classical computing. The approaches to optimization or decision problems range from heuristic hybrid algorithms that quantum annealing and are designed for special-purpose quantum computers, to the design of specific hybrid algorithms that produce solutions that are, with high probability, exact solutions.

Approximate & heuristic hybrid techniques. Quantum annealing is an optimization technique that was formulated in 1998 by Kadowaki and Nishimori [38]. It is a heuristic used for finding a global minimum of a given objective function using quantum fluctuation based computation. Adiabatic quantum computing (based on the adiabatic theorem [9]) is a form of quantum computing that relies on a particular kind of quantum annealing. D-Wave announced the first quantum annealer in 2011 [37]—a special-purpose quantum computer that, through quantum annealing, can only be used for problems defined as energy minimization. Santoro and Tosatti [62] outline applications of quantum annealing in the area of hard optimization problems, including the traveling salesperson (TSP) and the Boolean satisfiability problem. Tran et al. [72] propose a hybrid heuristic framework, which uses quantum annealing as part of a complete search when tackling scheduling optimization problems.

QAOA embodies a technique using the variational method that enables approximation algorithms for combinatorial optimization problems [25]. One of its main features is the ability to control the depth of the quantum circuits required by the algorithm, so that, in principle, one can restrict the algorithm to shallow quantum circuits while sacrificing solution quality.

Speeding up Classical Algorithms for Optimization Problems. Good candidate problems for potential speed-up are members of classes of computational problems for which no classical algorithm is known to solve the problem faster than a certain exponential time. As example serve the NP-complete problems for which exact solutions are desired. While there is no expectation to solve an NP-complete problem using a quantum algorithm in polynomial time [1], it is worthwhile investigating how to combine quantum routines with classical problem-solving strategies to speed up solutions for such problems and in turn enable faster practical implementations.

Tran et al. [72] advocate that *"an effective approach to solving complex problems is to decompose them and integrate dedicated solvers for those sub-problems."* While hybrid algorithms can be used to realize such strategy, by decomposing the problem on the CPU, and invoking quantum building blocks on the QPU, or by solving sufficiently small sub-problems directly. This approach is highly appropriate for NISQ systems. Also, classical algorithm design-techniques such as divide-and-conquer and dynamic programming make use of the idea to solve smaller sub-problems and then combining those to solutions for the problem instance. Also, subroutines such as a search or sorting are taking advantage of solving smaller sub-problems as

part of a larger algorithm. Can such ideas be realized for designing hybrid algorithms for NP-complete problems?

Since NP-complete problems typically can be solved using a brute-force search, Grover's search algorithm can be applied to achieve a quadratic speedup of the brute-force algorithm. More efficient strategies involve exact methods to solve NP-complete problems including dynamic programming [5, 35, 74], divide and conquer [3, 29], and other methods in the toolkits of exact algorithms [28] and fixed-parameter tractability [16, 21].

For a number of fixed-parameter algorithms that consist of a kernelization step (i.e., a polynomial-time pre-processing algorithm that reduces the given decision problem to a (smaller) one that has a size that depends on a given fixed-parameter instead of the original input size), followed by a brute-force Grover's search, would achieve a speedup over the classical running time. However, in many cases applying Grover's search to speed up an existing sophisticated exponential-time algorithm is difficult.

Ambainis et al. [3] describe how to apply Grover's search [30] or its variant of finding a minimum item in a set [24] to speedup particular exponential-time dynamic-programming algorithms, for example for the NP-complete problems Hamiltonian Circuit, TSP [5, 35] or Minimum Set Cover [28]. What these dynamic-programming approaches have in common is that they solve subproblems corresponding to subsets of an n-element set. Such a dynamic programming approach in its most basic form typically runs in exponential time in the size of the input. The achieved dynamic programming quantum speed-up relies on pre-computing solutions for sub-problems of a specific size (e.g., say, a quarter of the original instance size) using the classical dynamic programming algorithm, followed by searching—on the remaining subsets—for an answer to the problem using a quantum search in the pre-computed answers for the sub-problems. With their approach, they can improve upon the famous algorithm by Bellman, Held, and Karp from 1962, which is still state of the art for general TSP instances.

While in quantum dynamic programming algorithms as described by the approach above, "quantum power" is used in a *top down* manner by applying Grover's search to large sets of precomputed solutions of sub-problems, a *bottom up* approach is used in a framework of hybrid divide and conquer algorithms: Ge and Dunjko [29] describe the technique by generalizing the specific quantum algorithm [22] that solves the problem 3SAT by introducing a quantum version of Schöning's algorithm [63], and is designed specifically to work on small quantum devices. They show that applying a quantum algorithm to small sub-problems of a specific size (e.g., a quarter of the original input size) in the divide-and-conquer method yields a polynomial speed-up that is achievable using quantum computers with only a few qubits, and therefore such an algorithm can be run on NISQ systems. Note that for this general hybrid divide-and-conquer approach, a quantum algorithm solving the problem in question is necessary to be executed on sufficiently small sub-instances.

4 HYBRID QUANTUM MACHINE LEARNING

In 2001, Gupta et al. proposed a mathematical model of computation called *Quantum Neural Network (QNN)*, which is based on Deutsch's model of computational networks [18]. Since then, the fields of quantum computing and classical machine learning have exploded [8, 75]. With the advent of NISQ systems, using quantum computing for machine learning has been revisited in the past decade. Conversely, the well-established classical machine learning methods help extend and improve quantum information theory [66]. Schuld and Killoran [65] outline similarities between kernel methods in machine learning and quantum computing and lay the theoretical foundations of *quantum machine learning (QML)*. *Classical machine learning (CML)* is classified into *supervised learning*, *unsupervised learning*, and *reinforcement learning*. Quantum computing can play a role in different parts of the ML pipeline.

Recently, deep learning techniques have lead to breakthroughs in the analysis of vision, text, audio, and speech [45]. These techniques rely on vectors and tensors that are transformed from one representation to a high dimensional representation where complex functions can be learned. These computational units are continuous and are currently only approximated on classical computers. One area of research is investigating whether the continuous nature of quantum computers can be leveraged to perform computations with continuous tensors and vectors. Unlike qubit-based quantum computers, the continuous model leverages the wave-like properties of nature. Killoran et al. describe quantum neural networks that use the Continuous-Variable (CV) model [43]. Here, instead of encoding quantum states as qubits, quantum states are encoded as electromagnetic fields. Strawberry Fields [44] is a software framework that is based on the CV model of quantum computing.

Supervised Machine Learning. Supervised ML uses labeled data to identify labels for unlabeled data. Supervised ML techniques, such as classification, have largely satisfactory algorithms in classical computing [15, 48, 76]. Havlíček et al. address the limitations to classification problems that have large feature spaces, where kernel functions are computationally expensive to solve classically [34]. Their methods take advantage of exponentially large quantum states. Rebentrost et al. [60] introduce a *quantum support vector machine (QSVM)* for big-data classification that relies on exponentiation techniques for non-sparse matrices for training with a complexity that is logarithmic in feature size and the number of training data points (the complexity of the classical SVM is linear in comparison). Farhi and Neven [26] describe a generic framework for implementing *QNNs* for supervised learning with classical simulation and show that these networks can indeed be used to classify data but were restricted to about 17 qubits for simulation.

Schuld et al. [64] describe a low depth variational quantum-classical training scheme for a weak non-linear classification. They compare this against a purely classical implementation, provide the mapping of quantum gates to layers of a neural network, and introduce a quantum dropout regularization scheme. The quantum-classical implementation (using simulators) performs well on standard classical benchmark datasets, such as MNIST256 and SEMEION. However, it is prone to overfitting without regularization.

Unsupervised Machine Learning. Unsupervised ML looks for patterns in a dataset with no pre-existing labels. Perdomo-Ortiz et al. [57] describe some of the challenges and opportunities for quantum-assisted ML and argue that the promising "killer" applications in quantum computing might not come from the well-researched area of supervised ML, but rather from the areas the classical ML faces

challenges, such as unsupervised and semi-supervised learning. Otterbach et al. [56] describe a hybrid approach to solving clustering problems using QAOA. They leverage the statistical distributions that are available on quantum computers to improve performance in clustering. QAOA has also been used for generative models of ML. Verdon et al. describe such a low-depth hybrid algorithm for generative neural network learning [73]. Related research includes the use of quantum annealers for the implementation of QML, which is limited by the sparse connectivity among qubits in the physical hardware. Benedetti et al. address this challenge demonstrating the feasibility of using quantum annealers for implementing unsupervised ML models [6]. Lloyd et al. provide an overview of some hybrid paradigms for cluster assignment and cluster finding [47]. The authors describe algorithms where quantum speedups can occur in different parts of the ML pipeline especially for "big quantum data." They provide algorithms for exponential speedups (compared to their classical counterparts) in estimating distances and inner products. For unsupervised techniques, such as clustering, their clustering algorithm exhibits an exponential speedup as opposed to the classical counterpart.

Reinforcement Learning. Reinforcement learning is an ML paradigm that studies how (software) agents take actions to maximize the cumulative reward. Reinforcement learning differs from supervised and unsupervised techniques—the focus is on balancing exploration (discovering unchartered territory) and exploitation (usage of the agents' current knowledge). Using hybrid algorithms for reinforcement learning is a recent field of study, due to Dong et al. [20], who showed promising results in this area—they found a good trade-off between exploration and exploitation that speeds up the learning process. Dunjko et al. [23] describe quantum-accessible reinforcement learning for certain game playing scenarios where the games have a recursive structure and the agent can learn by playing against itself. Examples of such games include AlphaGo and AlphaGo Zero, which have been studied extensively in classical reinforcement learning [69].

5 HYBRID VARIATIONAL ALGORITHMS

Over the past decade, key quantum algorithms emerged for optimization, machine learning, and simulation problems. Popular gems include VQE [58], QAOA [25], and identifying classifiers in machine learning [64].

At the core of these algorithms are the *variational principle & method* and the notion of a *hybrid quantum-classical variational algorithm*, which together represent one of the most promising and practical algorithmic patterns or frameworks for near-term NISQ systems.

As for any other hybrid approach, *variational quantum algorithms (VQA)* involve iterating between a CPU and a QPU. The variational quantum circuit on the QPU gets parameters as inputs from the CPU and delivers expectation values as outputs back to the CPU. A classical optimization loop, minimizing a cost function and executing on the CPU, drives the hybrid algorithm by repeatedly feeding revised parameters to the variational quantum circuit.

VQAs have been proposed for many different applications [12], including computational quantum chemistry and molecule simulation [39, 50], computationally hard combinatorial optimization problems [41, 72], such as MaxCut, Scheduling, TSP, and Knapsack, linear algebra problems [77] including solving systems of linear equations [46], integer factoring [4], and machine learning [64].

6 CONCLUSIONS

Now that a quantum technology industry has emerged in the NISQ era with QPUs, QDKs, algorithmic toolkits including practical variational algorithms and circuits for the near-term, it is imperative that we also focus on quantum software engineering before being overwhelmed with a quantum software crisis as we faced for classical computing in 1968 when the term software engineering was coined [53].

In their inspiring 2017 paper, Rigetti engineers [78] argued convincingly that *"we need a new breed of quantum programmer to study and implement quantum software—with a skillset between that of a quantum information theorist and a software engineer."* Scientist and engineers, who implement hybrid quantum-classical toolkits, algorithms, frameworks, architectures, and platforms, need not only quantum information skills but also software engineering and domain-specific application skills.

This paper aimed to outline the landscape of hybrid quantum-classical problem solving for NISQ systems. After presenting fundamental building blocks for hybrid problem solving and algorithms, we described hybrid approaches for solving quantum optimization, simulation, and machine learning problems. The most promising and practical avenue in the quest to achieve quantum advantage in the NISQ era is called *hybrid quantum-classical variational algorithms and circuits*. In summary, this paper provides solid starting points for researchers interested in experimenting with practical hybrid quantum-classical problem-solving approaches, especially variational ones, for different application domains. Software engineering skills are critical for building and generating the infrastructure for highly scaleable hybrid quantum-classical ecosystems.

REFERENCES

[1] S. Aaronson. 2013. *Quantum computing since Democritus*. Cambridge Univ. Press.
[2] S. Aaronson. 2015. Read the fine print. *Nature Phys.* 11, 4 (2015), 291–293.
[3] A. Ambainis, K. Balodis, J. Iraids, M. Kokainis, K. Prūsis, and J. Vihrovs. 2019. Quantum Speedups for Exponential-Time Dynamic Programming Algor.. In *Proc. 30th Ann. ACM-SIAM Symp. on Discr. Algor. (SODA)*. ACM, 1783–1793.
[4] E. R. Anschuetz, J. P. Olson, A. Aspuru-Guzik, and Y. Cao. 2018. Variational Quantum Factoring. arXiv:1808.08927
[5] R. Bellman. 1962. Dynamic Programming Treatment of the Travelling Salesman Problem. *J. ACM* 9, 1 (1962), 61–63.
[6] M. Benedetti, J. Realpe-Gómez, R. Biswas, and A. Perdomo-Ortiz. 2017. Quantum-assisted learning of hardware-embedded probabilistic graphical models. *Phys. Rev. X* 7, 4 (2017), 52–54.
[7] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, M. S. Alam, S. Ahmed, J. M. Arrazola, C. Blank, A. Delgado, S. Jahangiri, K. McKiernan, J. J. Meyer, Z. Niu, A. Száva, and N. Killoran. 2018. PennyLane: Automatic differentiation of hybrid quantum-classical computations. arXiv:1811.04968
[8] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd. 2017. Quantum Mach. Learn. *Nature* 549, 7671 (2017), 195–202.
[9] M. Born and V. Fock. 1928. Beweis des Adiabatensatzes. *Matrix* 6 (1928).
[10] S. Bravyi, D. Gosset, and R. König. 2018. Quantum advantage with shallow circuits. *Science* 362, 6412 (2018), 308–311.
[11] W. Buchanan and A. Woodward. 2017. Will quantum computers be the end of public key encryption? *J. Cyber Sec. Techn.* 1, 1 (2017), 1–22.
[12] M. Cerezo, K. Sharma, A. Arrasmith, and P. J. Coles. 2020. Variational Quantum State Eigensolver. arXiv:2004.01372
[13] D. Coppersmith. 2002. An approximate Fourier transform useful in quantum factoring. arXiv:0201067
[14] A. D. Corcoles, A. Kandala, A. Javadi-Abhari, D. T. McClure, A. W. Cross, K. Temme, P. D. Nation, M. Steffen, and J. M. Gambetta. 2019. Challenges and

