COPPE
UFRJ
Instituto Alberto Luiz Coimbra de
Pós-Graduação e Pesquisa de Engenharia

MACHINE LEARNING EXPLORATION FOR EMERGING STORAGE AND NETWORKING APPLICATIONS

Victor da Cruz Ferreira

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientadores: Felipe Maia Galvão França
Sandip Kundu

Rio de Janeiro
Novembro de 2022

MACHINE LEARNING EXPLORATION FOR EMERGING STORAGE AND NETWORKING APPLICATIONS

Victor da Cruz Ferreira

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientadores: Felipe Maia Galvão França
Sandip Kundu

RIO DE JANEIRO, RJ – BRASIL
NOVEMBRO DE 2022

da Cruz Ferreira, Victor

Machine Learning exploration for Emerging Storage and Networking applications/Victor da Cruz Ferreira. – Rio de Janeiro: UFRJ/COPPE, 2022.

XXVI, 228 p.: il.; $29,7$cm.

Orientadores: Felipe Maia Galvão França
Sandip Kundu

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2022.

Referências Bibliográficas: p. 117-139.

1. Computational Storage. 2. Machine Learning. 3. Artificial Intelligence. 4. NAND Flash Media. 5. Reinforcement Learning. 6. Reliability. 7. Fake News. I. França, Felipe Maia Galvão *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Agradecimentos

First of all, I would like thank my two advisors, Professor Felipe França and Professor Sandip Kundu. Professor Felipe has guided me since my Master's and always helped me find incredible opportunities which have made my PhD experience worthwhile. Professor Sandip is constantly coming up with new ideas, which tremendously assisted me in finishing my PhD. I am very appreciative of both because without them this thesis would not be the same.

I would also like to thank some of my colleagues I made along the journey. Brunno, Santiago, and Leopoldo were always available and great people to brainstorm with. The jokes, gaming, and existential crisis conversations were also top-tier. Our Açai stand shall become a reality some day. Other colleagues from LAM including Rui, Victor Cracel, Caio, Diego, and Hugo were amazing friends that I am thankful for. Thank you to my former advisors Marzulo and Popov who were always there for me whenever I needed them.

I want to express my gratitude to UCI Professor Nader for hosting me as a visiting scholar in California. Thank you, Vladimir, Shafa, Siavash, Mahdi, Min Soo, and Vince for being good friends and always helping me out during my time at NGD. The basketball times with Patrick and Ifthikar were also pretty good (although I would always break their ankles). Venky and Laura were the power couple that lived with me and are dear friends I carry to this day. Thank you for your friendship and for putting up with me during COVID isolation.

My girlfriend Aline was also an integral part of the last sprint of my thesis, I am extremely grateful for your support babe. I would also like to thank my mother and father as well as my blood and non-blood family for their incredible support. A 10+ year brotherhood is pretty special to have, so a special thanks to Saldanha, João, Marcos, Sampaio, Luis, Guigs, and Gabriel.

I would like to thank PESC/COPPE/UFRJ for their amazing program and environment. Thank you CAPES, CNPq and COPPETEC for their help during my research.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

EXPLORAÇÃO DE APRENDIZADO DE MÁQUINA PARA TECNOLOGIAS EMERGENTES DE ARMAZENAMENTO E REDE

Victor da Cruz Ferreira

Novembro/2022

Orientadores: Felipe Maia Galvão França
Sandip Kundu

Programa: Engenharia de Sistemas e Computação

O desenvolvimento de tecnologias está acontecendo cada vez mais rápido, o que está criando uma maior produção de dados, que por sua vez acabam gerando novos problemas. Estes problemas possuem muitas variáveis para modelar, o que aumenta sua complexidade, ocasionando soluções tradicionais não serem escaláveis para as novas tecnologias. *Machine Learning* (ML) oferece uma possível solução para aprender tal complexidade e se auto-ajustar com novas informações recebidas. Com isso em mente, esta tese foca em problemas diferentes situados em três diferentes tecnologias recentes propondo diferentes soluções e estudos através de ML. Primeiro, é realizada uma análise de *Solid State Drives* (SSDs), mais especificamente um problema de durabilidade de mídias NAND Flash e o deslocamento de suas distribuições de voltagem. Este trabalho também avalia uma variação recente de SSDs que adiciona recursos de processamento *in-situ*. A avaliação usa vários algoritmos de ML de *object tracking* para estudar diversas métricas de desempenho. O texto também incorpora o problema de controle de admissão para redes de próxima geração relacionado ao novo recurso chamado *Network Slicing* introduzido no 5G atual e que deve ser aprimorado ainda mais para as gerações futuras. O autor formaliza um ambiente dinâmico e propõe uma solução de modelo de ML com o objetivo de maximizar um objetivo importante para o provedor de recursos. Por fim, o problema de detecção de *Fake News* é sujeito a diversas questões como censura e confiança na previsão. Portanto, com o objetivo de melhorar a confiabilidade nos classificadores de *Fake News* existentes, este trabalho apresenta uma metodologia para aumentar a transparência dos modelos caixa-preta de ML que utilizam redes *feed-forward*.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

MACHINE LEARNING EXPLORATION FOR EMERGING STORAGE AND NETWORKING APPLICATIONS

Victor da Cruz Ferreira

November/2022

Advisors: Felipe Maia Galvão França
Sandip Kundu

Department: Systems Engineering and Computer Science

New technologies generate new problems and complicate existing ones. Such problems come with too many variables to be modeled and traditional solutions hardly scale. Machine Learning (ML) may provide a doorway for learning new variables and adjusting automatically to incoming new information. With that in mind, this thesis focuses on three different developing technologies. It proposes different solutions and studies by using ML to solve different problems that arose due to the complexity of the data and technology itself. We first look into Solid State Drives (SSDs), more precisely one reliability problem related to NAND Flash Media and shifting voltage distributions. This work also evaluates a recent variation of SSDs called Computation Storage Devices that powers SSDs with in situ processing capabilities. The evaluation uses several object tracking ML algorithms to study different performance metrics such as power consumption and throughput. The text also incorporates the Next-Generation Networks admission control problem related to the Network Slicing feature introduced in current 5G networks and is expected to be improved further for future generations. The author formalizes a dynamic environment and proposes an ML model solution to maximize an objective important to the provider of resources. Finally, the problem of Fake News detection incurs several issues such as censorship and prediction trust. Therefore, to improve trust in existing Fake News classifiers, this work also introduces a methodology for increasing transparency of black-box feed-forward ML models.

Contents

List of Figures **x**

List of Tables **xiii**

1 Introduction **1**

1.1 Motivation 1

1.2 Main Contributions 3

1.3 Thesis Outline 4

2 Background **6**

2.1 Storage Devices and Components 6

2.1.1 NAND Flash Media 6

2.1.1.1 NAND Flash Architecture 6

2.1.1.2 Flash Operations 8

2.1.1.3 Reliability Issues 8

2.1.2 Solid State Drives (SSD) 10

2.1.3 Computational Storage Device (CSD) 12

2.2 Machine Learning Repertoire 15

2.2.1 Deep Neural Networks 15

2.2.2 Weightless Neural Networks 16

2.2.2.1 WiSARD Classifier 17

2.2.2.2 Regression WiSARD 18

2.2.3 Boosted Trees - XGBoost 18

2.2.4 Deep Q-Networks 19

3 Dynamic NAND Flash Media Read-Level Threshold Adjustment **23**

3.1 Motivation 23

3.2 Related Work 24

3.3 Read-Level Threshold Adjustment with ML 26

3.3.1 Dataset Collection 27

3.3.2 Machine Learning Implementation 29

3.3.2.1 Pre-processing & Features 29

3.3.2.2 Training . . . 32
3.3.2.3 Inference . . . 33
3.4 Experiments & Results . . . 34
3.4.1 Evaluation Metrics . . . 35
3.4.2 Training Set Evaluation . . . 36
3.4.3 Time-series Comparison . . . 38
3.4.4 Model Evaluation . . . 39
3.4.5 Execution Time & Memory Footprint . . . 41
3.5 Concluding Remarks . . . 43

4 Object Tracking in Computational Storage **44**
4.1 Motivation . . . 44
4.2 Related Work . . . 46
4.3 Object Tracking for CSD . . . 47
4.3.1 DNN Trackers . . . 48
4.3.2 Correlation Filters . . . 50
4.3.3 Weightless Neural Network Tracker . . . 51
4.4 Experiments & Results . . . 52
4.4.1 Throughput Analysis . . . 53
4.4.2 Energy Consumption Analysis . . . 55
4.4.3 Accuracy Discussion . . . 56
4.5 Concluding Remarks . . . 58

5 Admission Control on a NextG Dynamic Environment **59**
5.1 Motivation . . . 59
5.2 Related Work . . . 60
5.3 Reinforcement Learning Framework for Slice Admission Control . . . 62
5.3.1 System Model . . . 62
5.3.2 MDP Formulation . . . 64
5.3.3 CNN-based Slice Admission Control . . . 66
5.3.4 Dynamic Pricing . . . 67
5.4 Experiments & Results . . . 69
5.4.1 Model Comparison . . . 70
5.4.2 Arrival rate & Slice duration . . . 73
5.4.3 Hyperparameter Discussion . . . 75
5.5 Concluding Remarks . . . 77

6 Improving transparency on Fake News Classification **79**
6.1 Motivation . . . 79
6.2 Related Work . . . 81

6.3 Increasing transparency of Fake News Model 83
6.3.1 Explainability & Interpretability Techniques 83
6.3.2 SAFE: Similarity-Aware Multi-modal Fake News Detection . . 84
6.3.3 A Transparent SAFE$-\beta$ Model 86
6.4 Experiments & Results 87
6.4.1 Explainability on Gossipcop 88
6.4.2 Explainability on Politifact 92
6.4.3 Accuracy and F1 94
6.4.4 Representation Erasure 94
6.4.5 Impacts of transparency for future Datasets construction . . . 96
6.5 Concluding Remarks 97

7 Conclusion **98**

References **101**

Appendices **118**

A Dynamic Read-level Voltage Adjustment Additional Experiments **119**
A.1 Model Parameters 119
A.1.1 DNN Parameters 119
A.1.2 XGBoost Parameters 120
A.1.3 WiSARD Parameters 120
A.2 Supplementary Time-series Comparison Plots 121

B Object Tracking IoU for individual videos of OTB-100 **125**

C Fake News Transparency Dataset Relevance Scores **131**
C.1 Politifact Top-100 Tokens 131
C.2 Gossipcop Top-100 Tokens 142

List of Figures

2.1 A floating gate transistor representative of a flash memory cell . . . 6
2.2 Representative state distributions for different flash memory cell technologies. Optimal reference read-level voltages are displayed by the red dashed vertical line. 7
2.3 A 3D TLC NAND read threshold voltage distribution measured at 40°C with 7K P/E Cycle count and 3 Months of Data Retention. . . 9
2.4 A generic view on SSD architecture components presenting some basic controller functionalities. 11
2.5 Newport CSD Architecture block diagram consisting of the firmware operation system and *in situ processing system* (Image from [1]) . . . 13
2.6 WiSARD standard architecture and operations receiving a 3x3 binary input. 17

3.1 Data collection, model training and inference with a CSD. 26
3.2 Tabular example of all the collected metrics available in our dataset. . 27
3.3 Failure Rate for all model types, time-series lengths and training sets with using TR of 1.0 and MOL Data. 36
3.4 Failure Rate for all model types, time-series lengths and training sets with using TR of 1.5 and MOL Data. 37
3.5 Failure Rate for all model types, time-series lengths and training sets with using TR of 2.0 and MOL Data. 37
3.6 Model comparison in MOL flash data based on best performing time-series and training sets by model. 40
3.7 Model comparison in BOL flash data based on best performing time-series and training sets by model. 40

4.1 Object tracking general flow: (a) Generic view of how object tracking is performed via a black-box algorithm. Its result can be see by the yellow bounding boxes. (b) Usual flow of the object tracking algorithm in conventional storage media. Data is loaded from said media and the host executes the algorithm. (c) An example of a Flow for a CSD device. Data is loaded and processed in-storage, the host only merges the result and results data preprocessed. (Image from [1]) 47
4.2 Experimental server blade with high capacity in-storage processing Newport CSDs. (Image from [1]) 53
4.3 Energy consumption (Joule/frame) versus accuracy metric (IoU) of the tested models in different devices. 57

5.1 Admission control system components and flow overview. 62
5.2 Our matrix representation of the state and how the features are correlated with the current queue requests a a given time step t. 65
5.3 Amount of cumulative reward at the final training epoch for each model and each dataset scenario. 71
5.4 Model convergence as training progresses throughout our environment for the different models and different datasets 72
5.5 Overall average percentage of acceptance for each service class. The values are an average of the final epoch of the models for each dataset. 74
5.6 Total amount of accepted requests for the final training epoch for each model and dataset. 75

6.1 Word Importance when using Gossipcop with SAFE-β 89
6.2 Word Importance when using Gossipcop with SAFE-β text-only . . . 89
6.3 Representation Erasure for Politifact with SAFE-β text-only varying removal of 0 to 15 tokens. Orange line is for erasures on headline only, green is for body-text only, while blue represents erasing words on both inputs. 95
6.4 Representation Erasure for Gossipcop with SAFE-β text-only varying removal of 0 to 15 tokens. Orange line is for erasures on headline only, green is for body-text only, while blue represents erasing words on both inputs. 95

A.1 Different DNN time-series variation comparisons on RBER % vs DR for both BOL and MOL datasets. 121
A.2 LINEAR time-series comparisons on RBER % vs DR for both BOL and MOL datasets. 122

A.3 XGBoost time-series lengths comparisons on RBER % vs DR for both BOL and MOL datasets. 123
A.4 Different REW time-series variation comparisons on RBER % vs DR for both BOL and MOL datasets. 124

List of Tables

2.1 Attributes for different memory cell technologies. Values based on Kingston flash media [2] 9

3.1 Prediction time of one sample in milliseconds (ms) and approximate memory consumption in Megabytes (MB). 41

4.1 Object Tracker implementation availability per device tested. 52
4.2 Several performance metrics for each model executed on their appropriate available devices. 54
4.3 Accuracy-related Metrics for the Object Tracking algorithms. 56

5.1 Average Waiting times in time steps for the final training epoch of each model and dataset scenario. 70
5.2 Reinforcement Learning parameters at the end of Ray Tune Random Search. 76

6.1 Top-10 tokens and their normalized relevance scores for each of the three text inputs on Gossipcop Dataset using SAFE-β 90
6.2 Top 10 Words for each prediction type on Gossipcop using only Politifact using only Headline and Body-text. 92
6.3 Top-10 tokens for each prediction type on Politifact using only Headline and Body-text. 93
6.4 Different Metric results for both FakeNewsNet datasets. SAFE* values are the ones reported from the original paper, while SAFE-β is the tree input version and SAFE-β Text-Only represents our Headline and Body-text only version 94

A.1 DNN with LSTM parameters. 119
A.2 XGBoost parameters 120
A.3 Regression WiSARD parameters 120

B.1 IoU per Object Tracker algorithm for every video present in OTB-100. 130

C.1 Politifact Headline top-100 token scores for different predictions. . . . 136
C.2 Politifact Body-text top-100 token scores for different predictions. . . 141
C.3 Gossipcop Headline top-100 token scores for different predictions. . . 147
C.4 Gossipcop Body-text top-100 token scores for different predictions. . . 152

Chapter 1

Introduction

1.1 Motivation

As new technologies emerge at an increasing rate, the amount of data and variables to assess becomes increasingly burdensome. Big Data computation is not just an issue for the future but also for the present. It is currently unclear how to discern important information or process on such a massive amount of data in a reasonable time frame while providing reliable solutions and concrete analysis.

Artificial Intelligence (AI) is increasingly being used as it outperforms other previous techniques due to several recent breakthroughs [3]. More specifically, Machine Learning (ML), a subset of the more significant AI Field, encompasses algorithmic approaches that learns patterns from the data without any explicit programming. By providing an automated learning algorithmic solution, one does not need to depend on rigid structures, which may decrease the system's overall reliability. Moreover, by letting the algorithm learn by itself critical information, it tends to be a more scalable solution. Still, it is essential to acknowledge that other fields inside the AI ambit are being explored in the research community, such as chatbots, which are usually categorized as AI rule-based, algorithmic approach [4].

With this in mind, the author would like to offer insight and motivation for a few specific emerging fields where open problems are still fresh. They typically present a complex scenario with many dynamically changing variables where out-of-the-box solutions do not perform well.

Storage: For years, the storage domain was plagued as the slowest branch of the traditional memory hierarchy. Slow Hard Disk Drives (HDDs) with mechanical parts significantly impaired progress in storage throughput speed. However, the continued development of flash media by improving architecture, miniaturization, and cost-efficiency has enabled mass-storage Solid State Drives (SSDs) to become the front-runner technology to improve disk access and increase system throughput.

However, with the introduction of new media, new problems arise. In flash media specifically, several reliability issues that may disrupt data consistency have been researched. Measures were implemented within the SSD controller to mitigate and avoid data loss [5–7]. Still, some issues are ingrained in the memory cells floating gate architecture, which requires further attention. For example, finding the correct voltage threshold is essential to avoid bit errors when performing read operations. Unfortunately, the optimal voltage values shift due to several factors innate to the floating gate transistor technology and SSD operation. For example, Data Retention (DR) is a well-known time-related problem that occurs on traditional DRAM and is also one of the major causes of negative voltage shifts in flash. The manufacturing process variation is another reason that may produce different voltage threshold values. Therefore, an ML solution that can correctly predict and take action to find the correct reference voltage given the available values collected and provided by the SSD will significantly improve flash media's reliability. Moreover, using the correct read-level voltage threshold values reduces the probability of a high bit error count in the SSD, which can then use fewer resources devoted to fixing them. For example, it might leverage less powerful Error Correction Codes (ECCs), decreasing read latency or hardware area consumption. Unlike other static approaches where one needs to research and create assumptions on how the will behave to determine a threshold [8–10], a ML based approach simply learns from the data and can be better prepared for unaccounted patterns.