Opportunities of Near-Term Quantum Comp. Systems. *Proc. of the IEEE* (2019), 1–15.
[15] C. Cortes and V. Vapnik. 1995. Support-vector networks. *Machine learning* 20, 3 (1995), 273–297.
[16] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. 2015. *Parameterized Algorithms*. Springer.
[17] D. Deutsch. 1985. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proc. R. Soc. Lond.* 400 (1985), 97–117.
[18] D. Deutsch. 1989. Quantum computational networks. *Proc. R. Soc. Lond.* 425 (1989), 73–90.
[19] D. Deutsch and R. Jozsa. 1992. Rapid solution of problems by quantum computation. *Proc. R. Soc. Lond.* 439 (1992), 553–558.
[20] D. Dong, C. Chen, H. Li, and T. J. Tarn. 2008. Quantum reinforcement learning. *IEEE Trans. Systems, Man, and Cybern.* 38, 5 (2008), 1207–1220.
[21] R. Downey, M. Fellows, and U. Stege. 1998. Parameterized Complexity: A Framework for Systematically Confronting Computational Intractability. *DIMACS Ser. in Discr. Mathem. and TCS* 49 (1998).
[22] V. Dunjko, Y. Ge, and J. I. Cirac. 2018. Computational Speedups Using Small Quantum Devices. *Phys. Rev. Lett.* 121, 25 (2018).
[23] V. Dunjko, Y.-K. Liu, X. Wu, and J. M. Taylor. 2017. Exponential improvements for quantum-accessible reinforcement learning. (2017). arXiv:1710.11160
[24] C. Dürr and P. Høyer. 1996. A Quantum Algorithm for Finding the Minimum. *CoRR* 9607014 (1996).
[25] E. Farhi, J. Goldstone, and S. Gutmann. 2014. A Quantum Approximate Optimization Algorithm. arXiv:1411.4028
[26] E. Farhi and H. Neven. 2018. Classification with quantum neural networks on near perm processors. (2018). arXiv:1802.06002
[27] R. P. Feynman. 1982. Simulating physics with computers. *Int. J. Theor. Phys.* 21, 6–7 (1982), 467–488.
[28] F. V. Fomin and D. Kratsch. 2010. *Exact Exponential Algorithms*. Springer.
[29] Y. Ge and V. Dunjko. 2020. A hybrid algorithm framework for quantum computers with application to finding Hamiltonian cycles. *J. Math. Phys.* 61 (2020).
[30] L. K. Grover. 1996. A fast quantum mechanical algorithm for database search. In *Proc. 28th Ann. ACM Symp. on Theory of Computing (STOC 1996)*. 212–219.
[31] L. K. Grover and J. Radhakrishnan. 2005. Is Partial Quantum Search of a Database Any Easier?. In *Proc. 17th Ann. ACM Symp. Parallel. in Alg. and Arch.* 186–194.
[32] E. Grumbling and M. Horowitz (Eds.). 2019. *Quantum Computing: Progress and Prospects (2019)*. National Academies of Sciences, Engineering, and Medicine. The National Academies Press. https://doi.org/10.17226/25196
[33] A. W. Harrow, A. Hassidim, and S. Lloyd. 2009. Quantum algorithm for linear systems of equations. *Phys. Rev. Lett.* 103, 150502 (2009).
[34] V. Havlíček, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, and J. M. Gambetta. 2019. Supervised learning with quantum-enhanced feature spaces. *Nature* 567, 7747 (2019), 209–212.
[35] M. Held and R. M. Karp. 1961. A Dynamic Programming Approach to Sequencing Problems. In *Proc. 16th ACM National Meeting*. 201–204.
[36] IBM Quantum Experience. [n.d.]. Qiskit Aqua: Algorithms for quantum computing applications. https://qiskit.org/aqua/
[37] M. W. Johnson, M. H. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A. J. Berkley, J. Johansson, P. Bunyk, E. M. Chapple, C. Enderud, J. P. Hilton, K. Karimi, E. Ladizinsky, N. Ladizinsky, T. Oh, I. Perminov, C. Rich, M. C. Thom, E. Tolkacheva, C. J. Truncik, S. Uchaikin, J. Wang, B. Wilson, and G. Rose. 2011. Quantum annealing with manufactured spins. *Nature* 473, 7346 (2011), 194–198.
[38] T. Kadowaki and H. Nishimori. 1998. Quantum annealing in the transverse Ising model. *Phys. Rev. E* 58, 5 (1998), 5355–5363.
[39] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta. 2017. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature* 549, 7671 (Sep 2017), 242–246.
[40] P. J. Karalekas, N. A. Tezak, E. C. Peterson, C. A. Ryan, M. P. da Silva, and R. S. Smith. 2020. A quantum-classical cloud platform optimized for variational hybrid algorithms. *Quantum Sci. Techn.* 5, 2 (2020), 024003.
[41] S. Khairy, R. Shaydulin, L. Cincio, Y. Alexeev, and P. Balaprakash. 2019. Learning to optimize variational quantum circuits to solve combinatorial problems. arXiv:1911.11071
[42] A. Khoshaman, W. Vinci, B. Denis, E. Andriyash, H. Sadeghi, and M. H. Amin. 2018. Quantum variational autoencoder. *Quantum Sci. Techn.* 4, 1 (2018), 014001.
[43] N. Killoran, T. Bromley, J. Arrazola, M. Schuld, N. Quesada, and S. Lloyd. 2019. Continuous-variable quantum neural networks. *Phys. Rev. Res.* 1, 3 (2019), 1–21.
[44] N. Killoran, J. Izaac, N. Quesada, V. Bergholm, M. Amy, and C. Weedbrook. 2019. Strawberry Fields: A Software platform for photonic quantum computing. *Quantum* 3 (2019), 129.
[45] Y. Lecun, Y. Bengio, and G. Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.
[46] Y. Lee, J. Joo, and S. Lee. 2019. Hybrid quantum linear equation algorithm and its experimental test on IBM Quantum Experience. *Nature Scient. Reports* 9 (2019).
[47] S. Lloyd, M. Mohseni, and P. Rebentrost. 2013. Quantum algorithms for supervised and unsupervised machine learning. arXiv:1307.0411
[48] M. E. Maron. 1961. Automatic indexing: An experimental inquiry. *J. ACM* 8, 3 (1961), 404–417.
[49] M. Martonosi and M. Roetteler. 2019. *Next Steps in Quantum Computing: Computer Science's Role*. Technical Report. Computing Community Consortium (CCC).
[50] S. McArdle, S. Endo, A. Aspuru-Guzik, S. Benjamin, and X. Yuan. 2020. Quantum computational chemistry. *Rev. Mod. Phys.* 92, 1 (2020), 015003.
[51] A. J. McCaskey, D. I. Lyakh, E. F. Dumitrescu, S. S. Powers, and T. S. Humble. 2019. XACC: A System-Level Software Infrastructure for Heterogeneous Quantum-Classical Computing. *Quantum Sci. Techn.* 5, 2 (2019), 024002.
[52] A. J. McCaskey, Z. P. Parks, J. Jakowski, S. V. Moore, T. D. Morris, T. S. Humble, and R. C. Pooser. 2019. Quantum chemistry as a benchmark for near-term quantum computers. *npj Quantum Inf.* 5 (2019), 99.
[53] P. Naur and B. Randell. 1968. Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee.
[54] M. A. Nielsen and I. L. Chuang. 2011. *Quantum Computation and Quantum Information*. Cambridge University Press.
[55] T. E. O'Brien, B. Tarasinski, and B. M. Terhal. 2019. Quantum phase estimation of multiple eigenvalues for small-scale (noisy) experiments. *New J. Phys.* 21, 2 (2019), 023022:1–28.
[56] J. S. Otterbach, R. Manenti, N. Alidoust, A. Bestwick, M. Block, B. Bloom, S. Caldwell, N. Didier, E. S. Fried, S. Hong, P. Karalekas, C. B. Osborn, A. Papageorge, E. C. Peterson, G. Prawiroatmodjo, N. Rubin, C. A. Ryan, D. Scarabelli, M. Scheer, E. A. Sete, P. Sivarajah, R. S. Smith, A. Staley, N. Tezak, W. J. Zeng, A. Hudson, B. R. Johnson, M. Reagor, M. P. da Silva, and C. Rigetti. 2017. Unsupervised machine learning on a hybrid quantum computer. (2017). arXiv:1712.05771
[57] A. Perdomo-Ortiz, M. Benedetti, J. Realpe-Gómez, and R. Biswas. 2018. Opportunities and challenges for quantum-assisted machine learning in near-term quantum computers. *Quantum Sci. Technol.* 3, 3 (2018), 30502.
[58] A. Peruzzo, J. McClean, P. Shadbolt, M. H. Yung, X. Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'Brien. 2014. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications* 5 (2014).
[59] J. Preskill. 2018. Quantum Computing in the NISQ era and beyond. *Quantum* 2 (2018), 79.
[60] P. Rebentrost, M. Mohseni, and S. Lloyd. 2014. Quantum support vector machine for big data classification. *Phys. Rev. Lett.* 113, 3 (2014), 1–5. arXiv:1307.0471
[61] D. Ristè, M. Silva, C. Ryan, A. Cross, J. Smolin, J. Gambetta, J. Chow, and B. Johnson. 2017. Demonstration of quantum advantage in machine learning. *npj Quantum Information* 3 (2017), 1–5.
[62] G. E. Santoro and E. Tosatti. 2006. Optimization using quantum mechanics: Quantum annealing through adiabatic evolution. *J. Phys.* 39, 36 (2006).
[63] U. Schöning. 1999. A Probabilistic Algorithm for K-SAT and Constraint Satisfaction Problems. In *Proc. 40th Ann. Symp. Found. of Comp. Sci. (FOCS)*. 410.
[64] M. Schuld, A. Bocharov, K. M. Svore, and N. Wiebe. 2020. Circuit-centric quantum classifiers. *Phys. Rev. A* 101, 3 (2020).
[65] M. Schuld and N. Killoran. 2019. Quantum machine learning in feature Hilbert spaces. *Phys. Rev. Lett.* 122, 4 (2019), 40504:1–12.
[66] M. Schuld, I. Sinayskiy, and F. Petruccione. 2015. An introduction to quantum machine learning. *Contemp. Phys.* 56, 2 (2015), 172–185.
[67] N. Shenvi, J. Kempe, and K. B. Whaley. 2003. Quantum random-walk search algorithm. *Phys. Rev. A* 67, 5 (2003), 052307:1–7.
[68] P. W. Shor. 1994. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proc. 35th Ann. Symp. Found. of Comp. Sci. (FOCS)*. 124–134.
[69] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Van Den Driessche, T. Graepel, and D. Hassabis. 2017. Mastering the game of Go without human knowledge. *Nature* 550, 7676 (2017), 354–359.
[70] M. Stechly. 2019. Variational Quantum Eigensolver Explained. https://www.mustythoughts.com/variational-quantum-eigensolver-explained
[71] R. Sutor, T. Hickey, and L. Feller. 2018. Taking the quantum leap. https://www.ibm.com/thought-leadership/institute-business-value/report/quantumleap
[72] T. T. Tran, M. Do, E. G. Rieffel, J. Frank, Z. Wang, B. O'Gorman, D. Venturelli, and J. C. Beck. 2016. A Hybrid Quantum-Classical Approach to Solving Scheduling Problems. In *Proc. 9th Int. Symp. Comb. Search (SoCS)*. 1–9.
[73] G. Verdon, M. Broughton, and J. Biamonte. 2017. A quantum algorithm to train neural networks using low-depth circuits. (2017). arXiv:1712.05304
[74] X. Wang and J. Tian. 2011. Dynamic Programming for NP-Hard Problems. *Procedia Engineering* 15 (2011), 3396–3400.
[75] P. Wittek. 2014. *Quantum machine learning: What quantum computing means to data mining*. Academic Press.
[76] X. Wu, V. Kumar, Q. J. Ross, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z. H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg. 2008. Top 10 algorithms in data mining. *Knowledge and Inf. Syst.* 14, 1 (2008), 1–37.
[77] X. Xu, J. Sun, S. Endo, Y. Li, S. C. Benjamin, and X. Yuan. 2019. Variational algorithms for linear algebra. arXiv:1909.03898
[78] W. Zeng, B. Johnson, R. Smith, N. Rubin, M. Reagor, C. Ryan, and C. Rigetti. 2017. First quantum computers need smart software. *Nature* 549 (09 2017), 149–151.

Workshops

CASCON x EVOKE 2020

1st Workshop on AIOps and Systems Compliance

Kostas Kontogiannis
University of Western Ontario
London, ON., Canada
kostas@csd.uwo.ca

Alberto Giammaria
IBM
Austin, TX., USA
agiammar@us.ibm.com

Chris Brealey
IBM
Toronto, ON., Canada
cbrealey@ca.ibm.com

Marios Grigoriou
University of Western Ontario
London, ON., Canada
mgrigori@uwo.ca

ABSTRACT

Over the past two years we have seen a growth on AIOps. AIOps is the area of software engineering which aims to apply Artificial Intelligence on IT operations and on the big data which are harvested from different IT systems across an organisation. AIOps comes as a direct response to the dire need to break the siloed approach to IT operations in order to support a holistic system analysis which allows not only for the identification and proactive or even predictive resolution of issues, but also for assessing the compliance of a system against certain policies such as policies related to safety, security, and regulatory. This workshop brought together researchers and practitioners to discuss the latest developments on AIOps and system compliance and identified key challenge issues.

CCS CONCEPTS

• **Software and its engineering** → *Software verification and validation*; **Development frameworks and environments**;

KEYWORDS

Software bug prediction, machine learning, code repositories, software metrics

ACM Reference Format:
Kostas Kontogiannis, Alberto Giammaria, Chris Brealey, and Marios Griogriou. 2020. 1st Workshop on AIOps and Systems Compliance. In *CASCON Evoke 2020: 30st Annual International Conference on Computer Science and Software Engineering, November 10-13, 2020, Markham, Ontario, Canada. IBM Corp., Riverton, NJ, USA, 2 pages.*

1 INTRODUCTION

Large software systems encompass complex interactions among their components and are subjected to frequent maintenance activities applied in order to fix bugs, add new functionality, port to new platforms, or interoperate with other systems. An important issue to consider, is how such activities can be achieved in a way that first minimizes the risk of failures, and second how these maintenance activities can be integrated in a continuous deployment (CD) / continuous integration (CI) DevOps process. One major aspect on achieving this objective is to identify and remediate early on, possible vulnerabilities which are manifested as violations of known published controls. The solution to this problem is even more important for large scale systems such as federal enterprise information systems. In this respect, a key part of the certification and accreditation process for large enterprise information systems is selecting and implementing a subset of the controls (safeguards) from the Security Control Catalog (see NIST 800-53 vulnerabilities controls list). Another part is the mining of run-time data and the consequent analysis and reasoning on such data so that issues can be identified before they occur and move from a reactive process dealing with compliance and system vulnerabilities to a proactive process where issues are identified and resolved proactively or even predictively. This workshop brought together researchers and practitioners in order to discuss and debate the state-of-the-art and state-of-practice techniques for modeling information collected from diverse logging and monitoring infrastructures, techniques for analysing run-time data to proactively identify and resolve risks, and techniques to assess the compliance against policies aiming to avoid security risks and known vulnerabilities, achieving thus a state we refer to as "Continuous Compliance".