Since SSDs provide several additional features that original HDDs could do without, their processing power, in general, is more extensive. Similar to what happened at the beginning of Graphical Processing Units (GPUs), researchers started using the extra computational capabilities to execute user-defined applications [11–15]. With the growing demand, Computation Storage Devices (CSDs) came into play with in-storage computational capabilities. These devices usually offer a low-power solution that decreases data movement around the system as the data can be either preprocessed or fully processed inside the storage device. Although their computational capabilities are not as strong as the main CPU or GPU devices, the concurrent use of multiple devices while decreasing data transfers can be used to leverage ML applications. To study their capabilities in an ML environment, we carry out several experiments and measurements to assess various aspects of their performance. Moreover, we also weigh the in-storage processing when predicting read level voltages in CSDs as they will have closer access to the media.

Next-Generation (NextG) Networks: Today, the world is moving towards deploying 5G networks across multiple countries. 5G is thought to be an essential step into enabling IoT and massive device communication in cities, factories, and homes. However, 5G lacks the capacity to enable complete automated orchestration and integration of different system layers and services that will provide everything as a service [16]. For example, emerging new services such as Virtual Reality are expected to require a much higher throughput than currently available in 5G [17]. Therefore, NextG networks, like 6G, are already being speculated and formalized to improve on 5G features.

One of the traits anticipated from 6G is connected intelligence with machine learning capabilities [18]. The idea is to natively support AI and allow it to be the nucleus of the network, being included in every communication step [19]. The AI models should manage by being intelligent and agile, allowing the network to adapt by learning the changing dynamics of the environment. Already in 5G, Virtualization plays a significant role in deploying infrastructure-as-a-service, enabling important services such as network slicing. However, thoroughly having an adaptable decision algorithm when managing incoming resource requests to allow for automation and maximization of an objective is not available in the ambit of the current generation but goes in line with the proposed goal of NextG. With that in mind, we look to leverage an ML algorithm in a dynamic NextG environment to study its viability as a solution for automating request admission control.

Fake News Classification Transparency: The explosion of social media and information access has led to the worldwide spread of fake news. The problem has become so pervasive that many consider it a threat to democracy, leading it to affect essential areas such as economics and elections [20]. Trying to solve this problem with ML algorithms is quite challenging. Automated classification of news articles comes with complex structures, including data with multiple modalities and hard-to-label samples.

Unlike the other problems we will tackle throughout this thesis, Fake News Classification has been researched for a while, and some progress has been made to improve ML algorithm accuracy and performance. However, these algorithms usually work as a black box without any transparency to the user. In fake news detection, user trust is essential as the problem may deal with inappropriate article filtering and censorship. For that reason, generating a viable human understandable explanation for the model decision is preferred when selecting a model due to its transparency [21]. We provide a novel methodology using explainability and interpretability techniques together to introduce transparency into a state-of-the-art Fake News Classification ML algorithm.

1.2 Main Contributions

This thesis tackles several emerging problems from newly developing technologies and fields using ML algorithms and techniques. Consequently, we explore several

ML algorithms such as Boosted Trees, Deep Neural Networks, and Weightless Neural Networks. Each of the three scenarios (Storage, NextG, and Fake News) requires domain specific approaches and different evaluation strategies.

Each chapter's introduction section will give a more thorough overview of the contributions of the corresponding work. However, the main contributions of this thesis are summarized below.

- A cohesive dataset with several metrics regarding flash media read error information through data collection and preprocessing steps. The dataset can be used in distinct ways and tries to convey realistic values that can be probed and shared by an SSD firmware;
- Novel read-level voltage threshold prediction ML approach to dynamically adjust voltage thresholds without the explicit use of data retention by leveraging the many features of the collected dataset;
- Study of performance for object tracking ML applications on a multi-CSD environment;
- Introduction of a dynamic admission control NextG environment with dynamic pricing modeling that accounts for supply and demand;
- A Dueling Deep Q-Network Reinforcement Learning algorithm that uses a vision-based approach to maximize the infrastructure provider profit in a dynamic setting;
- Novel transparency enabling methodology for a black-box Fake News Classifier, which can be generalized for any other Deep Neural Network.

1.3 Thesis Outline

The remainder of the thesis is organized as follows: Chapter 2 gives some necessary background to understating several dynamics of Storage structure and components, including flash media, Solid State Drives, and Computational Storage. Moreover, it provides an overview of several ML models used throughout the thesis. Chapter 3 presents and discusses a solution to use ML for improving flash media reliability. The reader will find some of the network parameters from this work in Appendix A.1. Chapter 4 provides a study on several ML techniques executing in a computation storage environment. Appendix B is related to results discussed in this chapter. Chapter 5 introduces and proposes a solution to admission control in dynamic environments using reinforcement learning. Chapter 6 provides a novel methodology to increase black-box model transparency. The reader will find some of the results discussed in this chapter in Appendix C. Lastly, Chapter 7 gives home final thoughts on the chapters by concluding the thesis.

Chapters 3 to 6 follow a same section structure to improve readability. Each of these chapters starts with an introductory section, giving more specific insight into the motivation behind the work. A prior/related work section later follows this to give the reader a broader view of the current solutions being researched and highlighting how the proposed work differs from such solutions. The third section explains the proposed solutions, providing a thorough explanation of the implementation and reasoning behind our choices. The fourth section evaluates the solution by explaining the experiments and discussing their results. The final section gives an overview and some final comments about the chapter.

Chapter 2

Background

2.1 Storage Devices and Components

Increased development of new chip technologies by manufacturing process scaling enabled NAND Flash as the primary source of non-volatile media in SSDs. Because of flash memory chips, SSDs deliver orders of magnitude higher sequential and random performance in a smaller form factor and with better power efficiency when compared to traditional hard disk drives (HDDs), which are greatly limited by mechanic components.

In this section, we will discuss architecture, operations, and the main components necessary to understand better the work presented in the following chapters of this thesis.

2.1.1 NAND Flash Media

2.1.1.1 NAND Flash Architecture

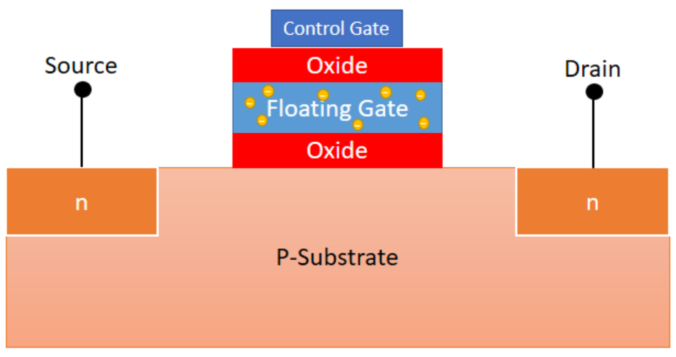


Figure 2.1: A floating gate transistor representative of a flash memory cell

NAND Flash stores data in a floating gate transistor; this is known as a memory cell as per Figure 2.1. The floating gate sits around two oxide insulators that keep

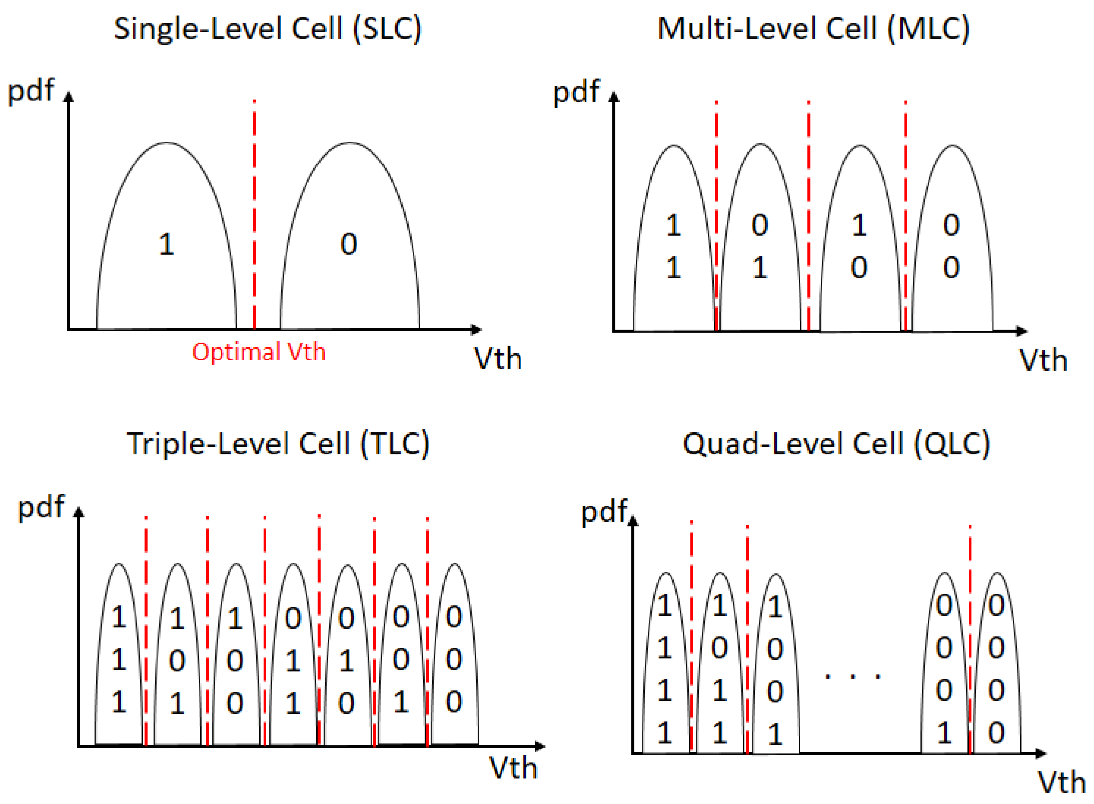


Figure 2.2: Representative state distributions for different flash memory cell technologies. Optimal reference read-level voltages are displayed by the red dashed vertical line.

the electrons trapped within the transistor, enabling its non-volatile properties. The amount of charge trapped imposes a threshold voltage (V_{th}), which is later used to determine the bits stored in the cell.

Initially, each cell packed 1-bit of data and only a threshold value was necessary to discern between two voltage levels, also referred to as states, representing bit 0 (programmed) and bit 1 (erased). These cells are known as a Single-Level-Cell (SLC). Due to the consistent development of scaling technology, decreasing the size of the floating gate transistors, memory cells are more densely packed, storing more than one bit. Figure 2.2 display a simplistic representation of voltage probability density functions in the currently available cell technologies in the market. We have SLC with 1 bit and two voltage levels, Multi-level-Cell (MLC) storing 2 bits with four levels, Triple-Level Cell with 3 bits and six levels, and Quadruple-Level-Cell (QLC) allowing 4 bits per cell with 16 voltage levels. Another progress that enabled flash chips to store more data is the use of 3D stacked memory cell layers vertically which greatly optimizes area, differently than its predecessor (2D) that had layers placed horizontally.

In general, a *chip* may contain from 4, 8 or 16 *dies* which can operate interdependently of each other. Every die is divided into *planes* which contain thousand of *blocks*. Every block can be visualized as a matrix with a set of hundreds or thousands of rows representing *wordlines* and columns named *bitlines*. Inside the blocks, we can have the division of *pages* which are usually the smallest granularity available for reading and writing operations. The memory cells within a *wordline* are connected to form a page that usually consists of 8 or 16 KBytes in size [22]. It is the responsibility of the SSD firmware to orchestrate and maximize the parallelism of this structure.

2.1.1.2 Flash Operations

In general, the basic flash operations can be summarized into *read*, *program* and *erase*.

To perform a *read operation* one must distinguish between the several states within the cell. For example, when dealing with SLC, we must tell apart if the cell is in state zero or one, which can be done by sensing the current-voltage in the control gate of the transistor and comparing it with a read reference voltage threshold, which is the V_{th}. The higher the number of bits packed in a single cell, the more reference voltages are required to distinguish the states, as seen in Figure 2.2 by the dashed red lines.

When performing a *programming operation*, flash cells use incremental step-pulse programming (ISPP) [23] to force a set of electrons into the floating gate, bypassing the isolation layer by applying a high positive voltage into the control gate. In previous architecture such as SLC, one-shot programming was the proffered method for programming operations. However, due to more density-packed transistors, the programming interference between neighboring cells was too high, and a slower but more controllable scheme using ISPP is currently in use.

In *erase operations* we must apply a high negative voltage to release the electrons trapped within the floating gate. Note that, to reprogram a cell, one must first perform a erase followed by a program which is referred to as a *Program/Erase (P/E) Cycle*. Moreover, due to flash block cells being connected, the smallest unit of erasure in an SSD is a block [24].

2.1.1.3 Reliability Issues

As one can expect, by performing several operations, the transistor components start to degrade over time. More specifically, when performing program and erase operations, the isolation layer starts to lose its properties and allows the electrons trapped within the floating gate to tunnel to the outside. The current technologies further exacerbate the degradation and tunneling problem because they pack more

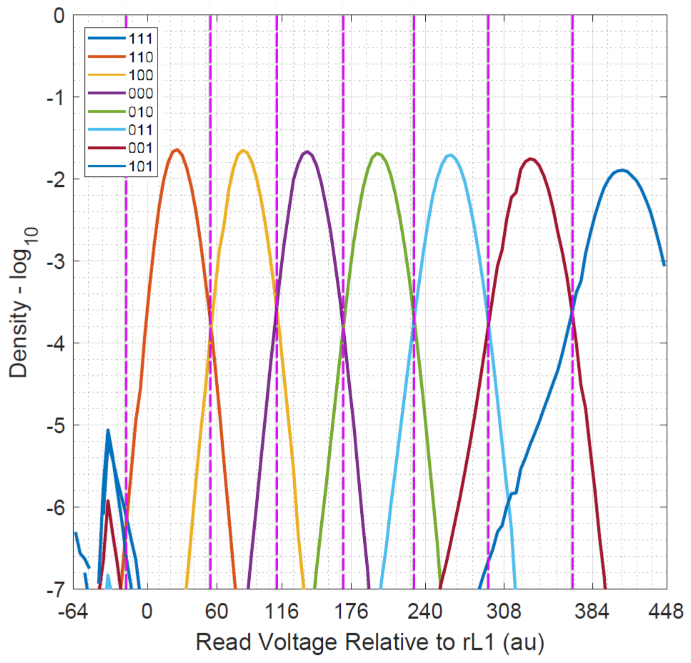


Figure 2.3: A 3D TLC NAND read threshold voltage distribution measured at 40°C with 7K P/E Cycle count and 3 Months of Data Retention.

	SLC	**MLC**	**TLC**	**QLC**
Max P/E	100K	10K	3K	1K
Bits per cell	1	2	3	4

Table 2.1: Attributes for different memory cell technologies. Values based on Kingston flash media [2]

bits into a single cell with fewer electrons due to smaller transistor size [25]. Usually, the lifetime of a flash block is defined by the number of P/E Cycle counts it supports. After the speculated number of P/Es, the number of errors may become too large, and the block may be unable to execute reads or writes. Table 2.1.1.3 shows endurance information for different cell technologies, and, as we can see, it dramatically decreases as there are more bits per cell. Note that P/E specification may change depending on the manufacturer, and the table shows the overall trend, which is consistent between the different technologies.

Another main source of errors is related to read reference voltages. Throughout the lifetime of the flash, due to several factors which change electron distribution within the floating gate, V_{th} starts to shift for each state. Therefore by applying the same reference voltage, we will come across bit errors as the state distributions start to overlap. Figure 2.3 displays an example of real voltage distributions within a TLC. Since measurements were done in a TLC, we can notice eight different distributions representing the possible data bits stored in the cell. The optimal V_{th} to discern between every state is the one that best separates the two neighboring distributions [8], seen by the vertical purple dashed lines. As we can see, cases where the curves are fully separable, are not realistic due to distribution overlaps. Whenever there is overlap, there is a chance of bit errors, as it might not be able to differentiate between states completely. Therefore, to minimize the number of errors, the optimal V_{th} is the one where the overlap is the smallest at the intersection point of the curves.

One of the main causes for V_{th} shift is *Data Retention* (DR). The name 'data retention' comes from the amount of charge the memory cell can hold. As time passes, the charge leaks out and shifts the reference voltages around. Moreover, the temperature can greatly accelerate this process. For example, keeping a chip in an oven at 90 °C for 1-hour would emulate 3-months of DR. For determining DR and temperature parameters, developers usually follow the Arrhenius Law with constant activation energy [26, 27]. Another factor that directly impacts DR is P/E; as the flash ages and isolation capabilities decrease, the amount of charge leaked changes, and DR moves at a different pace [28]. If that is not enough, there still is the manufacturing procedure that is not perfectly equal for every transistor. Therefore each may have different isolation properties. It is important to note that this problem is not exclusive to flash cells, as DRAM also suffers from the same issue, but it solves it by using a refresh mechanism to restore the lost charge, keeping data consistent [29]. Unfortunately, the flash controller is not aware of the exact retention time as it is quite sensitive to temperature [30] and any time-measuring between reads and writes will consume extra hardware and area [31].

Both P/E and DR will shift the V_{th} negatively, while a *read disturb* will shift it to the positive side. Read disturbs happens when performing a read on a cell. Usually, cells are connected in a *bitline* and read separately at a time, with the cells not being read having a passthrough voltage applied to them. This passthrough step may induce electron tunneling and shift distributions causing more errors [32].