CASCON 2020, November 10-13, 2020, Toronto, Canada

2 WORKSHOP TOPIC AREAS

2.1 Modeling of Big IT Data

Modeling, reconciling, and storing data collected from different logging and monitoring tools were topics discussed in the workshop. The major objective here is to be able to break the siloed approach in IT where different applications and systems in an organization have their own data collected and stored in a way that is not easily shareable with other units so that a holistic analysis across all IT systems can be performed. More specifically, the issues raised had to do with *a)* schemas for modeling data collected from diverse tools; *b)* techniques to reconcile collected data including schema normalization, heuristics related to timestamping, semantics and informal information, and association mappings between the individual logger schemas; *c)* archiving methods for efficient processing, including distributed data repositories and; *d)* identification of noisy or unrelated data by efficiently filtering big data pools.

2.2 Proactive Issue Identification

Proactive and predictive issue identification were two other topics discussed in the workshop. These topics pose a number of open

problems that require our attention for devising efficient solutions for. First, we need to devise techniques to predictively identify issues and warn system operators of the possibility of a violation or failure. In this respect, the attendees discussed the use of reasoning techniques, AI, and modeling frameworks which can be used to associate the possible root causes with system behavior. The major challenge here is how to identify system states or system behavior which can elude that an imminent failure, violation, or risky situation can arise. In order to address this challenge one has not only to go over large volumes of logged data at real or near real-time, but also to identify those pieces of data and events which are string predictors of impeding issues.

2.3 Intelligent Issue Resolution

Another area presented and discussed during the workshop was how to resolve issues arising during system operation. The attendees debated three major facets of this problem. The first facet has to do with techniques which can be used to zoom-in the attention and focus of engineers towards investigating the root cause of a problem and associating root causes with remedial actions. The second facet has to do with techniques to proactively and constantly re-configuring a system so that system states which can lead to a failure or a security violation can be avoided before they pose a major threat or violation. Finally, the third facet has to do with analytics techniques for identifying the most qualified engineering team or developer to assign an issue for resolution.

2.4 Issue Impact and Risk Determination

The next area discussed in the workshop was how to identify system vulnerabilities and how to assess and possibly quantify the risk and potential impact of these vulnerabilities on IT operations. This was identified as a very challenging area which highly relates to compliance (see below). The attendees discussed open issues and challenges that need be addressed. These included *a)* schemas for modeling vulnerabilities as these are presented in standard threat and vulnerability tables (e.g. the NIST 800.53 lists); *b)* schemas and structures for modeling complex system component dependencies and; *c)* modeling the potential impact an issue or a vulnerability on a component has on system operations. A key challenge here is to be able to identify the scope and boundaries a vulnerability can extend given the operational context system component dependencies, and business flows.

2.5 Compliance Analysis

Compliance analysis has been identified as a major issue in the discussion during the workshop. The problem has been consid-ered as a multi-faceted one. The first facet is the type of compliance sought. Here, the attendees identified three major areas and namely, compliance related to regulatory constraints, compliance related to service level agreements, and compliance related to security threats and vulnerabilities. The second facet has to do with modeling issues and most specifically how policies are modeled and associated in a way that can facilitate efficient processing. The third facet deals with the design of reasoning engines in order to be able to evaluate vulnerability policies against system data and configuration mod-els. Finally, the fourth facet relates to the use of analytics for the prediction of the likelihood a vulnerability can occur on a given system state or configuration and the estimation of effort requited to resolve a vulnerability.

3 KEY TAKE-AWAYS

The workshop attendees identified three key short and medium term challenge topics in the area of AIOPs, as follows:

- *Continuous Compliance*: The focus here is compliance of cloud offerings both in the in the production and operations environment. Continuous compliance can be achieved through a process where compliance controls are continuously monitored (default time-interval can be 1 day or continuous), and when deviations are detected they are remediated in a period of time based on the selected baseline-impact and the control's severity and priority. Sub-areas here include *a)* the automatic detection of anomalous/risky situations; *a)* the evaluation of a weighted overall compliance status or score of a system; *c)* the compilation of compliance state related at-a-glance graphs and heat maps and *d)* the design of software analytics techniques for failure and vulnerability prediction.
- *Data extraction and modelling*: The focus here is the design and implementation of miners whereby information extracted from logging and monitoring tools can be reconciled, modelled, and stored for efficient processing. The objective of such processing id to assess whether compliance is achieved or not and whether the overall compliance level indicators deteriorate from one release to the next, or from one system configuration to another.
- *Vulnerabilities assessment reasoning frameworks*: The focus here is to investigate probabilistic or fuzzy reasoning frameworks in order to assess and quantify the likelihood of compliance violations on a given offering as a function of planned changes, bug fixes, or feature enhancements.

REFERENCES

[1] IBM AIOps. In *https://www.ibm.com/cloud/learn/aiops.*

[2] Y. Dang, Q. Lin, P. Huang. 2019. AIOps: Real-World Challenges and Research Innovations. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion).* ACM, pp.4–5.

[3] Yangguang Li, et al. 2020. Predicting Node Failures in an Ultra-Large-Scale Cloud Computing Platform: An AIOps Solution. In *ACM Trans. Softw. Eng. Methodol. 29, 2, Article 13 (April 2020).* ACM.

[4] Masood A., Hashmi A. 2020. AIOps: Predictive Analytics & Machine Learning in Operations. In *Cognitive Computing Recipes.* Apress, Berkeley, CA.

[5] Fan Y., et al. 2020. Design of Integrated Operation and Maintenance Platform Based on AIOps. In *Kountchev R., Patnaik S., Shi J., Favorskaya M. (eds) Advances in 3D Image and Graphics Representation, Analysis, Computing and Information Technology. Smart Innovation, Systems and Technologies, vol 180.* Springer.

[6] Banica, L., et al. 2020. Empowering IT Operations through Artificial Intelligence – A New Business Perspective. In *KnE Social Sciences, 4(1).* Springer, pp.412–425.

[7] Mohanty S., Vyas S. 2018. IT Operations and AI. In *How to Compete in the Age of Artificial Intelligence.* Apress, Berkeley, CA.

[8] D. Zhang, S., et al. 2013. Software Analytics in Practice. In *IEEE Software, Sept. 2013.* IEEE, pp.30–37.

[9] Gulenko A., et al. 2020. Anomaly Detection and Levels of Automation for AI-Supported System Administration. In *Lossio-Ventura J., Condori-Fernandez N., Valverde-Rebaza J. (eds) Information Management and Big Data. SIMBig 2019. Communications in Computer and Information Science, vol 1070.* Springer, pp.1–7.

Automation, Control, and Analysis of Knowledge-intensive Processes

Eric Yu
Arik Senderovich
eric.yu@utoronto.ca
arik.senderovich@utoronto.ca
Faculty of Information,
University of Toronto
Toronto, Canada

Hajo A. Reijers
h.a.reijers@uu.nl
Department of Information and
Computing Sciences,
Utrecht University
Utrecht, The Netherlands

Allen Chan
Sebastian Carbajales
avchan@ca.ibm.com
sebastia@ca.ibm.com
IBM Canada
Toronto, Canada

ABSTRACT

Process automation has been a widely used methodology for improving business processes since the late 90's. Typically, automation was applied to well-structured and highly routine business processes. Knowledge-intensive processes that exhibit ad-hoc workflows and involve mainly knowledge workers are now becoming more and more common in most industries. Traditionally, these processes were considered harder to model, analyze and thus automate due to their unstructured and flexible nature. Recent advances in the worlds of Artificial Intelligence and Machine Learning supported by ample data availability has led to developments in knowledge-intensive process modeling, analysis, and automation. Specifically, applications such as natural language processing and information retrieval allows researchers to identify workflows in unstructured data (e.g., emails) and map ad-hoc behavior to workflow patterns. In this workshop, we aim at sharing experiences and research results on recent advances in data-driven approaches to process automation with a special emphasis on ad-hoc business processes, i.e., work patterns with no predefined process models. The workshop will explore theories, algorithms, and engineering methods at the intersection of AI and process management to develop next generation intelligent business automation solutions. Speakers will be researchers and industry leaders from diverse backgrounds including but not limited to process automation in knowledge-intensive processes, business process management, data-driven activity and workflow recognition, and flexible process mining using constraint learning. The workshop will give the audience a broad exposure to the field, and create opportunities for collaborations among academic research groups and industry teams between various partners that wish to develop The workshop will run as a single-session online event and will be adjusted to the special circumstances of the new normal under COVID-19 restrictions.

CASCON 2020, November 10-13, 2020, Toronto, Canada

KEYWORDS

Business Process Automation; Knowledge-intensive Processes; Ad-Hoc Workflow Management

1 SCIENTIFIC SCOPE

Knowledge-intensive business processes, processes that involve mostly knowledge work, are receiving much attention in the world of business process management and analytics [8]. In fact, many companies realize nowadays that improving knowledge-intensive processes (KiP) often yields a competitive advantage over others [1]. Although these processes are prevalent in many industries, their automation, control and testing has been considered extremely challenging due to their high levels of flexibility and their ad-hoc workflows [2, 12]. Therefore, their improvement is typically enabled by standard process re-engineering techniques such as functional analysis and process audit, rather than on workflow automation [3].

With recent developments of machine learning technology and the ample availability of data, automation of KiP must be reconsidered. Methods like robotic process automation (RPA) are becoming established in routine and structured processes [11, 15]. Furthermore, data science paradigm such as process mining, enable data-driven RPA [10]. However, we identify a gap in attempting to automate knowledge-intensive processes. One exception is the line of work by Di Ciccio et al. [5, 6] that proposes the use of declarative process mining using non-standardized data resources (e.g., emails) to analyze and automate KiP (or what they call, artful processes). The authors claim that having a collection of declarative process representations improve the capability to control and execute the underlying KiPs.

Another stream of work proposes novel requirement engineering and analysis methods for KiPs [4], as well as a process ontology for representing these processes [7]. Another human-oriented process automation methodology uses crowd-sourcing data to elicit process requirements [13]. The approach can be viewed as an alternative to both traditional requirements engineering and to structured process mining, which may be useful for KiPs that do not emit structured

data nor whose workflows cannot be easily identified by standard requirements techniques.

In an innovative paper, Hull and Nezhad propose to re-think the way business processes (and KiPs, beign their special case) are being designed and executed using cognitive computing [9]. Specifically, they propose the Plan-Act-Learn paradigm that would replace the traditional BPM life-cycle. Plan-Act-Learn can support the full spectrum of business processes, starting from routine processes and ending with KiPs, in a seamless and systematic manner. The planning component decides which of the processes (or their parts) are going to be human-centered or automated, the learning component uses data and cognitive AI technologies to extract information that would facilitate automation, and the last phase of acting would devise new decisions and goals based on the plan and learn phases.

Recently, socially-aware process redesign was proposed as an approach to balance between automation and human training was proposed in [14]. To this end, the paper solves an optimization problem that would help managers to decide which parts of a process should be automated, and which parts should be executed by humans. In the proposed workshop, we wish to discuss the natural tension between human-oriented management of KiP and the potential to automate (parts of) these ad-hoc processes.

2 RATIONALE

After providing the scientific background for our workshop, we shall outline the rationale behind proposing it for CASCON. In recent years, the main reasons for why automation has became possible in knowledge-intensive processes is the improvement of the machines ability to understand and extract information from structured, semi-structured, and unstructured data sources or by the ability to replicate human activities (e.g. replicating activities on screen as in RPA). When this type of automation meets BPM, we see the emergence of new industrial agendas in KiP, e.g., 'digital workers' that are participating as delegate or in place of human workers in a workforce. This gives rise to 'hybrid workforce management', which is a workforce composed of both human workers and 'digital workers'.

Hence, the question that we would like to answer in the proposed workshop is 'can we move closer to automating (parts of) knowledge-intensive business processes?'. We believe that answering this question is of high value and relevance for both IBM products and for computer science research in the broader sense. We wish to explore scientific and industry-oriented answers to the question that we posed above at the intersection between traditional process management and automation techniques and data-driven methodologies such as machine learning and process mining. We wish to explore views from researchers who advocate for automation and researchers who speak for a human-centered approach to KiP management, which avoids automation.

We believe that advances in the field of intelligent process automation for KiP is an important and hot topic in the world of computer science research. The workshop will bring together world-leading researchers that would speak to the mentioned topics (both from academia and from industry) with the aim of bringing the community closer to solving some of the challenges posed by KiPs with ad-hoc workflow patterns.

The research will be presented from several viewpoints of the business process management and intelligence fields in their broad sense, with a special focus on their intersection, which include but are not limited to: process automation, process modeling and mining, process control, and process analysis. We shall invite speakers that work on applying techniques from the aforementioned methodologies to knowledge-intensive processes.

3 WORKSHOP FORMAT

The workshop is planned to be a half-day event with virtual coffee breaks between the sessions. We shall invite research and industry speakers that will give half-hour presentations followed by Q&A sessions. To conclude the event, we shall have a half-hour round-table discussion that will allow the audience to participate and ask questions and for the speakers to discuss each other's works.

4 TARGET AUDIENCE AND EXPECTED OUTCOMES

The workshop caters both researchers and practitioners who are working on related scientific and applied areas of business process automation, with special focus on artful and ad-hoc processes. The participants will be exposed to the novelties in the relevant fields and will be given the chance to discuss their research and practice with our world-leading invited speakers. We expect that new collaborations and research directions will flourish among our participants and their potential colleagues. For IBM, we expect that the workshop will facilitate and advance existing CAS projects, and pave the way for novel project ideas and products.

5 ORGANIZERS

Eric Yu is a Professor at the Faculty of Information at the University of Toronto. His research interests include conceptual modeling, software requirements engineering, information systems engineering, knowledge management, enterprise modeling, and most recently, enterprise AI. In his PhD work, he developed the i* framework for social modeling. The work has inspired hundreds of research papers, dozens of PhD theses, and many software tools. A version of i* is part of an international standard. He is co-editor of the MIT Press book series on Information Systems, and is on the editorial boards of the Requirements Engineering journal, IET Software, and the Journal on Data Semantics. He was Program Co- Chair for ER 2008 and 2014, and for CAiSE 2020. He was recipient of the 2019 Peter P. Chen Award.