It is important to mention that other factors can impact increasing flash errors or change V_{th}, but the ones presented are argued to be the main causes of flash media today. [33]

2.1.2 Solid State Drives (SSD)

SSDs come in different interfaces, some brought from classic HDDs such as SAS and SATA [34] and newer ones that provide higher speed. NVM Express (NVMe) is a host interface protocol designed specifically for high-speed NAND Flash SSDs connected via the PCI Express interface. Typical SSDs reside with multiple flash chips to increase their overall capacity. Its main component is the controller, responsible for executing a firmware that will orchestrate and manage several services an SSD must provide [1]. Different from HDD, SSDs must provide a plethora of robust algorithms to manage their chips. Many of them work on extending flash media's

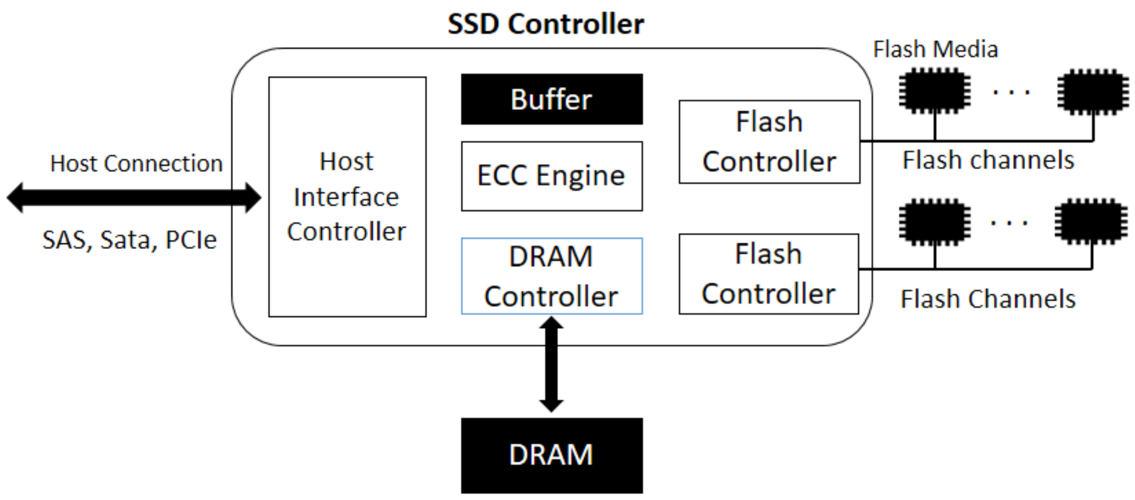


Figure 2.4: A generic view on SSD architecture components presenting some basic controller functionalities.

life by using clever solutions that mitigate the underlying problems discussed in the previous section.

Before we move further into SSD structure, it is crucial to define how we will be referencing *host* and *in situ*. A *host* is an outside-of-storage machine or system from which the storage device is being accessed. For example, the host CPU is the main general-purpose CPU used in typical architectures. On the other hand, when referring to *in situ*, we are referring to the unique computational storage capabilities inside the drive. In this case, a *in situ* CPU is referencing the processing unit inside the storage device.

Figure 2.4 shows a generic view of an SSD architecture. The device disposes of several *channels* which work as a communication bus for the controller to access its non-volatile media. These channels usually work independently by one or more flash controllers and can be accessed in parallel to decrease access latency. Note that a controller may have a buffer and access to a local SSD DRAM for managing and speeding up write and read operations. Another important part of SSD functionality is an interface manager to manage protocol and information from and to the host.

The *Flash Translation Layer* (FTL) primary responsibility is to provide support in I/O operations by translating arriving virtual addresses into physical locations. By providing this low-level mapping table, the firmware can physically move data around without having to report back to the host file system or operating system. For example, the FTL performs *Garbage Collection* (GC) [5] to preemptively move page data from the block to optimize block/page allocation or to avoid a possible lousy block that comprises too many high error pages. Another robust functionality an FTL must provide is called *Wear Leveling* (WL) [6] which tries to homogenize the amount of P/E cycles between the blocks from all chips so as not to have different lifetime regions to manage.

As we discussed in the previous section, voltage distributions tend to shift around as data is written and read throughout the lifetime of the device. Moreover, as we see in Figure 2.3 V_{th} distributions overlap in the real world, which indicates the probability of bit errors is big. Consequently, SSDs must rely on powerful *Error Correction Codes* (ECCs) to correct such errors and avoid data loss. Private customized algorithms based on BCH [35] or LDPC [7] are used in industry today. When performing a write, page data is divided into different chunks in which the controller creates a *codeword* containing the data and correction code with parity information to perform the correction. The stronger the ECC capabilities, the more area it consumes and the higher latency it requires. Therefore, the firmware must always take into account the available trade-offs the algorithm may offer.

The ECC may also report back to the firmware the number of erroneous bits it fixed, known as *Bit Error Count* (BEC). This metric works as an indication mechanism about the current state of the page, block or die. In case the number of errors exceeds the ECC capabilities, a *uncorrectable error* occurs, and the read fails. It is commonplace that read-retries will occur to try and perform a successful read; otherwise, the data is considered lost unless redundancy mechanisms are in place. BEC is also used to derive the likes of *Raw Bit Error Rate* (RBER) and *Uncorrectable Bit Error Rate* (UBER), which are some of the metrics used to evaluate SSD reliability issues and performance [33, 36, 37].

Other techniques such as data scrambling, additional metadata information, and compression techniques can also be deployed by the controller to mitigate errors. Note that all of these algorithms are essential for creating a reliable SSD, especially now that new memory cell technologies with higher bits per cell are being deployed, and reliability has become an increasing issue.

2.1.3 Computational Storage Device (CSD)

With the increased need for computational power in different segments, coupled with the explosion of Artificial Intelligence, traditional Von Neumann architectures are being augmented to provide a more heterogeneous ecosystem of processing devices [1]. Due to a more robust controller, SSDs, in general, provide more computational power than their previous HDD counterparts. Different projects started taking these computational resources and executing user applications instead of usual I/O only operations [11, 12]. Later, Computational Storage came into play as an available system for deploying a general application in a near-data processing paradigm.

Computational Storage provides less data movement by allowing data to be accessed and processed within the CSD. In a typical storage flow, applications request the device for data which is then transferred back to the host system at the host DRAM, which is later consumed and processed by the host CPU. On the other hand, CSDs promote *in situ* processing eliminating the need to send all the data back to the host with better energy efficiency due to less power-hungry processors and high parallelization between multiple storage devices. Moreover, by sharing the workload and spending less time on data transfers, the host processing systems are free to compute different workloads.

In this thesis, we deployed some experiments and measurements in an Enterprise-grade CSD device from NGD Systems known as Newport. For this reason, we will dive into greater detail on this specific style of CSD. It is essential to mention that there are other devices on the market with computational storage capabilities, such as Samsung's SmartSSD [1] which uses an FPGA as its general processing computational system.

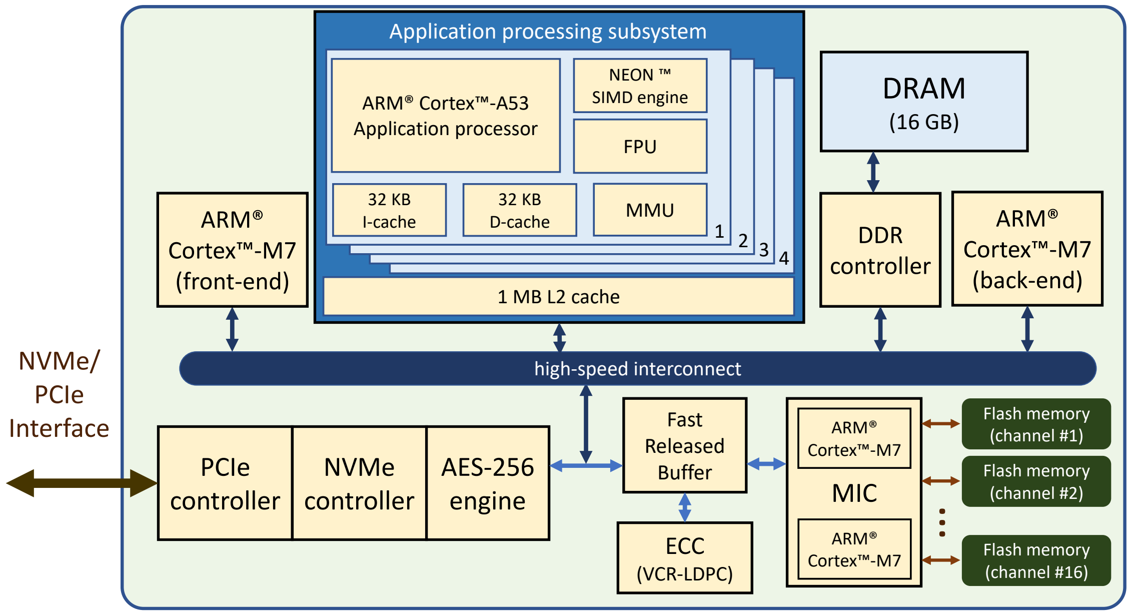


Figure 2.5: Newport CSD Architecture block diagram consisting of the firmware operation system and *in situ processing system* (Image from [1])

Figure 2.5 shows a block diagram for the Newport Drive architecture. The drive can be divided into two main areas, an *in situ* processing system and a Firmware operation system partially represented by the *Application processing subsystem* and the firmware operations area, which mainly works in the background to ensure correct SSD functionality.

Firmware operation system: This region of the drive can be located mainly below the high-speed interconnect in Figure 2.5. Here we will divide functionalities

[1]https://www.xilinx.com/applications/data-center/computational-storage/smartssd.html

by front-end and back-end subsystems that execute routines on four 32-bit embedded real-time ARM M7 processors.