Arik Senderovich is an Assistant Professor of Human-Centered Data Science at the Faculty of Information at University of Toronto. Before his appointment at the Faculty of Information, he received the Lyon Sachs scholarship and worked as a postdoctoral fellow in the Toronto Intelligent Decision Engineering Laboratory (TIDEL) at the University of Toronto. He received his Ph.D. in the area of process mining, focusing on queueing perspectives in process mining, from the Technion – Israel Institute of Technology in 2016, for which he also received the 2017 best dissertation award at BPM, the annual Business Process Management conference. Arik's research focuses on data analytics in business processes, with emphasis on

complex systems with scarce resources. He published papers on the above in journals, and leading conferences in the field.

Hajo Reijers is a full professor in the Department of Information and Computing Sciences of Utrecht University, where he leads the Business Process (BPM) Management & Analytics group. He is also a part-time, full professor in the Department of Mathematics and Computer Science of Eindhoven University of Technology. Previously, he worked for various management consultancy companies and led the BPM research group at Lexmark. Hajo's research and teaching focus on BPM, data analytics, and information systems engineering. On these and other topics, he published over 200 scientific papers, chapters in edited books, and articles in professional journals. He was Program Co - Chair for BPM 2009 and CAiSE 2018.

Allen Chan is an IBM Distinguished Engineer, a Technical Executive of IBM Digital Business Automation. He is currently the CTO for Digital Business Automation, responsible for IBM Cloud Pak for Automation including capabilities such as Business Process Management and Case Management, Application Designer, and others. He is passionate about ensuring customers' successful use of IBM products. Prior to that role, he had held various technical leadership roles in IBM BPM such as the Chief Architect for Workflow (IBM BPM & Case Manager) and Blueworks Live. In addition, he was the IBM BPM SWAT Technical Lead where he led a team of experts to help ensure customer success in key IBM BPM deployment and rollout scenarios. He is the holder of multiple patents in Canada, United States and China, and has written a number of papers and articles.

Sebastian Carbajales is a senior technical lead and architect for the IBM Business Automation Workflow and IBM Automation Workstream Services (IAWS) components of the IBM Cloud Pak for Automation. As the IAWS architect, his main focus is on enabling the business user to automate their work with a simple, no-code approach. In addition, he is also leading the effort to infuse AI and machine learning into Workflow and Workstreams and making it accessible to business users. His other areas of responsibility include the integration of Business Process Management and Case Management, both from an authoring and execution perspective as well as the Workflow tooling platform, in general.

REFERENCES

[1] Selena Aureli, Daniele Giampaoli, Massimo Ciambotti, and Nick Bontis. 2019. Key factors that improve knowledge-intensive business processes which lead to competitive advantage. *Business process management journal* (2019).

[2] Fabrice Boissier, Irina Rychkova, and Bénédicte Le Grand. 2019. Challenges in Knowledge Intensive Process Management. In *2019 IEEE 23rd International Enterprise Distributed Object Computing Workshop (EDOCW)*. IEEE, 65–74.

[3] Peter Dalmaris, Eric Tsui, Bill Hall, and Bob Smith. 2007. A framework for the improvement of knowledge-intensive business processes. *Business Process Management Journal* (2007).

[4] Claudio Di Ciccio, Andrea Marrella, and Alessandro Russo. 2015. Knowledge-intensive processes: characteristics, requirements and analysis of contemporary approaches. *Journal on Data Semantics* 4, 1 (2015), 29–57.

[5] Claudio Di Ciccio and Massimo Mecella. 2012. Mining constraints for artful processes. In *International Conference on Business Information Systems*. Springer, 11–23.

[6] Claudio Di Ciccio and Massimo Mecella. 2013. Mining artful processes from knowledge workers' emails. *IEEE Internet Computing* 17, 5 (2013), 10–20.

[7] Juliana Baptista dos Santos França, Joanne Manhães Netto, Juliana do ES Carvalho, Flávia Maria Santoro, Fernanda Araujo Baião, and Mariano Pimentel. 2015. KIPO: the knowledge-intensive process ontology. *Software & Systems Modeling* 14, 3 (2015), 1127–1157.

[8] Norbert Gronau and Edzard Weber. 2004. Management of knowledge intensive business processes. In *International Conference on Business Process Management*. Springer, 163–178.

[9] Richard Hull and Hamid R Motahari Nezhad. 2016. Rethinking BPM in a cognitive world: Transforming how we learn and perform business processes. In *International Conference on Business Process Management*. Springer, 3–19.

[10] Volodymyr Leno, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, and Artem Polyvyanyy. 2020. Automated Discovery of Data Transformations for Robotic Process Automation. *arXiv preprint arXiv:2001.01007* (2020).

[11] Volodymyr Leno, Artem Polyvyanyy, Marlon Dumas, Marcello La Rosa, and Fabrizio Maria Maggi. 2020. Robotic Process Mining: Vision and Challenges. *Business & Information Systems Engineering* (2020), 1–14.

[12] Olivera Marjanovic and Ronald Freeze. 2012. Knowledge-intensive business process: deriving a sustainable competitive advantage through business process management and knowledge management integration. *Knowledge and Process Management* 19, 4 (2012), 180–188.

[13] Pradeep K Murukannaiah, Nirav Ajmeri, and Munindar P Singh. 2017. Toward automating crowd RE. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. IEEE, 512–515.

[14] Joop J. Senderovich, Arik Schippers and Hajo A. Reijers. 2020. Socially-aware Business Process Redesign. In *International Conference on Business Process Management*. Springer.

[15] Wil MP Van der Aalst, Martin Bichler, and Armin Heinzl. 2018. Robotic process automation.

Deploying a Collaborative Framework for Crowd Sourcing the Evaluation of AI Model Effectiveness

Sarah Packowski
spackows@ca.ibm.com
IBM

Joshua Allard
jmallard@us.ibm.com
IBM

ABSTRACT

Evaluating the effectiveness of a binary classification model can be as simple as calculating the percent of inputs that are correctly classified by the model.

But when it comes to evaluating the effectiveness of a speech to text model or natural language understanding model, simply counting the number of incorrect words (for example) doesn't capture the nuances required to understand if the model is effective enough to do the job you need it to do. One way to tackle this problem is to experiment with multiple evaluation methods to see what fits your needs best.

In this workshop, participants deployed a sample Python Flask app in IBM Cloud that enables teams to dynamically collect and apply multiple model evaluation methods contributed by collaborators.

In this workshop, participants discussed common challenges with developing collaborative AI or data science solutions, including:

- Streamlining the workflow so specialists can focus on their area of expertise
- Making it easy for all team members to understand all solution components
- Scaling out the assembled solution
- Turning the solution into a platform by exposing endpoints

CCS CONCEPTS

• **Software and its engineering** → **Programming teams**; **Software prototyping**; • **Computing methodologies** → **Natural language processing**.

KEYWORDS

collaboration, prototyping, continuous delivery, AI, effectiveness

CASCON 2020, November 10-13, 2020, Toronto, Canada

1 RATIONALE

Imagine two speech to text models produce the following output for the spoken phrase *the quick, brown fox jumped over the lazy dog*:

Model A result

```
the quick brown fox jumped under the lazy dog
```

Model B result

```
the quick brown fox jumped over the lazy frog
```

Both **Model A** and **Model B** got one word wrong. So, how would you decide which model performed better? The answer would vary, depending on your use case:

- If you are analyzing the transcript to determine *what actions happened*, the result from **Model A** would be more "wrong" (jumping under instead of jumping over)
- If you are analyzing the transcript to determine *who was involved*, the result from **Model B** would be more "wrong" (frog instead of dog)

To assess AI model performance for different use cases like this, our team developed a simple framework that applied multiple evaluation methods simultaneously so we could easily see which methods fit our needs best. The framework was deployed to IBM Cloud using a GitHub-integrated continuous delivery pipeline. Team members implemented each evaluation method in its own Python file; and then when team members uploaded their files to GitHub, the framework was automatically redeployed with their new methods included.

Using this simple framework and continuous delivery pipeline has had several advantages:

- The simplicity of the framework made it easy to get started
- Each team member could focus on their own implementation, without worrying about deployment or app management
- We have reused the framework for multiple, different sorts of projects with only minor adjustments

2 WORKSHOP FORMAT

In this workshop, participants gained hands-on experience deploying a Python Flask app to IBM Cloud, using a continuous delivery pipeline with GitHub.

The direct experience in this workshop, as well as the group discussions about how to enable both generalist and specialist team members to contribute to a solution, gave participants strategies for future collaborative AI or data science projects.

How has COVID-19 changed the development and adoption of data science across firms and industries?

Michelle Alexopoulos
University of Toronto
m.alexopoulos@utoronto.ca

Kelly Lyons
University of Toronto
kelly.lyons@utoronto.ca

Rohan Alexander
University of Toronto
rohan.alexander@utoronto.ca

Aije Egwaikhide
IBM
aije.e@ibm.com

R. Blair Frost
University of Toronto
rb.frost@utoronto.ca

ABSTRACT

Since ancient times, individuals have recognized that innovation and adoption of new technologies is affected by demand (i.e., "necessity is the mother of all invention"). Advances in data science, an interdisciplinary scientific approach that combines computation methods with data to understand and solve problems in an evidence-based manner, is no exception. Prior to the COVID-19 outbreak, the speed of data science adoption within organisations faced barriers such as legal/regulatory challenges, available workforce skills and financial costs. The emergence of the pandemic has altered the incentives to invest in, and adopt these innovations. This shifting landscape, will likely have both short run and long run impacts.

This workshop brought together panel and audience members drawn from academia, government agencies and different industries to discuss how COVID-19 has affected the deployment and development of data science. For discussion purposes, COVID-19 related impacts were grouped into three themes: the effect on Research & Development investment; commercialization of new and existing technologies; and changes in the barriers to adoption across different areas.

CCS CONCEPTS

• **Social and professional topics** → **Government technology policy**; **Computing / technology policy**.

KEYWORDS

COVID-19, data science adoption, challenges, legal, organisational, business practices

ACM Reference Format:
Michelle Alexopoulos, Kelly Lyons, Rohan Alexander, Aije Egwaikhide, and R. Blair Frost. 2020. How has COVID-19 changed the development and adoption of data science across firms and industries?. In *CASCON 2020, November 10-13, 2020, Toronto, ON*. IBM, Riverton, NJ, USA, 2 pages.

CASCON '20, November 10-13, 2020, Toronto, ON

1 BACKGROUND AND RATIONALE

Data science provides methods to understand and solve problems in an evidence-based manner by combining data and experience, with scientific methods. Despite the clear benefits from its adoption, many firms have faced challenges, be that legal, organisational, or business practices, when seeking to integrate data science within their business frameworks. The emergence of COVID-19, and the unprecedented disruption that has followed, has altered the incentives to invest in, and adopt these innovations. This has created challenges moving forward for certain applications because of factors such as supply-chain interruptions for necessary hardware (e.g., challenges in the education space moving online), lack of funds due to plunging profits and share prices (e.g., self-driving ride-sharing platforms), and access to skilled workers due to travel restrictions (e.g., due to the inability to have skilled labour immigrate and poor/blocked internet access from other jurisdiction).

For other firms, COVID-19 has created significant opportunities. Legislative lock-downs have driven sharp spikes in demand for products and services allowing for remote work and purchases. Review of privacy issues related to tracking apps, health care advances, remote work/studies and collection and use of data has been accelerated. Soaring share prices and record profits for some big tech companies during COVID-19 has also created opportunities for these companies to investment more in their R&D activities or purchase smaller start-ups. The rationale for this workshop is to bring together data science practitioners, policy makers and academics to share their experiences with, and insights on, these challenges.

2 WORKSHOP FORMAT

In this workshop, panel and audience members drew on their experiences to elaborate on the data science challenges firms and workers have encountered pre and post COVID-19. Most of the discussion and comments can be categorised within three themes: the effect of COVID-19 on: research and development investment; commercialization of new and existing technologies; and changes in the barriers to adoption across different areas.

Panel and audience members came from business, academia, and think-tanks. In the first half of the workshop the panel members discussed their own experience from prepared remarks and in the second half of the workshop audience members commented on and responded based on their own experience.

3 ORGANIZERS AND PARTICIPANTS

The organisers of the panel were: Michelle Alexopoulos (University of Toronto), Kelly Lyons (University of Toronto), Rohan Alexander (University of Toronto), Aije Egwaikhide (IBM), and Robert Blair Frost (University of Toronto). The organizers invited Robert Fay (CIGI), Alex Ince-Cushman, Alvin Francis (IBM), and Denise Almeida (UCL) to join Michelle Alexopoulos as panelists in the discussion of the evolving landscape in the Post-COVID-19 economy.

Dr. Michelle Alexopoulos is a Professor of Economics who is cross appointed to the Faculty of Information at the University of Toronto, a faculty affiliate of the Schwartz Reisman Centre for Technology and Society, and a Bank of Canada Fellow. Prof. Alexopoulos' recent research focuses on creating and analyzing new measures of technical change for economies. She is a member of the Productivity Partnership, and is a qualified legal expert in the fields of Technical change, Applied Econometrics and Macroeconomics.

Denise Almeida is a doctoral candidate at University College London whose main research interests are centred on privacy, change, ethics and AI, and algorithmic accountability, particularly around how these areas interact with different social, demographic and contextual factors. She is a senior privacy professional, currently working on user rights management and data protection at New Vector/Matrix.org.

Robert (Bob) Fay is managing director of digital economy at CIGI and responsible for its research direction and related activities. He has extensive experience in macro- and micro-economic research and policy analysis. Recently, he has engaged in research exploring how data has changed the economy and how COVID-19 has expedited the digital transformation. Prior to joining CIGI, Bob was a senior director at the Bank of Canada overseeing work to assess developments and implications arising from the digitization of the Canadian economy.

Alvin Francis is the Executive Director of Development, Cognos and Planning Analytics at IBM. He is an expert in statistical analysis, predictive analytics and machine learning. Prior to IBM, he spent several years in the telecommunications industry where he held various senior leader positions.

Dr Alex Ince-Cushman served as Chief Technology Officer at Just Energy from December 2018 to September 2020. Dr Ince-Cushman was an Executive with Palantir Technologies from March 2015 to November 2018, and was an Associate Partner a McKinsey & Company from November 2008 to February 2015.

4 OUTCOMES

In the first half of the workshop Kelly Lyons and Rohan Alexander asked the panel to provide their insights on answers to the following questions:

(1) Do you believe that COVID-19 has had a significant impact on research and development?
(2) What impact has COVID-19 had on the adoption and commercialization of new technologies in the field?
(3) What barriers do you think are currently hindering the adoption of AI and advances in data science? Have these barriers changed or become increasingly burdensome post COVID-19?
(4) What, if anything, can the government do to support research and development, adoption and commercialization for Canadian firms? Has COVID-19 affected your views on what the Government can or should do?

Each panelist provided initial opinions from prepared remarks. A summary of the discussion is outlined below.

Dr. Alexopoulos began the discussion by reviewing the economic changes that have occurred due to COVID-19, and discussed challenges in forecasting the future progression of economic growth, job creation and investment in digital economy. She then discussed the barriers to adoption that were identified in a related 2019 CASCON workshop [1]. Her comments were followed by comments from the other panelists. Each agreed that COVID-19 has dramatically impacted the economy and adoption patterns with changes in research and development funding across sectors differing in their magnitudes. Further they confirmed that many of the challenges pre-COVID-19 remain– issues with governance, outdated and fragmented legal frameworks governing data science, and access to the supply of skilled workers. However, the severity of the problems have been affected by COVID-19. Guaranteeing an adequate supply of qualified workers in the short run has become more challenging as Government policy has hampered international mobility and immigration patterns and COVID-19 has impacted educational attainment and enrollments. The panel also generally agreed that COVID-19 has exposed greater gaps in governance of new technologies, and illustrated factors such as privacy concerns, intellectual property protection, and the development of standards that need to be addressed to help adoption.

In the second half of the workshop, the organizers polled the audience to uncover (1) how COVID-19 had impacted the development and adoption of data science across their own industries and firms, and (2) how they ranked of the importance of the themes and broader concerns that the panelists were describing for the economy. The results were shared in a chat format to provide a starting point for the discussion. The panelists were invited to address comments relayed by the audience and the audience members were invited to further comment on the points discussed.

The panel and audience discussion provided important evidence to better understand the challenges of adopting data science in the post-COVID-19 economy. It also generated ideas for related future work.

REFERENCES

[1] Rohan Alexander, Kelly Lyons, Michelle Alexopoulos, and Lisa Austin. 2019. Workshop on barriers to data science adoption: why existing frameworks aren't working. In *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering*. 384–385.

4th Workshop on Advances in Open Runtimes and Cloud Performance Technologies

Daryl Maier
IBM Canada
Markham, ON, Canada
maier@ca.ibm.com

Vijay Sundaresan
IBM Canada
Markham, ON, Canada
vijaysun@ca.ibm.com

David Bremner
Faculty of Computer Science
University of New Brunswick
Fredericton, NB, Canada
bremner@unb.ca

ABSTRACT

Cloud services such as IBM Cloud or Amazon Web Services are increasingly becoming the environments where applications are developed, tested, and deployed, data gets stored, and businesses are run. Many of the features that define a cloud (e.g., resiliency, elasticity, consistency, security) are realized through runtime technologies. Clouds are polyglot environments, and therefore advances in cloud development are directly driven by innovation in runtime technologies. However, cloud environments pose unique and often conflicting demands on runtime systems that are generally less of a concern in isolated systems. Throughput performance (how many results can my application produce?), density (how many instances of my application can I create and run simultaneously in my provisioned environment?), startup performance (how quickly can I start a new instance of my application?), and language interoperability are all examples of important considerations that require innovative solutions.

Modern language runtimes are complex, dynamic systems that involve a myriad of components that must work cooperatively to achieve the functional and performance requirements of a given language. Typical core runtime technologies include dynamic just-in-time compilers for performance, garbage collection for heap management, platform abstraction for ease of portability to different hardware and operating system environments, test infrastructure for quality control, developer tooling for diagnosis and tuning of the various components, and interoperability between different language environments.

Cloud workloads are typically containerized and employ microservice and serverless architectures. Achieving peak performance in such environments requires careful tuning of the cloud services and applications in concert with the runtime system.

The goal of this workshop was to bring together research, industry, and developers from runtimes and cloud communities to share and discuss innovations, challenges, and research across a broad set of open source technologies (such as Eclipse OMR, Eclipse OpenJ9, Node.js, Open Liberty, Kubernetes) to improve performance in cloud environments. The focus on open solutions rather than proprietary was key as it allowed for greater collaboration amongst individuals, communities, researchers, and industry through shared learning on common technology.

This was a full-day virtual speaker session workshop where researchers, students, and practitioners presented their recent innovative work and findings. Topics discussed in the workshop included, but were not limited to:

- Open runtime technology frameworks;
- Compiler technology and innovative optimizations for dynamic cloud environments;
- Garbage collection and memory subsystem performance;
- Runtimes/Cloud cooperative tuning;
- Hardware techniques to assist runtime technologies;
- Dynamic languages for the cloud;
- Testing and correctness of runtime technology;
- Throughput and startup performance, and memory footprint reduction;
- Use of tools and infrastructure built on open technologies; and
- Innovative ways of exploiting open runtime technologies.

This workshop was successful in exploring a wide range of innovative runtime problems and solutions for the cloud environments. Many of the innovations were based on the open-source Eclipse OMR and Eclipse OpenJ9 projects. Eclipse OMR is a toolkit of language-agnostic runtime components that can be integrated in runtime environments to provide or extend the desired runtime features. The most popular components include garbage collection and compilation technologies as well as a common, portable interface for abstracting operating system functionality. Eclipse OpenJ9 is an open source, high-performance Java Virtual Machine that fully implements that Java Virtual Machine Specification and is used by several open source Java projects such as Open Liberty.

CASCON 2020, November 10-13, 2020, Toronto, Canada

Hands-on Workshop: Jumpstart your Application into a Reactive Event-centric World

Yee-Kang Chang
IBM
Markham ON Canada
yeekangc@ca.ibm.com

Meswan Bhaugeerutty
IBM
Markham ON Canada
meswan@ca.ibm.com

Gilbert Kwan
IBM
Markham ON Canada
gkwan@ca.ibm.com

Grace Jansen
IBM
Hursley, United Kingdom
grace.jansen1@ibm.com

ABSTRACT

We now live in a world with data at its heart. The amount of data being produced every day is growing exponentially and a large amount of this data is in the form of events. Whether it be updates from sensors, clicks on a website or even tweets, applications are bombarded with a never-ending stream of new events. So, how can we architect our applications to be more reactive and resilient to these fluctuating loads and better manage our thirst for data? In these quicklabs you'll learn what it means to easily build a cloud-native reactive application through the Eclipse MicroProfile reactive messaging framework and Apache Kafka.

2 Workshop Outline

This workshop covers the topics below:

- Creating reactive Java microservices
- Testing reactive Java microservices
- Acknowledging messages using MicroProfile Reactive Messaging
- Integrating RESTful services with a reactive system
- Consuming RESTful services using the reactive JAX-RS client
- Hands-on Labs

The sections that follow offer an introduction to the concepts and technologies covered in the workshop.

2 Microservices Architecture

Microservice architecture is a popular approach for building cloud-native applications so that each component is an individual service that fulfills a specific purpose. It enables small, autonomous teams to develop, deploy, and scale their respective services independently. One benefit is that the application can be scaled on a more granular level because each service is built and managed independently. The high-traffic services can be individually scaled to efficiently use resources rather than scaling up the entire system. Another benefit is that failures in one service can be isolated from the rest of the system; if a service fails, services that are independent are unaffected while dependent services can employ fault tolerance strategies to prevent the failure from cascading to other services.

Although a microservice architecture provides many benefits, it also introduces new challenges not apparent in traditional monolithic applications. Eclipse MicroProfile addresses these challenges so that you can easily develop cloud-native applications. These challenges include toleration of service failures, end-to-end security for an authenticated user request flowing through a set of microservices, and problem determination for requests spanning many services.

2 Eclipse MicroProfile

Eclipse MicroProfile is a modular set of technologies designed so that you can write cloud-native Java microservices.

Cloud-native is an industry-wide approach to developing and rapidly deploying applications to the cloud at scale. Cloud-native applications are designed around team-aligned microservices and developed by using agile practices and continuous integration/continuous delivery (CI/CD) to streamline deployment. With a range of vendors providing cloud platforms, open source and open standards are essential enablers for avoiding vendor lock-in.

MicroProfile enables you to develop and deploy cloud-native Java applications as loosely-coupled, lightweight services, each representing one unique business function. This approach is modular and makes the application easy to understand, easy to develop, easy to test, and easy to maintain.

2.1 MicroProfile layers of functionality

MicroProfile can be organized into three layers of functionality. The bottom layer represents REST services, the middle layer is for scaling towards hundreds of microservices, and the top layer contains tools to help you detect and diagnose issues.

- Open Tracing
- Metrics
- Health Check

- Open API
- Fault Tolerance
- JWT
- Config

- JSON-B/JSON-P
- Rest Client
- CDI
- JAX-RS

2.2 MicroProfile simplifies developing cloud-native Java microservices

The vast majority of cloud-native microservices are based on REST APIs, making the bottom layer the most essential. At its foundation, MicroProfile provides a set of technologies that make developing and using REST APIs easy. MicroProfile takes a small set of Java EE APIs: JAX-RS; CDI; JSON-B and JSON-P and augments them with a simple type-safe REST client API making it easy to consume REST services.

2.2.1 Build REST services. The JAX-RS, CDI, JSON-B and JSON-P Java EE technologies provide the base for MicroProfile. If you're new to Java EE and MicroProfile, this is a good place to start. JAX-RS is a Java API that allows you to build REST APIs by creating resource classes and adding appropriate annotations to create the necessary web endpoints. Context and Dependency Injection (CDI) provides objects with the dependencies that they need through the @Inject annotation rather than directly creating an object or finding them using a factory. JSON-P and JSON-B makes it easy to automatically serialize and deserialize classes to and from JSON.

2.2.2 Consuming a REST service with type-safe Java. MicroProfile Rest Client provides a type-safe approach for invoking REST services over HTTP. This API greatly simplifies the client-side API as defined by JAX-RS. MicroProfile Rest Client handles the communication between the client and service. You only need to define and annotate an interface that describes the actions that you need to perform on a REST resource. An implementation of this interface is automatically generated for you when CDI is used to inject your client into a dependent class.

2.3 MicroProfile simplifies scaling your organization

Handling hundreds of autonomous, collaborating and frequently evolving services introduces a number of new challenges. These challenges include, for example, documenting and sharing APIs across teams, propagating security across services, handling network or service failures, and continuously integrating and deploying service updates. Thankfully, the middle layer of MicroProfile features provide a number of APIs to simplify these tasks.

2.3.1 Document your REST APIs. MicroProfile OpenAPI provides a Java API for the OpenAPI specification that you can use to expose API documentation for your REST APIs. You can natively produce OpenAPI documents from your JAX-RS applications. OpenAPI is a standardization of the Swagger specification.

2.3.2 Handle unexpected failures in your microservices. MicroProfile Fault Tolerance provides an API and annotations for building robust behavior to cope with unexpected failures in the service you depend on. Aspects of fault-tolerance include timeouts, retries, fallbacks, bulkhead processing, and circuit breakers.

2.3.3 Authentication and role-based access control. MicroProfile JWT provides for interoperable authentication and role-based access control for your services. It allows for an authenticated JWT token to be shared across multiple microservices even if these services are running on multiple vendor implementations. It also allows for access to microservice operations to be controlled based on user and role information passed within the JWT token.

2.3.4 Externalize configuration to improve portability. MicroProfile Config externalizes configuration from the application to improve portability of the application. A core principle is to be able to override configuration at deployment time using system properties and environment variables. This means you can build your microservice once and deploy it many times through your CI/CD pipeline by changing the configuration for each deployment.

2.4 MicroProfile helps you detect and diagnose problems

Handling hundreds of microservices requires a strong operations focus. If the system is beginning to exhibit problems, how do you track down the root cause when a request might span tens or hundreds of services? How can you tell which service is not performing well, or understand the journey a request took through those microservices? The top layer of the MicroProfile feature set helps you answer these questions. It provides APIs to help you understand the health of services, how they're performing, and how requests are flowing through them.

2.4.1 Determine a microservice's availability. MicroProfile Health Check provides a common REST endpoint format to determine whether a microservice is healthy or not. Health can be determined by the service itself and might be based on the availability of necessary resources (for example, a database) and services. The service itself might be running but considered unhealthly if the things it requires for normal operation are unavailable. The Health Check endpoints are also designed to be easily integrated into Kubernetes liveness and readiness probes.

2.4.2 Monitor a microservice's telemetry data. MicroProfile Metrics provides common REST endpoints for monitoring the telemetry data of a running microservice, similar in nature to JMX but a much simpler API that uses JAX-RS. Both built-in and application-defined metrics are accessible, with the output in either JSON or Prometheus text formats. This API provides more extensive detail than the simple up and down reporting provided by MicroProfile Health.

2.4.3 Enable distributed tracing of your microservices. MicroProfile OpenTracing allows services to easily participate in a distributed tracing environment. OpenTracing defines behaviors and an API for accessing an OpenTracing-compliant Tracer object within your microservice. These trace logs can then be consumed by a third-party distributed tracing facility such as Zipkin or Jaeger.

3 Open Liberty

Open Liberty is an application runtime designed for the cloud. It's small, lightweight, and designed with modern cloud-native application development in mind. It supports the full MicroProfile, Jakarta EE and Java EE APIs and is composable, meaning that you can use only the features that you need, keeping the server lightweight, which is great for microservices. It also deploys to every major cloud platform, including Docker, Kubernetes, OpenShift and Cloud Foundry.

4 MicroShed Testing

MicroShed Testing offers a fast and simple way of writing and running true-to-production integration tests for Java microservice applications. MicroShed Testing exercises your containerized application from outside the container so you are testing the exact same image that runs in production.

MicroShed Testing aims to:

1. be easy to get started with
2. work with any Java EE, Jakarta EE or MicroProfile runtime
3. provide true-to-production tests

5 MicroProfile Reactive Messaging

MicroProfile Reactive Messaging provides an easy way to asynchronously send, receive, and process messages that are received as continuous streams of events. You simply annotate application beans' methods and Open Liberty converts the annotated methods to reactive streams-compatible publishers, subscribers, and processors and connects them up to each other. MicroProfile Reactive Messaging provides a Connector API so that your methods can be connected to external messaging systems that produce and consume the streams of events, such as Apache Kafka.

morPOP: A fast and granular agent-based model of COVID-19 to examine school mitigation strategies in Newfoundland & Labrador

Dionne M. Aleman
aleman@mie.utoronto.ca
University of Toronto
Toronto, Ontario, Canada

Benjamin Tham
benjamin.tham@mail.utoronto.ca
University of Toronto
Toronto, Ontario, Canada

Sean J. Wagner
wagnerse@ca.ibm.com
IBM
Toronto, Ontario, Canada

Justin Semelhago
justin.semelhago@mail.utoronto.ca
University of Toronto
Toronto, Ontario, Canada

Asghar Mohammadi
asghar.mohammadi@med.mun.ca
Memorial University
St. John's, Newfoundland and Labrador, Canada

Paul Price
pprice@mun.ca
Memorial University
St. John's, Newfoundland and Labrador, Canada

Jordan Bradfield
jordan.bradfield1@ibm.com
IBM
Halifax, Nova Scotia, Canada

Randy Giffen
randy_giffen@ca.ibm.com
IBM
Toronto, Ontario, Canada

Proton Rahman
prahman@mun.ca
Memorial University
St. John's, Newfoundland and Labrador, Canada

ABSTRACT

The Medical Operations Research Lab's Pandemic Outbreak Planner (morPOP) is an agent-based simulation of pandemic disease spread, wherein each individual in the population is an agent with unique demographic and comorbidity characteristics; for Newfoundland & Labrador, there were ≈520,000 agents. Individuals interacted in various environments, and each infectious contact increased probability of infection. Individuals potentially changed behaviors to self-isolate or seek medical care when symptomatic. By comparing the resulting disease spread across proposed school mitigation measures, including physical distancing and masks, effective policies were identified, allowing public health officials to make evidence-based decisions about appropriate measures to enact. The model was unique in its ability to capture such a large population while requiring ≈25 s computation time for a 100-day simulation on a single processor, thanks to implementation in C++ with various cost-saving techniques. Parallelization support was provided by the Center for Health Informatics and Analytics (CHIA) at Memorial University.