In the area known as *Front-end Subsystem*, Newport provides a PCIe Interface that supports NVMe Protocol through a PCIe and NVMe Controller, which orchestrate protocol and packaging information from a request coming in and out of the drive. It also contains an encryption and decryption engine called the AES-256 XTS unit, which provides cryptography functionality to the drive. This portion of the firmware is executed in one of the four available ARM M7 processors.

The other firmware region is called *Back-end Subsystem.* This subsystem is responsible for most of the standard firmware techniques discussed in Section 2.1.2. The FTL uses one more ARM M7 processor to provide translation functionalities alongside WL and GC. A fast release buffer implemented in hardware is used to organize write operation requests and optimize latency access to the flash media. Here we will also find a dedicated ECC engine unit that uses a variant of LDPC (VCR-LDPC) to ensure data correctness.

We then come across the Memory Interface Controller (MIC), which uses the last two ARM M7 processors responsible for managing eight channels each, totaling 16 channels. The channel bus drives data, address, and commands to all available die within the chips. MIC will send low-level read and write commands to a finite state machine that synthesizes available control signals to compatible values of the flash interface [38].

Even though we are dealing with a CSD *in situ* processing power, it still must provide complete SSD functionality. Notice how similar this firmware portion of the drive is to our general view of an SSD in Figure 2.4.

***In situ* processing system:** To allow for general application execution, the *in situ* subsystem executes a Linux operation system on top of a 64-bit quad-core ARM® Cortex-A53 at 1.5GHz, 1MB L2 cache and supporting up to 16GB DDR4. In practice, most Newport devices (including the one used in this work) will have a total of 8GB of DDR available, with 6GB dedicated to the *in situ* Linux and 2 GB used by the firmware operating system.

The *in situ* subsystem can communicate with the host machine and the internet transparently through an exclusive tunneling application. The tunnel uses a TCP/IP protocol encapsulated inside the NVMe/PCIe headers allowing for complete Linux functionalities and more complex communication using MPI between storage devices or different machines. The TCP/IP Tunnel also allows accessing the device remotely through SSH, just like a separate machine within the machine.

It is clear at this point that Newport must deal with two different flows regarding storage access. The first one is the general SSD flow a storage device must provide: Data operation and access by the host side. The second flow is related to in-storage data access from the Linux *in situ* subsystem. A proprietary block device driver is deployed to transparently interface any I/O communications coming through *in situ* to the controller's internal hardware, resuming normal flow for the firmware side.

2.2 Machine Learning Repertoire

In this section, we will present an overview of multiple ML models and a few of the techniques that were applied to different problems throughout this work. Every model discussion section will overview its architecture and usual operations such as training and inference.

2.2.1 Deep Neural Networks

DNNs have grown in popularity in recent years, achieving state-of-the-art results in different areas and services [3]. With increased popularity and research, the number of available model structures and techniques is steadily growing. In this section, we will focus on structures pertinent to the work developed, more specifically Recurrent Neural Networks (RNN) using Long-Short Term Memory (LSTM) Layers and Convolution Neural Networks (CNNs). Nevertheless, let us start with an overview of a standard deep neural network.

A standard Feed Forward Neural Network is comprised of one or more layers, each with several nodes. Layers are divided into an input layer, where data is fed into hidden layers, and the output layer. The most common nodes in a DNN are based on the original perceptron from McCulloch, and Pitts [39]. Every node can receive one or more inputs that will be multiplied by weights (W) and fed into an activation function that produces a single output. The name Feed Foward comes from data flowing from the input to output, where the outputs of a layer l are the input of layer $l + 1$.

The ultimate goal when training a DNN is to be able to change a set of optimal weights so that it can correctly predict or classify any input sample correctly. The model modifies the weights by trying to optimize an objective, which is generally referred to as a cost or loss function. The cost function dramatically depends on the problem we are trying to solve. To try and adjust the weights, most DNNs apply Stochastic Gradient Descent, or some variation of it [40] coupled with backpropagation [41]. The overall idea of gradient descent is to use the gradient of our cost function to find the way we should modify W to go towards local or global minima. Stochastic Gradient Descent speeds up training by calculating a gradient of a random number of samples called mini-batch instead of the whole data at once. We call the process step of going through all batches from the training data an epoch. Backpropagation comes into play by finding the gradient using the partial derivatives of our cost function [42]. During backpropagation, gradient calculation follows the opposite of the normal network flow, where we start from the output layer and go up to the first layer. Gradient calculations from a layer l are used for the gradient of layer $l - 1$.

Note that several other techniques can and should be applied to optimize training and avoid overfitting.

Feedforward DNNs usually deal with independent samples. Time series problems that require either prediction of future values possess a dependency between samples, where there is a clear indication of past, present, and future. To better grasp this concept and provide better learning, DNNs must have a 'remembering' mechanism that stores past information to optimize the future prediction. Recurrent Neural Networks (RNNs) allow for circular connections between neurons on the same layer while using special units to recognize data sequences and patterns. Such architecture allows RNNs to create a memory of past events that may influence the current input [43].

LSTM Networks are a type of RNN well-suited for time series prediction. Differently from the previously described typical perceptron, an LSTM cell [44] connects with other LSTM cells within the same layer. We can divide an LSTM cell into three main gates: the forget, input, and output gate [43]. Gates use a sigmoid activation coupled with a point-wise operation to control information passing through the cell. The forget gate allows the cell to forget past information that is no longer necessary by managing its internal state. The input gate operates to define how important the current information is and how it should impact the cell state moving forward. At last, the output gate controls how much cell content will be projected into other cells.

Although Fully Connected (FC) networks can achieve good results for multiple problems, they fail to consider spatial information about the data. CNNs introduce a new layer that applies a convolution operation between a matrix of weights and the matrix input. The weight matrix is referred to as a filter or kernel. The resulting matrix after the convolution is known as a feature map. It is commonplace to apply a pooling operation to condense the feature map and decrease the number of parameters after convolution. Although CNNs are mostly used on images, we can also apply them to text data.

2.2.2 Weightless Neural Networks

Weightless Neural Networks (WNNs) use Random Acess Memory units (RAM-neuron) to emulate neuron dendritic tree signaling [45]. The structure and procedure are based on the *n-tuple* classifier [46] which uses a binary memory array accessed by an address set.

In this thesis, we carried out two experiments with different finalities on two versions of a WNN called WiSARD (Wilkie, Stoneham, and Aleksander's Recognition Device) [47]. We first deployed it in a classification problem using a version based on the original WiSARD. Later, we applied it to a Regression problem requiring the usage of the Regression WiSARD (ReW) [48]. The subsections below discuss architecture, training, and inference procedures and their different characteristics.

2.2.2.1 WiSARD Classifier

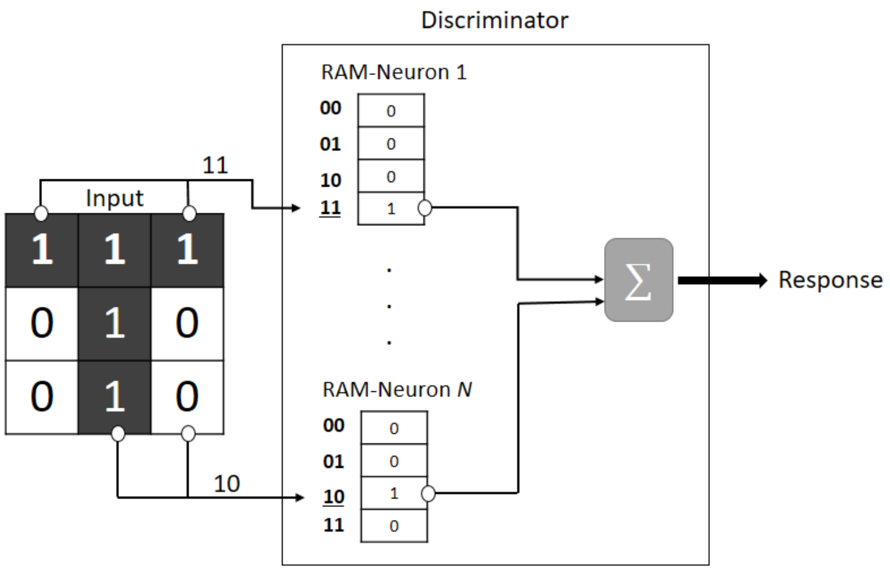


Figure 2.6: WiSARD standard architecture and operations receiving a 3x3 binary input.

The original version of WiSARD is comprised of several discriminator units, each associated with a classification label. Every discriminator is structured with the same number of N RAM-neurons, each containing the same amount of M entries. Each entry can be accessed with a specific address and enable the storage of binary values.

When performing training, all RAM-neurons must be initialized with zero. WiSARD requires the input to go through a binarization preprocessing procedure before feeding it to the model. Afterward, a mapping algorithm will collect the binary input information and generate the appropriate addresses. In general, pseudo-random mapping is used at this stage, but different mapping strategies have been proposed [49]. Note that once the mapping is generated, it is applied similarly for all available inputs. Since WiSARD is a supervised neural network algorithm, all classes must have a ground-truth label. Each label is associated with a different discriminator.