CCS CONCEPTS

• **Applied computing → Consumer health**; • **Computing methodologies → Massively parallel algorithms**; **Agent / discrete models**.

CASCON 2020, November 10-13, 2020, Toronto, Canada

KEYWORDS

COVID-19, agent-based simulation, pandemic disease spread, school mitigation strategies

1 INTRODUCTION

As COVID-19 has progressed throughout the world, it has become apparent that the high-level disease spread predictions provided by traditional epidemiological models, while fast to implement and requiring little data to gain useful high-level insights, are insuffi-cient to measure the impact of small-scale, nuanced public health interventions. Agent-based simulation (ABS) models, where each individual in the population is an agent with unique characteristics and behaviors, are better suited to this task. To measure the im-pact of school mitigation strategies in Newfoundland & Labrador (NL), we adapted the existing morPOP (Medical Operations Re-search Lab's Pandemic Outbreak Planner) ABS [1] to the specific nature of the NL population and COVID known disease properties. The school mitigation strategies tested included various levels of physical distancing and mask usage.

2 METHODS

2.1 morPOP agent-based simulation model

morPOP modeled each individual in the population as a unique entity, representing individual behavioral, demographic, and health

characteristics. Unique environments where people interact are represented: households, hospitals, schools, businesses, etc. Each individual had an infection status following the typical SIRD (susceptible, infectious, recovered, dead) model, with a five-day latent pre-symptomatic infectious period after initial infection. Forty-two percent of cases were assigned to be asymptomatic [4]. It was assumed that 90% of individuals would self-isolate upon symptom onset or when a household member becomes symptomatic.

Individuals followed behavior patterns like being at home for a certain number of hours per day, going to work or school, observing physical distancing guidelines or not, and seeking medical care when infected. At each location visited, there was interaction with other individuals at the same location (household members, fellow students, fellow employees, etc.), and an individual's chance of becoming infected was determined by contact with infected individuals in each location and the nature of that contact. Per-minute disease transmission rates between age groups were adapted from pandemic influenza rates [3], and contact transmission rates within workplaces were adjusted by the Vancouver School of Economic risk factors [5]. Since the original publication of morPOP [1], the model was updated so that infection probabilities are calculated in a multiplicative fashion, rather than additive.

2.1.1 Implementation. Unlike most ABSs that struggle to capture more than ≈100,000 agents or that require lengthy computation times and memory to perform less-detailed simulation, morPOP was written in C++ for computational speed and parallelization. Using a number of coding approaches specifically designed to speed up run times, morPOP simulated a 100-day outbreak on the NL population of ≈ 520, 000 agents in ≈25 s with < 1 GB RAM; for the Greater Toronto Area (Ontario) for which morPOP was originally designed, the simulation time for ≈6 million agents was < 1 min with 4 GB RAM. The most impactful coding decisions to achieve this performance were the use of arrays instead vectors and the ordering of the population array such that all susceptibles and infecteds are sequentially ordered, eliminating the need to unnecessarily loop over recovered or dead individuals in updating statuses, yielding a complexity improvement from $O(n^2)$ to $O(n \log n)+|S|+|I|$, where S and I are the sets of susceptible and infectious people, respectively.

The model was parallelized with one simulation per processor and implemented on high-performance computing infrastructure provided by the Center for Health Informatics and Analytics (CHIA) at Memorial University. The specific infrastructure used was three Linux nodes, each with 32 cores (64 threads) and 256 GB RAM.

Daily SIRD counts for the population and census sub-divisions were outputted from the model as `.csv` files, and were uploaded to a custom-built web interface for high-level analysis.

2.1.2 Limitations. The benefit of such a granular ABS was the ability to simulate complex populations and public policies. The drawback is that it relied heavily on data that is may not be available, and may therefore not have accurately modeled certain aspects of reality. Thus, ABS models are best used as "what-if" machines to compare different scenarios rather than as crystal ball predictors of exact pandemic outcomes. Specific limitations of the current implementation of morPOP for NL-COVID is that there was no contract tracing in this analysis (resulting in overestimation of infections), and no imported infections (underestimation).

2.2 School mitigation policies

School mitigation policies included 1 m physical distancing (79.7% reduction in transmission probability [2]), 2 m physical distancing (estimated 90% reduction), and an imperfect distancing resulting in 50% reduction. Mandatory masks for senior and intermediate students (82% reduction [2]) with varying levels of effectiveness of 20%, 40%, and 60% reduction were tested, with and without physical distancing measures.

3 RESULTS

Schools were a primary driver of infections. Physical distancing measures provided significant improvement in the infection curve in all scenarios, but the difference between 1 m and 2 m distance was insignificant. Distancing alone was more effective than masks alone, since masks were only applied to intermediate and senior students. As an example, Figure 1 illustrates distancing with fully effective (82% reduction) and half effective (40% reduction) masks.

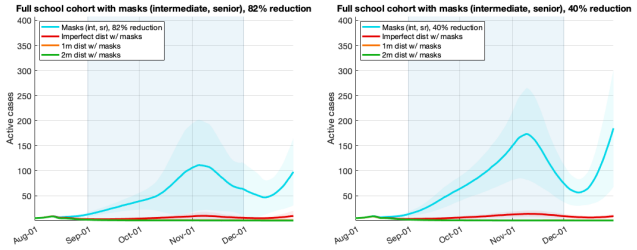


Figure 1: Distancing with fully effective (82% reduction) and half effective (40% reduction) mandatory masks for intermediate and senior students

4 CONCLUSION

Any one mitigation measure will likely dramatically reduce the number of COVID cases, and any combination of distancing and mandatory masks, even if imperfectly executed, is highly effective. All scenarios with any level of distancing are likely within NL Public Health's ability to control and possibly prevent, though such an assumption relies on fast detection of cases.

REFERENCES

[1] D.M. Aleman, T.G. Wibisono, and B. Schwartz. 2011. A nonhomogeneous mixing model for predicting pandemic disease spread. *Interfaces Special Issue on Humanitarian Applications: Doing Good with OR* 41, 3 (2011), 301–315.

[2] D.K. Chu, E.A. Akl, S. Duda, K. Solo, S. Yaacoub, and H.J. Schünemann. 2020. Physical distancing, face masks, and eye protection to prevent person-to-person transmission of SARS-CoV-2 and COVID-19: a systematic review and meta-analysis. *The Lancet* (2020).

[3] M.J. Haber, D. K. Shay, X.M. Davis, R. Patel, X. Jin, E. Weintraub, E. Orenstein, and W.W. Thompson. 2007. Effectiveness of Interventions to Reduce Contact Rates during a Simulated Influenza Pandemic. *Emerging Infectious Disease* 13, 4 (2007), 581–589.

[4] E. Lavezzo, E. Franchin, C. Ciavarella, G. Cuomo-Dannenburg, L. Barzon, C. Del Vecchio, L. Rossi, R. Manganelli, A. Loregian, N. Navarin, and D. Abate. 2020. Suppression of a SARS-CoV-2 outbreak in the Italian municipality of Vo'. *Nature* (2020).

[5] Vancouver School of Economics. 2020. Employment and COVID-19 Transmission Risks. https://lmic-cimt.shinyapps.io/vse_lmic_risk_analysis/. Accessed July 8, 2020.

Novel Hardware & Software Design for Mathematical and AI Acceleration

Robert F. Enenkel
enenkel@ca.ibm.com
IBM Canada Ltd.
Markham, Ontario, Canada

Silvia M. Müller
smm@de.ibm.com
IBM Systems
Böblingen, Germany

Christopher K. Anand
anandc@mcmaster.ca
McMaster University
Hamilton, Ontario, Canada

ABSTRACT

Special-purpose computational hardware, in integer and floating-point arithmetic units, as well as in memory systems, provides opportunities to accelerate a wide range of mathematical applications, including medical imaging, digital signal processing, artificial intelligence, and cryptography.

In this workshop, university and IBM speakers provided insight into a selection of novel uses of computer system design to accelerate important applications.

CCS CONCEPTS

• **Computer systems organization → Architectures**; • **Applied computing → Physical sciences and engineering**; • **Mathematics of computing → Mathematical software**.

KEYWORDS

mathematical functions, polynomial approximation, linear algebra, matrix operations, computer arithmetic, floating- point arithmetic, integer arithmetic, modular arithmetic, processor hardware design, processor architecture, system design, hardware acceleration, cryptography, artificial intelligence, neural networks, computational memory, phase change memory

ACM Reference Format:

Robert F. Enenkel, Silvia M. Müller, and Christopher K. Anand. 2020. Novel Hardware & Software Design for Mathematical and AI Acceleration. In *EVOKE CASCON 2020: Conference of the Centre for Advanced Studies on Collaborative Research.* IBM Corp., Riverton, NJ, USA, 10 pages.

1 PRESENTATIONS

High-Performance Matrix Math in IBM POWER Processors

Jose Moreira, IBM Research, Yorktown Heights

Power ISA processors have a long history of offering superior features for HPC applications and the Open Power ISA has enabled open access to many of these features. IBM's most recent contribution to Open Power ISA, in the form of Power ISA Version 3.1, includes the Matrix Math Assist (MMA) instructions. The MMA instructions are designed to deliver enhanced performance and efficiency for both classical high-performance computing, in the space of scientific and technical computing, and for the increasingly important space of business analytics. They offer increased computational intensity by implementing BLAS 2- and BLAS 3-level functionality, performing more operations per data element. Our goal is to raise awareness of and interest in these new HPC features, which we believe can lead to further research in processor architecture and programming environments. Some of the most promising application areas include graph algorithms, classical machine learning and deep learning.

Computational Phase-Change Memory for Neural-Network Inference

TBD, IBM Research, Zürich

HashedExpression: A Tool for High-Level Optimization of Neural Networks

Nhan Thai, *Curtis d'Alves*, *Christopher Anand*, McMaster University

We introduce a tool for symbolic code generation which we propose to use for high-level transformation of loss functions associated with neural networks (NNs) and symbolic differentiation. HashedExpression is a library for modelling and solving optimization problems. It places great emphasis on modelling correctness and fast, low-level code generation. Empowered by the purely functional programming language Haskell, HashedExpression provides users with a type-safe, correct by construction APIs for modelling optimization problems where invalid models will result in type errors. In the case of NNs, this ensures that vector shapes match. Transformations are supported by a simple, extensible pattern-matching language which makes it easy to add optimizations specific to a domain, such as particular neural networks. Currently, HashedExpression generates C code that can be compiled and linked with fast optimization solvers such as Ipopt, libLBFGS, or L-BFGS-B. It's design envisages future work on advanced parallelization, targeting SIMD and multi-core CPUs and GPUs, as well as future hardware acceleration.

Modular arithmetic engine for elliptic curve cryptography on the IBM z15 processor

Silvia M. Müller, IBM Systems, Böblingen

CASCON 2020, November 10-13, 2020, Toronto, Canada

2 BIOS

Silvia M. Müller

Silvia, an IBM Distinguished Engineer, is leading the development of competitive arithmetical units for POWER and z Systems. She has been driving a shift in IBM's Systems value towards stack solutions based on hardware/software co-design and co-optimizations, specialized arithmetic accelerator engines and hardware differentiation. Prior to joining IBM in 1999, she was a professor for processor arithmetic and processor design at the University of Saarland, Germany. She holds a Ph.D. in computer Science (1991) and an MSC in Mathematics (1989) from the same university. She authored over 150 issued patents, 3 books, and over 30 papers.

Jose Moreira

José E. Moreira is a Distinguished Research Staff Member in the Scalable Systems Department at the Thomas J. Watson Research Center. He received a B.S. degree in physics and B.S. and M.S. degrees in electrical engineering from the University of Sao Paulo, Brazil, in 1987, 1988, and 1990, respectively. He also received a Ph.D. degree in electrical engineering from the University of Illinois at Urbana-Champaign in 1995. Since joining IBM at the Thomas J. Watson Research Center, he has worked on a variety of high-performance computing projects. He was system software architect for the Blue Gene/L supercomputer and chief architect of the Commercial Scale Out project. He currently leads the IBM Research work on the architecture of POWER processor. He is an author or coauthor of over 100 technical papers and 15 US patents. Dr. Moreira is a Senior Member of the IEEE (Institute of Electrical and Electronics Engineers) and a Distinguished Scientist of the ACM (Association for Computing Machinery).

Christopher Kumar Anand

Christopher Kumar Anand is an Associate Professor of Computing and Software at McMaster University and Chief Science Officer of Optimal Computational Algorithms, Inc., innovating in Computer Science Education (drawing from academic disciplines form cognitive science to programming language theory) and computer architecture, and applying optimization (to instruction scheduling, electron microscopy, magnetic resonance imaging and machine learning). Prof. Anand is also a Faculty Fellow of the IBM Center for Advanced Studies.

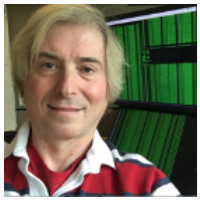
Nhan Thai

Nhan is an MSc (Computer Science) student at McMaster University. He received his Bachelor at Hanoi University of Science and Technology where he wrote his thesis on routing algorithms in wireless sensor networks. He is a functional programming enthusiast, using Haskell for most of his research work on optimization and code graph transformations. He is also an open source advocate who authored and contributed to many projects, viz. https://github.com/dandoh. You can reach him at `thain1@mcmaster.ca`

Curtis d'Alves

Curtis is a Ph.D candidate at McMaster University, and IBM CAS Student Fellow of the year (2019). His research involves compiler optimization (in particular instruction scheduling) and continuous optimization algorithms. His work optimizing libraries for accelerating math functions has been successfully productized in releases of IBM MASS libraries. You can reach him at `dalvescb@mcmaster.ca`.