A binary input of size $N \times M$ bits is divided into N different addresses of size M bits, and the mapped address will access different RAM-neuron locations and store a value of '1'. A WiSARD variation enables a bleaching technique [50] which stores a counter that increases every time that address is accessed during training. To allow all possible combinations of binary addresses, every RAM-neuron has a total size of 2^M entries. Figure 2.6 exemplifies the described architecture and how the addresses are generated through pseudo-random mapping to later access the RAM-neurons inside the discriminator.

Instead of storing values inside the RAM-neurons during the inference stage, they are accessed as a look-up table. All discriminators are accessed, and the RAM-neuron contents are retrieved based on the generated M bit addresses. The read contents are summed together to generate a discriminator response. The label associated with the highest scored discriminator is our classification response.

2.2.2.2 Regression WiSARD

ReW is a modified version of the original WiSARD to support regression. The model is structurally similar to its classifier counterpart, as it still uses RAM-neurons to perform regression. However, instead of multiple discriminators, ReW only requires one, significantly decreasing memory consumption.

During training, all preprocessing steps are still required, that is, input binarization and mapping. Unlike WiSARD, the model receives a data pair of $< x, y' >$, where x is a RAM-neuron access address generated based on the input values and mapping, while y' is the ground-truth target of the input. The RAM-neuron content now consists of two variables, a counter c that represents the number of times the region has been accessed and the predicted value y calculated through a sum of ground-truth targets associated with samples that accessed the corresponding memory location during training.

On inference both the value c and y are used to calculate through a simple mean $(\frac{\sum y}{\sum c})$ the predicted target. ReW also supports several other calculations for c and y such as power mean, median, harmonic mean and others [48].

2.2.3 Boosted Trees - XGBoost

XGBoost (eXtreme Gradient Boosting) is a scalable tree boosting ML model that recently gained popularity due to several winning solutions on Kaggle challenges[51]. It bases its structure on combining several weak predictors to form an ensemble of decision trees and mainly works on supervised learning.

XGBoost applies a gradient boosting decision tree algorithm [52] to optimize an objective function. This function is comprised of a loss and regularization term.

Like DNNs, the loss depends on the problem we are trying to solve. In our case, we will be applying XGBoost to a Regression problem. The regularization term prevents the model from overfitting training data.

The model uses an additive approach by adding new trees if they improve the overall objective while the previously added tree structures are fixed in place. The optimization function is modified once a new tree is included to incorporate the new tree scores into the calculations. Every new tree is optimized from level to level by evaluating if splitting a leaf into a new branch would provide any meaningful gain to the model.

2.2.4 Deep Q-Networks

Reinforcement Learning (RL) consists of a general class of algorithms that learns interactively by acting and exploring an environment in a sequential decision-making fashion. The two main pieces of the system are an environment and an agent. We refer to the agent's current view of the environment as a *state*. The agent will take *actions* within that environment and receive a correspondent *reward*. This action will also modify the current state to a new state. To formalize a problem in RL ideas from incomplete theory [53] are applied through a Markov-Decision Process (MDP) [54].

A MDP can be formulated as a tuple $M =< T, S, A, Pr((s'|s), a), r(s, a) >$, where $T = \{1, 2, ...\}$ and represents the time horizon, S and A are the finite-state space and action set respectively, $Pr((s'|s), a)$ is the probability that at the given time t, by taking action a, where $a \in A$ the system transitions from state s to s' in $t + 1$, where $s \in S$ and $s' \in S$. $r(a, s)$ is the reward received from performing action a at the current state and time.

In essence, the model will learn about the environment and indicate to the agent which actions are the best to take given the current state. The model follows what is called a *policy* (ρ), which can be simplified as a set of rules it uses to make its decisions. An optimal policy (ρ^*) is the best way to navigate the environment while maximizing the cumulative reward. The way this policy is defined is entirely dependent on the model.

Interestingly, RL sits between supervised and unsupervised learning [55]. In supervised learning, we require trained data, usually labeled by an outside source, which then is used by the model to generalize and optimize its internal structures to provide a matching answer when similar inputs come by. In unsupervised learning, the idea is to find hidden structures within the data that work as latent characteristics of that input, which the model can adapt to learn and later detect from similar inputs. For RL, it is also vital to generalize internal structures to learn about the data, but actions are not as straightforward as labels, as there is not always one single action that may be best for the current input. Moreover, learning hidden characteristics of the data is essential but does not solve the problem of maximizing our reward in the long term.

By having a model that learns the environment, we introduce the trade-off between *exploration vs. exploitation*. In general, for the model to learn better alternatives to its current policy, it must *explore* new environments by taking new actions, which might cause an initial smaller reward and higher long-term reward. At the same time, the model must *exploit* what it has learned from previous interactions and base its decision on it. The model must strike a balance between only using the learned knowledge and choosing to contradict such knowledge to find better policies. It is essentially trying to find better local minima or, ideally, global minima. Implementing this depends entirely on the selected model, but the common idea is to apply a ϵ-greedy strategy.

In the ϵ-greedy algorithm, we first select an initial ϵ value, decreasing over time until it reaches a chosen minimal ϵ value. A random variable is produced and compared with the current ϵ during the model execution. If the value is lower than ϵ, we take a random action on the environment, exploring it. In the opposite scenario, we use the model knowledge to choose our action.

In this thesis, we will be deploying an RL agent to decide upon an admission control environment. We use a Deep Q-Network (DQN) [56] and Dueling DQN [57] as a solution to automate the decision process based on their previous success in similar environments. However, it is important to mention that other RL algorithms such as Policy Gradient [58] have been deployed in other fields.

DQN is a DNN with a similar architecture to the previously described DNNs in Section 2.2.1 based on Q-Learning [59] RL algorithm. Q-learning updates the value function ($v_\rho(s)$) through the Bellman Equation. A value function ($v_\rho(s)$) represents the goodness of being in the current state s given our current policy ρ. To find goodness, we must look into the expected reward from taking actions within s.

A Q-function ($Q_\rho(s, a)$) is defined as the expected reward of taking action a in state s, following the policy ρ. Its output is also known as Q-value and represents the quality, or the expected value of the action-state pair. Equation 2.1 displays $Q_\rho(s, a)$ formulation.

$$q_\rho(s, a) = \mathbb{E}[r(s, a) + \gamma max_{a'} q_\rho(s', a')] \tag{2.1}$$

The Q-function is comprised of the reward of taking a in s plus a discounted future expected reward. γ represents the discount factor, which works as a weighting parameter to manage long-term vs short-term reward. $max_{a'} q_\rho(s', a')$ indicates the maximum expected future reward, where s' and a' represents the next state-action pair at $t+1$. When $\rho = \rho^*$ this equation is referred to as Bellman Optimality Equation.

Now our optimal value function to find the goodness of state can be written as a function of Q executing the optimal policy ρ^* as shown below:

$$v_{\rho^*}(s) = max_a q_{\rho^*}(s,a) \quad \forall s \in S \tag{2.2}$$

Unfortunately, Q-learning requires a massive structure to store all available actions and state pairs and their corresponding Q-values, making it extremely hard to apply to realistic scenarios due to memory and computational cost. Deep Q-Learning comes into play using Neural Networks to provide a good approximation of Q-values via network training, substituting the memory-intensive Q-learning structure.

A DQN possesses a similar model structure to the other networks presented in Section 2.2.1. It will receive a state as input which will proceed in a feed-forward fashion until reaching the output layer. In general, the output values represent the available set of $q_\rho(s,a)$, $\forall a \in A$. This way, we can choose the action that maximizes our expected reward.

During training, we have to introduce two new concepts: *Experience Replay* and a *target network*.

Experience Replay is a structure used in DQNs to utilize better previous data experienced by the network, which helps with sample efficiency and network convergence [60]. The classic Experience Replay [61] work as a memory structure that stores past experiences in the tuple format of $< s, a, s', r(s,a) >$. After every interaction with the environment, a new tuple is saved. The buffer has a fixed size and will work as a circular buffer, where older experiences are discarded to add new ones. During training, the network will randomly sample a batch of data from the experience replay buffer and use it as training input. By random sampling the data in batch format, we can take advantage of the batch training through stochastic gradient descent. Moreover, it allows for more stable learning by providing data diversity in which the network can better learn a general view of the system [56].

As we explained earlier, RL falls into a paradigm between supervised and unsupervised learning, yet, a DNN still requires us to optimize a loss function (L). Here it takes a page of supervised learning by applying a Mean Squared Loss as per Equation 2.3.

$$L = \mathbb{E}[(y - q_{\rho'}(s,a))^2] \tag{2.3}$$

We need a predicted value (y) and a target corresponding to a ground truth value. Our y is straightforward to find as it is the network's output. However, since data is not labeled, we estimate the target by using the prediction of the same network but with a different policy ρ' that uses the old parameter of a previous time step t. Double Q-networks [62] introduce this concept by deploying a second DNN called target network while training the main DQN. The target network is initialized with the original parameters and weights from the main DQN. As time progresses and our main DQN is trained at every t, the target maintains the old parameters and weights. After an arbitrary number of interactions, we will update the target network with the parameters of the current main DQN. By using this slower update approach, we can better converge the network leading to better policy estimations. The traditional DQN tends to overestimate action-values, creating unstable training and lower quality policies [62]. In practice, the network only knows the reward $r(s, a)$ after the agent acts with a, and the target network is fed the next state-action pair $< s', a' >$ to calculate the maximum expected future reward which allows us to fully calculate $q_{\rho'}(s, a)$

Dueling DQNs build upon the new structures present in the double DQN and standard DQN by modifying the network architecture internally. After the initially hidden layers, Dueling DQN proposes to divide the flow by explicitly creating two separate FC layers. One will represent the state value and the other as a representation of the advantage of taking an action given the state. The original Dueling DQN paper argues that it is not necessary to know the value of all actions for every time step, as there are cases where the action does not particularly modify the environment in a meaningful way [57].