Robert F. Enenkel



Robert is a technical leader for mathematical libraries and numerical computing in the compiler group at the IBM Toronto Lab, where he works on the development of high-performance mathematical function libraries and the performance exploitation of IBM processors through compilers. Since joining IBM in 1998, he also spent 6 years as a Research Staff Member in the IBM Center for Advanced Studies, contributed to the organization of multiple CASCON conferences, and served as Program Co-Chair for CASCON x EVOKE 2019. He has a PhD and MSc in mathematics and computer science from the University of Toronto, and has authored or co-authored 13 patents and over 30 papers and technical articles.

ACKNOWLEDGMENTS

The authors would like to thank NSERC and the IBM Centre for Advanced Studies for financial support.

Z Modernization Open Tools Showcase

Steve Shao Software Developer Debug for z/OS
IBM Systems
steve.shao@ibm.com

Nitika Sharma
Software Developer
Debug for z/OS IBM Systems
n.sharma@ibm.com

Stephanie Kuan
Software Developer
Debug for z/OS IBM Systems
sbagot@ca.ibm.com

ABSTRACT

Z Modernization Open Tools Showcase is a collection of multiple demos and use-cases built into a single environment, illustrating how to configure and debug transactions in seconds. What's more exciting, it brings support to developers first choice IDEs including VS Code. With modern tools to existing z/OS workloads, developers can start interacting with z/OS like we would any other cloud environment.

Keywords

Mainframe, z/OS, Debugger, VS Code

1. INTRODUCTION

Mainframes are data servers designed to process up to 1 trillion web transactions daily with the highest levels of security and reliability. Mainframes are found in 92 of world's top 100 banks, handle 87% of all credit card transactions and nearly $8 trillion payments.

Traditionally, developers leverage emulators or eclipse IDE when debugging transactions running on Z. Nowadays, moving to the cloud gives everyone access to enterprise-class technology. It also allows smaller businesses to act faster than big established competitors. Additionally, increasing popularity of cloud services leads to the rise of cloud IDEs.

Z Modernization Open Tools bring COBOL and PL/I applications debugging support multiple modern IDEs including VSCODE, Open Shift, Thiea and many other cloud IDEs. It provides developers a modern debugging experience for IBM Z Enterprise Languages.

2. Background

2.1 What is Mainframe?

At their core, mainframes are high-performance computers with large amounts of memory and processors that process billions of simple calculations and transactions in real time. The mainframe is critical to commercial databases, transaction servers, and applications that require high resiliency, security, and agility.

Today's mainframes are much smaller than the early "Big Iron" machines. With a standard 19" rack, the latest mainframe seamlessly coexists with other platforms in the data center. One IBM z15™ single-frame system requires 75 percent less floor space than x86 2U servers running the same workloads and throughput – and reduces power consumption by 40 percent.

What happens in a second? There are 7812 tweets, 15,650 posts on Facebook, 63,386 google searches and 71,381 views on YouTube. On mainframe, 1,157,407 CICS Transactions are successfully executed every second.

CASCON 2020, November 10-13, 2020, Toronto, Canada

Figure 1. IBM z15 single-frame or multi-frame with one frame

2.2 Old-School Debugging Methodology

The most significant difference is that an IDE allows you to compile and run the code. Further, some IDEs have advanced features like debug.

In the traditional way, developers use ISPF emulators or eclipse IDEs to debug their transactions. Without support of modern IT tools like git or Kubernetes, developers should know more than about COBOL, PL/I and Assembler.

3. Challenges

Modernize the IBM z/OS Debugger by providing a more modern, familiar interface is not an easy job to accomplish. Our new methodology must not only support debugging cloud IDEs, but also compatible to existing eclipse IDEs.

3.1 Change with debug protocols

3.1.1 Debug Adapter Protocol

Adding a debugger for a new language to an IDE or editor is not only a significant effort, but it is also frustrating that this effort cannot be easily amortized over multiple development tools, as each tool uses different APIs for implementing the same feature.

The idea behind the Debug Adapter Protocol (DAP) is to abstract the way how the debugging support of development tools communicates with debuggers or runtimes into a protocol. Since it is unrealistic to assume that existing debuggers or runtimes adopt this protocol any time soon, we rather assume that an intermediary component - a so called Debug Adapter - adapts an existing debugger or runtime to the Debug Adapter Protocol.

The Debug Adapter Protocol makes it possible to implement a generic debugger for a development tool that can communicate with different debuggers via Debug Adapters. And Debug Adapters can be re-used across multiple development tools which

significantly reduces the effort to support a new debugger in different tools. The Debug Adapter Protocol is a win for both debugger providers and tooling vendors!

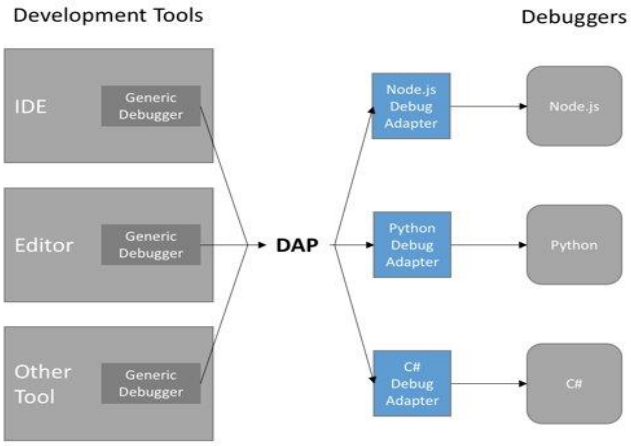


Figure 2. DAP with different debuggers

3.1.2 Shift from EDPC to DAP

Old IDE like IDz interacts with debug engine through EPDC protocol. To add support for DAP, we implement a new layer above current structure to translate data between engine and new IDE side.

3.2 New debug profile API

Old debug profile API does not have good support in terms of data integrity, speed and security. Define a new Debug profile API handles general user request is necessary. On the mainframe side, we set up a new profile service that help developers configure debug profiles in seconds.

3.3 Adapt to different environments

Our goal is to provide modern debugging interface to "Deb" persona to debug on z/OS. Primarily we have support for vs code in the first release. Recently we add support for Theia. As our journey to cloud moving, we embraced cloud environments like Red Hat open shift. Integrated with Red Hat Code Ready Workspaces, IBM Z Open debug could provide a modern IT experience with mainframe.

Built on the open Eclipse Che project, Red Hat Code Ready Workspaces uses Kubernetes and containers to provide any member of the development or IT team with a consistent, secure, and zero-configuration development environment. The user experience is as fast and familiar as an integrated development environment (IDE) on their laptop.

4. A Collaborative Approach

In the figure 3 below, we illustrate the architecture of a collaborative approach for IBM Z Open Debug. Remote debug service is acting as a translator for VS Code and debug engine by converting EPDC protocol to DAP. Inside VS Code, we have two extensions running on top of that. Profile UI extension provides an interface for developers better managing their debug profiles. The other extension provides interactive debugging support for debugging z/OS COBOL and PL/I applications, in conjunction with IBM z/OS Debugger.

On the other hand, we could also find the existing architecture of eclipse IDEs and also how it interacts with debug engine running on z/OS.

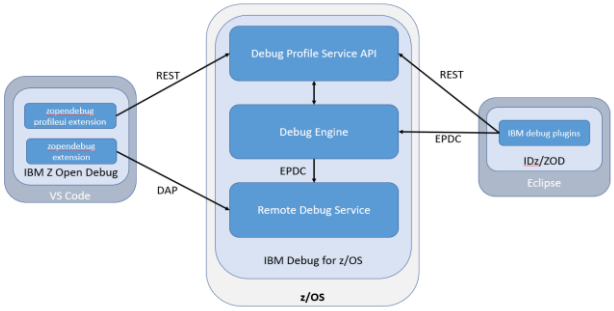


Figure 3. Collaborative Approach Architecture

5. Future Work

In the future, we will continue work on providing tools for IBM z/OS debugger by providing a more modern, familiar interface. Stay our journey to the cloud, IBM z/OS debugger can help you on your way to cloud-native infrastructure, with end-to-end coverage that allows for complete, holistic transformation.

6. ACKNOWLEDGMENTS

Our thanks to IBM Cascon for allowing us to provide us an opportunity to participate in this event.

7. REFERENCES

[1] https://developer.ibm.com/mainframe/2020/06/12/introducing-ibm-z-open-debug/

[2] https://developer.ibm.com/mainframe/2020/06/12/vscode-zopendebug-profile-view/

[3] Debug Adapter Protocol: https://microsoft.github.io/debug-adapter-protocol/

Smart Cities with Smart AI to fight back Covid19

How smart cities used smart AI to fight back the pandemic

Hina Sharma
IBM Cloud and Cognitive
IBM India Pvt Ltd, Pune MH India
hinsha26@in.ibm.com

ABSTRACT

As per the latest statistics (as of Sept 25,2020), total Covid cases have risen to 33.2 M with 1M lives being lost. The fear caused by this global outbreak has stranded the world. Conventional education system and work environments were now challenging but the show had to go on.

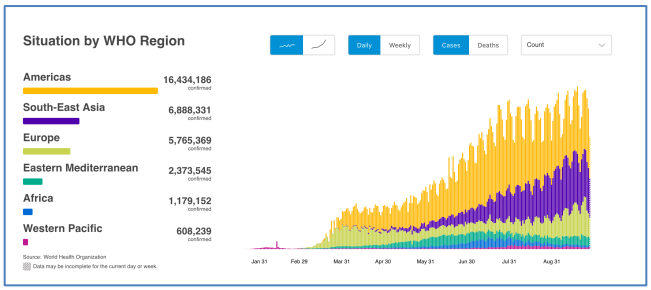
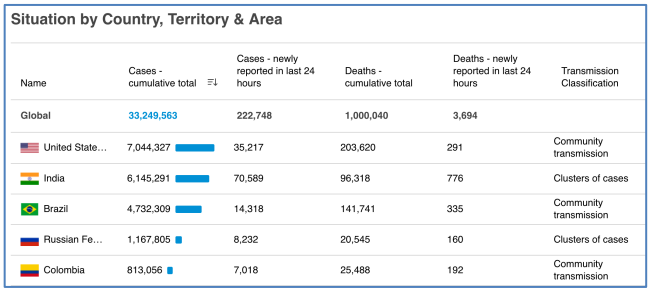


Figure 1: WHO-Region data representation of the current situation of Covid-19 across the world

With a sudden surge of requirement for online education, e-commerce and work from home, IT had a big role to play. While we had the doctors and policemen at fore front, IT was the backbone of all the systems up and running.
Cloud technology has had a major role in making sure that the world continues even from the comfort of our homes. Cloud migration became the no. 1 priority even for companies which were not planning migration. Businesses were now looking for resiliency, High Availability and Load balancing for a seamless customer experience.

Amongst all this chaos and complexity of managing work and education, countries and cities also had to control the spread of the deadly virus. With number of infected increasing on the graph, it became the focus area for the government to control the spread.

CASCON 2020, November 10-13, 2020, Toronto, Canada


Situation by Country, Territory & Area

Name	Cases - cumulative total	Cases - newly reported in last 24 hours	Deaths - cumulative total	Deaths - newly reported in last 24 hours	Transmission Classification
Global	33,249,563	222,748	1,000,040	3,694	
United State…	7,044,327	35,217	203,620	291	Community transmission
India	6,145,291	70,589	96,318	776	Clusters of cases
Brazil	4,732,309	14,318	141,741	335	Community transmission
Russian Fe…	1,167,805	8,232	20,545	160	Clusters of cases
Colombia	813,056	7,018	25,488	192	Community transmission

Figure 2: WHO Dashboard latest covid data

Major issues faced by countries during the pandemic:

1. Maintaining social distance to limit the contact with others.
2. With the number of infected cases increasing, hospitals were short of beds, oxygen and other resources.
3. Since doctors, policemen had to be at the forefront too were at the risk of getting infected.
4. Contamination and spread of virus at public places like hospitals, stations, airports etc.
5. Surface contamination at such public places where the infected person might have touched.

CCS CONCEPTS

• Artificial Intelligence • Cloud

KEYWORDS

Smart City, Artificial Intelligence, Cloud, covid19, innovations

1 The Smart Cities

Smart cities are the prepared cities, are the cities which have equipped themselves with the right innovations and have used technology to their benefit. Smart cities know how to fight the enemy with the right set of weapons. Initially, when the world was ignorant of what is waiting for it next, BlueDot, a Canadian startup and AWS customer was the first to raise an alert - using their machine learning algorithms. BlueDot helps government officials, airlines and hospitals to help them anticipate and better manage the risks.

1.1 Smart cities leveraged Smart AI

Different countries took different measures to ensure the safety of their area. Different rules were enforced to ensure the safety of its citizens.

1. Total lockdown of the entire country to limit the movement of people and limit contacts with each other.
2. Wearing masks has been mandated ever since.
3. Partial lockdown stays to limit people movement on roads and non-essential movement.

Although, every country is working towards the same cause and prevention, what would make a country smart is the way it handles the current times. How automated the country is in terms of its operations? How well the country or its city is using technology and especially Artificial Intelligence to fight the pandemic. In other words, How Smart is the city?

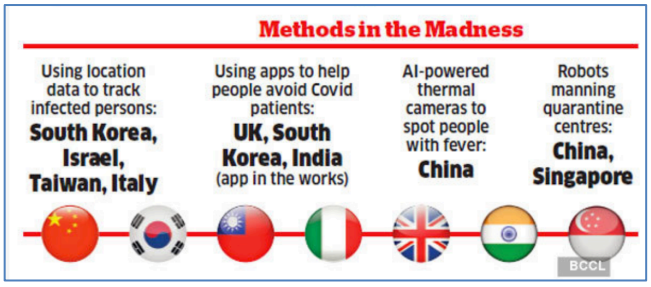


Figure 3: Countries using smart AI innovations. Courtesy: www.economictimes.com

Artificial Intelligence helps to collect and analyze data. It can help to predict what is next. It can help take preventive measures. AI in hospitals can help predict bed availability, help doctors analyze patient and his history quickly. So, let's discuss some of the great innovations have made a great difference in the way we are fighting this enemy. Let's learn what makes a city a smart city. And how smart cities are leveraging AI to be at the fore front of winning over these tough times:

1.1.1 Better Patient Management: Cities are using AI + IOT to better manage the patients in hospitals. Physicians are using AI technologies to detect the symptoms of covid-19 at an earlier stage. Early analysis of the infection and suggest therapies. Even the treatment is based as suggested by AI and further monitoring too. Better bed management in different hospitals in the city, can help patients find the availability at the earliest and save the previous time. The Citizen Covid19 Risk Self-Assessment app is being used by Agra city in India as part of smart city initiative.

.

1.1.2 Sanitize your way: Cities like Varanasi, India are using smart sanitizers. These drone-based sanitizers spray the sanitizers in covid-19 prone sensitive parts. Companies like Intel in collaboration with AI enabled organizations are coming up with AI enabled masks.

1.1.3 Contact *Tracing:*

Countries like China have effectively embraced AI and related technology to enable contact tracing applications. Automated robots are placed at hospitals to take care of the patients. This helps avoid contact of nurses with patients. Other countries have followed the suite too. These contact tracing applications have successfully helped the countries and its cities to track real-time contact tracing and help the virus not to spread. India too has Arogya-Setu app in India is the contact tracing app, which is being used by government to track its covid-19 infected citizens.

1.1.4 Emergency Service Response*:*

Cities are collaborating across to monitor the evolution of the diseases, sensitive areas in the cities, expect the behaviour of the citizen, and help manage public health emergencies.

1.1.4 Let's learn the smart way: AI innovations are being extended to online education. Using IBM Watson, we can have several innovations in online education too. Convert the teaching audio to notes for later reference, automated attendance system. Understand how the student is responding to the lectures. Countries are adopting these innovations to make sure that its citizens progress well and nothing stops them from learning, well not even the pandemic.

The above are just a few examples how smart cities can utilize AI to their benefit and prevent the spread of covid-19. This is just the start of the next revolution. Governments are now spending huge amounts to make their cities smarter cities. Smarter cities will be the connected cities. Connected cities will collect data, and this data will be utilized to find the reasons and thereby control this virus.

Smart city technologies have the power to improve the health and well-being of its people and the city as a whole. New

avenues for economic development in every domain - Transport, automobiles, health, environment, efficient energy usage. Only with the help of AI, IOT can we fight the biggest challenge that confronts us. the potential of technology has to be used at the best possible, and only then A Smart City with Smart AI can fight back Covid19.

Countries are empowering themselves with new innovations. Edge computing, Blockchain integrated with AI is doing wonders. Data analysis shows the strategic and financial gains to companies with technology as their friend. Businesses along with its city and country would innovate and that would be their only key to recovery during these tough times.

ACKNOWLEDGMENTS

The author thanks everyone who have helped provide the required information related to the smart cities and their smart innovations.

REFERENCES

[1] www.ipwatchdog.
[2] www.economictimes.com
[3] www.smartcities.gov.in

Quantum Computing: Synergies and Opportunities

Mehdi Bozzo-Rey
mehdi.bozzorey@cambridgequantum.com
Cambridge Quantum Computing Ltd.
Toronto, Ontario, Canada

Robert Loredo
loredo@us.ibm.com
IBM Quantum
Miami, Florida, USA

Hausi A. Müller
hausi@uvic.ca
University of Victoria
Victoria, British Columbia, Canada

Ulrike Stege
ustege@uvic.ca
University of Victoria
Victoria, British Columbia, Canada

ABSTRACT

Quantum computing has evolved from a field of scientific research to a quantum technology industry. Much progress is still needed to solve real-world problems with quantum technology and achieve quantum advantage. Industries, governments, and universities are experimenting with advanced quantum computing technologies to become quantum ready. One way forward is to combine quantum and classical approaches to form hybrid models, algorithms, and architectures to overcome the limitations of NISQ systems for near-term quantum computations. CASCON 2020 features a 2-day quantum computing workshop. This workshop discusses synergies and challenges in workforce training and opportunities in quantum computing applications featuring speakers from IBM, start-up companies, and academic research programs. The key goal of this workshop is to discuss synergies and opportunities in quantum computing along three dimensions:

- Existing and emerging quantum ecosystems across Canada
- Hybrid quantum-classical problem-solving and applications
- Vibrant quantum start-up companies

CCS CONCEPTS

• **Computer systems organization → Architectures**; • **Applied computing → Emerging technologies**; **Physical sciences and engineering**; • **Software and its engineering → Software notations and tools**; • **Social and professional topics → Computing education**; **Computing industry**; **Computing profession**; **Computing and business**.

KEYWORDS

Quantum computing, computing with nature, engagement, education, training, algorithms, programming, Qiskit, IBM Quantum, hybrid computing, applications, chemistry, finance, cryptography, machine learning, quantum start-ups

CASCON 2020, November 10-13, 2020, Toronto, Canada

1 INTRODUCTION

The field of quantum computing has seen many developments since Google's so called "quantum supremacy article" in 2019 [10]. We witnessed advances in hardware, both in terms of number of qubits and quality—reflected by the *quantum volume* metric [13], now used by hardware vendors to reflect the overall quality of their devices. IBM just released a roadmap that will take them from the noisy, intermediate-scale devices of today to the million-plus qubit devices of the future with the 1,000-plus qubit IBM Quantum Condor device for 2023 [16]. Honeywell has announced hardware releases based on their ion trap technologies with a quantum volume of 64. The overall quantum hardware ecosystems are blooming and promise a wider choice of technologies for end users soon [1, 19]. From an accessibility and delivery point of view, many cloud vendors can now be considered quantum cloud providers [17, 21, 22]; even startups, such as Xanadu [24], provide cloud access to their own devices. The software ecosystems bloomed vigorously with the emergence of the concept of quantum workflows with the release of Zapata's Orquestra [11]. On the full-stack front, Amazon released its first version of Amazon Braket for customers to access QPUs from select vendors. From a circuit optimization and noise mitigation point of view the race is still on. This is essential for getting the most from actual NISQ devices—with many startups involved [2, 6–8]. Language translation and the ability to access hardware devices outside a specific ecosystem while still using high-level quantum libraries and keeping the benefits of targeted circuit optimization, becomes critical for rapid prototyping and testing as pioneered by CQC [23].

2 EXISTING AND EMERGING QUANTUM ECOSYSTEM ACROSS CANADA

The first day of the workshop presents the Canadian quantum ecosystems and ongoing quantum education initiatives. Decades of research investments have put Canada at the global forefront of the Quantum Industry. In a study by McKinsey, published in The Economist in 2017, Canada is ranked 1st among G7 nations in per-capita spending on quantum research, 5th in the world in

total expenditure on quantum science, and 1st in the world in quantum computing science [12, 18]. Canadian research contributions to the highly interdisciplinary field of quantum are numerous—from the first quantum cryptography protocol (BB84) by Gilles Brassard (McGill) and Charles Bennett (IBM)—to Donna Strickland's (Waterloo) and Gérard Mourou's (France) Nobel Prize in Physics in 2018. These contributions were possible with strong and well targeted federal and provincial government initiatives to fuel not only basic research but also strengthen industry-academic collaboration. For example, Canada First Research Excellence Fund (CFREF) is"helping Canadian postsecondary institutions excel in research areas that will create long term economic advantages." Moreover, three major quantum institutes have been created: Institut Quantique (IQ)[1], Transformative Quantum Technologies (TQT)[2], and Stewart Blusson Quantum Matter Institute (SBQMI). Canadian provinces also funded platforms to strengthen the collaboration between academia and industry, with the Espace IBM Quantum in Québec and the British Columbia Quantum Algorithms Institute. Other investment in quantum technologies research and development came directly from the private sector, with private donations to support the creation and ongoing operations of the Institute of Quantum Computing (IQC) in Waterloo [14].

3 TOWARD PRACTICAL QUANTUM APPLICATIONS USING HYBRID PROBLEM SOLVING TECHNIQUES

Part of this workshop is dedicated to hardware and software providers, exploring two emerging fields in Quantum Information Science (QIS): Quantum Machine Learning (QML) and Quantum Natural Language Processing (QNLP). Today's quantum algorithms are generally limited to shallow-depth circuits due to noise and other environmental effects caused by the nature of current quantum devices. These algorithms are developed as hybrids to leverage the capabilities of near-term quantum computers. Variational techniques are some of the most promising hybrid approaches. By allowing classical and quantum systems to work together, the classical system can provide the QPU with parameters generated in a previous iteration.

Examples include: VQE (Variational Quantum Eigensolver is used to approximate the lowest energy level of a given Hamiltonian [20]), QAOA (Quantum Approximate Optimization Algorithm is a technique mostly used for Combinatorial Optimization problems [15]), QNNs (Quantum Neural Networks are quantum analogues or generalizations of classical neural nets [4]), VQLS (Variational Quantum Linear Solver is used to solve systems of linear equations [9]).

4 A VIBRANT QUANTUM STARTUP ECOSYSTEM

The last part of the workshop is dedicated to realm of startups. Hardware vendors or cloud providers have existing non-quantum programs dedicated to startups. Microsoft and IBM have been spearheading dedicated quantum startup programs over the past three years. The diversity of quantum startups is impressive globally and is rooted in Canada with the Creative Destruction Labs (CDL) quantum stream offered in Toronto [3]. The quantum stream brings together entrepreneurs, scientists, investors and hardware vendors to build ventures in quantum computing and other applications of quantum technologies. The program has been extremely successful so far with startups from Canada, US, Europe and Singapore attending. Some of them opened offices in Canada (e.g., Multiverse Computing), others built international partnerships (e.g., Miraex [5]) With advances in QML and increasing interest from the high performance computing community, it is expected to see ventures leveraging both the AI and quantum communities and startups paving the way for hybrid solutions and transitioning from classical to quantum platforms.

[1]Université de Sherbrooke, Québec
[2]University of Waterloo, Ontario

5 CONCLUSIONS

This 2-day CASCON x EVOKE 2020 quantum computing track is a great opportunity to engage in emerging quantum ecosystems, and to get an overview of Canadian initiatives. It provides ample opportunity to network and explore partnerships, to discuss challenges and opportunities with quantum researchers, scientists, engineers, entrepreneurs, developers, students, practitioners, educators, programmers, enthusiasts, and newcomers.

ACKNOWLEDGMENTS

This work was supported in part by IBM CAS, Canada, Cambridge Quantum Computing Ltd., IBM Research Yorktown Heights, NSERC Canada, and University of Victoria. The authors thank IBM CAS for funding the project entitled "Quantum Problem Solving and Algorithm Design on the IBM Quantum Platform."

REFERENCES

[1] 2020. *Archer.* https://archerx.com.au/
[2] 2020. *Cambridge Quantum Computing.* https://cambridgequantum.com
[3] 2020. *CDL Quantum Stream.* www.creativedestructionlab.com/streams/quantum
[4] 2020. *Hybrid quantum-classical Neural Networks with PyTorch and Qiskit.* https://bit.ly/34hL5Z4
[5] 2020. *Miraex as finalist of the NASA Challenge.* https://bit.ly/3jdPvXj
[6] 2020. *Quantum Benchmark.* https://quantumbenchmark.com
[7] 2020. *Riverlane.* www.riverlane.com
[8] 2020. *softwareQ.* www.softwareq.ca/
[9] 2020. *Variational Quantum Linear Solver.* https://qiskit.org/textbook/ch-paper-implementations/vqls.html
[10] F Arute, K Arya, and R Babbush et al. 2019. Quantum supremacy using a programmable superconducting processor. *Nature* 574 (Oct 2019), 505–510.
[11] Zapata Computing. 2020. *Orquestra.* www.orquestra.io/
[12] The Economist. 2017. Tech. Quart.: Quantum Devices, Here, there & everywhere.
[13] AW Cross et al. 2019. Validating quantum computers using randomized model circuits. *Phys Rev A* 100 (Sep 2019), 032328. Issue 3.
[14] B Sussman et al. 2019. Quantum supremacy using a programmable superconducting processor. *Quantum Sci Technol* 4, 2 (2019).
[15] E Farhi, J Goldstone, and S Gutmann. 2014. A Quantum Approximate Optimization Algorithm. *Arxiv* (Nov 2014). arxiv.org/abs/1411.4028v1
[16] J Gambetta. 2020. *IBM's Roadmap For Scaling Quantum Technology.* IBM. www.ibm.com/blogs/research/2020/09/ibm-quantum-roadmap/
[17] Microsoft. 2020. *Azure Quantum.* https://azure.microsoft.com/en-us/services/quantum/
[18] NRC. 2019. *Economic Impact of Quantum Tech.* https://nrc.canada.ca/en/research-development/research-collaboration/programs/economic-impact-quantum-technologies
[19] PASQAL. 2020. *PASQAL.* https://pasqal.io/
[20] A Peruzzo, J McClean, and P Shadbolt et al. 2014. A variational eigenvalue solver on a photonic quantum processor. *Nat Commun* 5 (Jul 2014), 4213.
[21] IBM Q. 2020. *IBM Quantum Experience.* https://quantum-computing.ibm.com/
[22] Amazon Web Services. 2020. *Amazon Braket.* https://aws.amazon.com/fr/braket/
[23] S Sivarajah, S Dilkes, and A Cowtan et al. 2020. t|ket⟩: A retargetable compiler for NISQ devices. *Quantum Science and Technology* (2020).
[24] Xanadu. 2020. *Xanadu Quantum Cloud.* www.xanadu.ai/cloud-platform

IBM Centre for Advanced Studies

CASCON x EVOKE 2020

About IBM Centre for Advanced Studies

Our mission

IBM Advanced Studies works with students, faculty and industry professionals to deliver innovation that matters—for IBM, and for the world. We strive to

1. **Equip** students to apply IBM technology to real-world challenges
2. **Enhance** innovation capacity through joint academic-industry projects
3. **Inspire** meaningful human outcomes through empathy for individuals

Our centres

We have a global network of **IBM Centres for Advanced Studies** (https://ibm.com/ibm/cas) which specialize in high touch collaborative partnerships with post-secondary institutions. In 2020, CAS Canada, CAS Atlantic and CAS Alberta together supported 57 academic-industry research projects associated with 105 IBM products, involving 124 IBMers, 59 faculty and 129 students.

Our conference

The annual **CASCON x EVOKE** academic-industry conference is a unique opportunity to engage with a community of software developers, researchers, students, academics, decision-makers and IBMers promoting cross-collaboration, driving forward-thinking research and building informed solutions.

The year 2020 marked the 30th anniversary of our Centre for Advanced Studies Conference (CASCON), our first-ever virtual edition, and our second year of partnership with the EVOKE Foundation. Due to extenuating circumstances, we moved our 30th anniversary celebration to CASCON x EVOKE 2021.

Our future conferences are planned as *hybrid* events—offering both convenient global *virtual* access and an immersive *physical* venue. Mark your calendars with confidence.

- **CASCON x EVOKE 2021** – November 21-26 – *Enercare Centre, Toronto* + virtual access
- **CASCON x EVOKE 2022** – November 15-18 – *Metro Toronto Convention Centre* + virtual access

Contact us

- Interested in work-integrated learning or research collaborations? Excited to get involved in CASCON x EVOKE? Want to help celebrate our 30th? **Contact us at casinfo@ca.ibm.com**

Marcellus Mindel
Head, IBM Centre for Advanced Studies