Chapter 3

Dynamic NAND Flash Media Read-Level Threshold Adjustment

3.1 Motivation

The continued improvement in NAND Flash media technology by allowing more densely-packed transistors, which significantly improve storage density and area optimization, has helped solidify SSDs position as the primary storage device in enterprise data centers and consumer devices [37, 63, 64]. As SSDs approach cost per gigabyte similar to HDDs, they also provide orders of magnitude faster read and write throughput [22].

As we previously detailed in Sections 2.1.1.3 and 2.1.2, several issues plague NAND Flash media components. Many such errors can provide data loss and restrain the lifetime of SSDs. Unlike HDDs, SSD architecture requires more powerful components, such as ECCs, to restrain such errors and improve data consistency. Many of these errors are aggravated by recent flash technologies that pack more and more bits into cells which is a trend expected to continue in the future [1].

To briefly recap, when performing a read, we must correctly find reference voltages (V_{th}) to properly discern between the possible flash states. One of the primary sources of data bit errors is caused by shifting voltage distributions inside the cell. Multiple factors influence these shifts, with P/E and DR being the most prominent ones. With that in mind, we propose to dynamically find and adjust read-level reference voltages to decrease the number of bit errors and increase overall SSD reliability. Previous works have tried to approach this problem [65–67], but many involve models using metrics that would be unrealistic to collect in real-time or less flexible structures.

ML, in general, has been successfully used to solve reliability problems [68, 69].

[1]https://www.techpowerup.com/283337/western-digital-may-introduce-penta-layer-cell-plc-nand-by-2025

Therefore, we implemented several ML algorithms with crafted time-series input measured on an enterprise-grade TLC flash media to dynamically predict V_{th}. ML models can be trained offline by using previously measured data on different stages of flash life and later deployed on an SSD drive to adjust the thresholds properly. Moreover, by choosing ML, we can extrapolate on never-before-seen inputs instead of other more static approaches [67]. Models can be trained offline to learn the different characteristics and stages of flash media life and then be deployed for predicting V_{th} read values more accurately.

For the work discussed in this thesis chapter, we propose a dynamically adjustable V_{th} by applying regression ML models on actual measured data from enterprise-grade NAND Flash chips. Our solution is evaluated on varying P/E and DR cycling scenarios with compatible reliability metrics for the storage scene. We chose to execute our application in a CSD environment as we believe near-storage processing can significantly benefit from this scenario.

Below the reader will find the main contributions provided for this work:

- A novel ML approach to dynamically adjust read level thresholds;
- Introduction and analysis of different training procedures using time series by only depending on execution-time available data;
- A representative and realistic dataset collected from enterprise-grade flash media data;
- Analysis of dynamic adjustable V_{th} resulting from extended P/E cycles and data retention;
- Evaluation of memory footprint and prediction time for the ML models in a CSD environment;

This remainder of the chapter is structured as follows: Section 3.2 highlights past work on V_{th} adjustment and a few other techniques on improving flash media reliability. Section 3.3 highlights our implemented ML models as well as data collection and preprocessing to create the dataset. Section 3.4 discusses and showcases the experiments and results with regards to different scenarios and ML models. Section 3.5 concludes the work with some final remarks.

3.2 Related Work

In this section, we discuss past work related to flash media reliability. A handful of such works tried to deploy some type of V_{th} adjustment, while others focused on different techniques.

Mei *et al.* [65] is one of few works that try to use a deep learning-aided approach to adjust read-level thresholds and improve flash media reliability in MLC. It deploys a stacked RNN architecture with Gated Recurrent Units (GRUs) instead of the LSTM. The network is fed a readback threshold voltage which the author argues can be detected by sensing the cell. Note that for the experiments, the inputs are simulated by some channel modeling, emulating programming noise, data retention, and other problems that may infer in flash problems.

One way to find our optimal V_{th} is to perform re-reads, shifting the threshold for every read until a read is performed successfully. This approach is slow and can significantly increase read latency. The work from Peleato *et al.* [66] uses a constant number of reads coupled with an algorithm that characterizes noise distribution for the voltage levels based on Gaussian distribution equations. The authors deploy a dynamic programming backward recursion framework to search for a policy that minimizes Raw Bit Error as a reward. Different than Peleato, we use a different approach to finding the optimal threshold without needing re-reads and using different inputs.

All of the above works use Gaussian voltage estimations to simulate conditions or data in some fashion. Although this assumption may provide a good estimation for some instances, different cell technologies are asymmetric with exponential tails that may look very different due to several factors [9].

Papandreou *et al.* [10] experiments with MLC and report how beneficial even a simple adaptive voltage threshold is in decreasing bit error rate. For their experiments, they varied both DR and did P/E Cycling to force shifting distributions. The work periodically sweeps six pages and compares the optimal, nominal (baseline), and adaptive voltage threshold strategies. The adaptive algorithm performs six reads by shifting V_{th} to find a better V_{th} for the read pages.

Using more recent cell technologies, Rajab *et al.* [67] introduces a V_{th} calibration procedure for TLCs while emulating P/E Cycling and DR for V_{th} shifts. It does so by reading meta-data in different life stages of the page and storing BEC values in a table. To perform calibration, they simply perform a lookup, trying to find a signature similar to the current read values. As expected from tabular approaches, it may grow exponentially due to the sheer size of flash media.

The last two prior works do not entirely account for different physical areas of the flash that might have distinct BEC and V_{th} behavior. As we explained in Section 2.1.1, temperature and manufacturing procedure are factors that modify read-level voltage behaviors. Our approaches deploy ML to provide a more robust solution to handle such variations while keeping a consistent memory footprint.

Fairly recently, Zuolo *et al.* [70] had a patent approved for identifying correct read-level thresholds using regression networks. The network uses several input values such as the reference voltage, retention time, and others that output a list of exponents used to identify what they called 'Threshold-Voltage-Shift Read-Error' curves.

Even though correctly finding V_{th} improves on RBER, a few other works are proposing different solutions to improve overall flash media reliability. Mahdisoltani *et al.* [69] uses S.M.A.R.T. log features as input to an ML model to predict possible SSD failures and preemptively apply solutions.

Wilson *et al.* [71] provides a way of reviving NAND cells by dynamically switching TLCs to work as MLCs. As previously discussed, the lower the number of bits per cell, the higher the number of P/E cycles it sustains. Wilson *et al.* proposes that whenever a block reaches its maximum P/E and is considered dead or retired, we bring it back as an MLC, allowing for more P/E to be performed.

3.3 Read-Level Threshold Adjustment with ML

This section will explain our proposed approach to adjusting read voltage thresholds dynamically. We will also supply knowledge on how we collected and processed our data to fit a generic dataset.

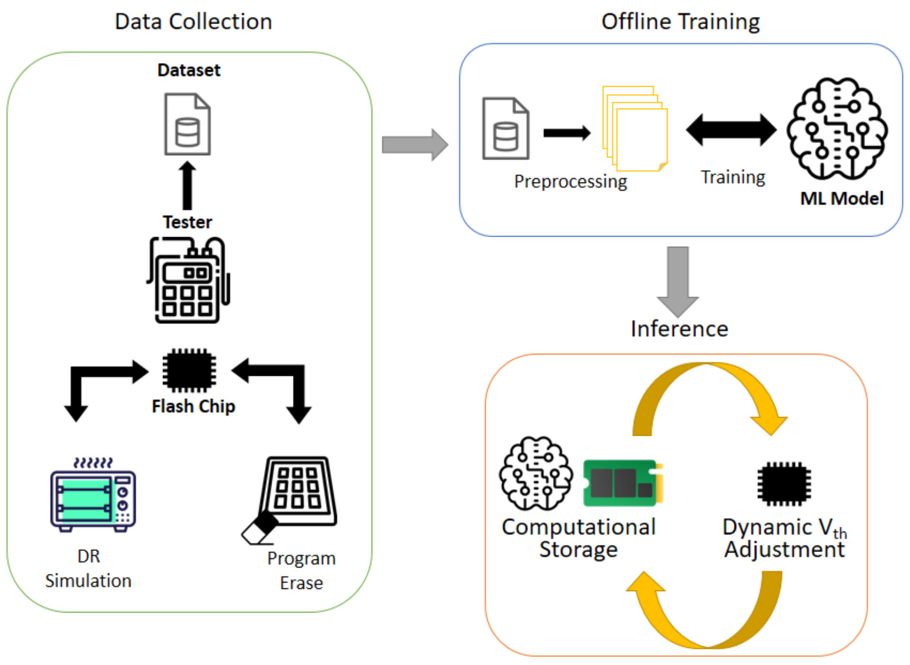
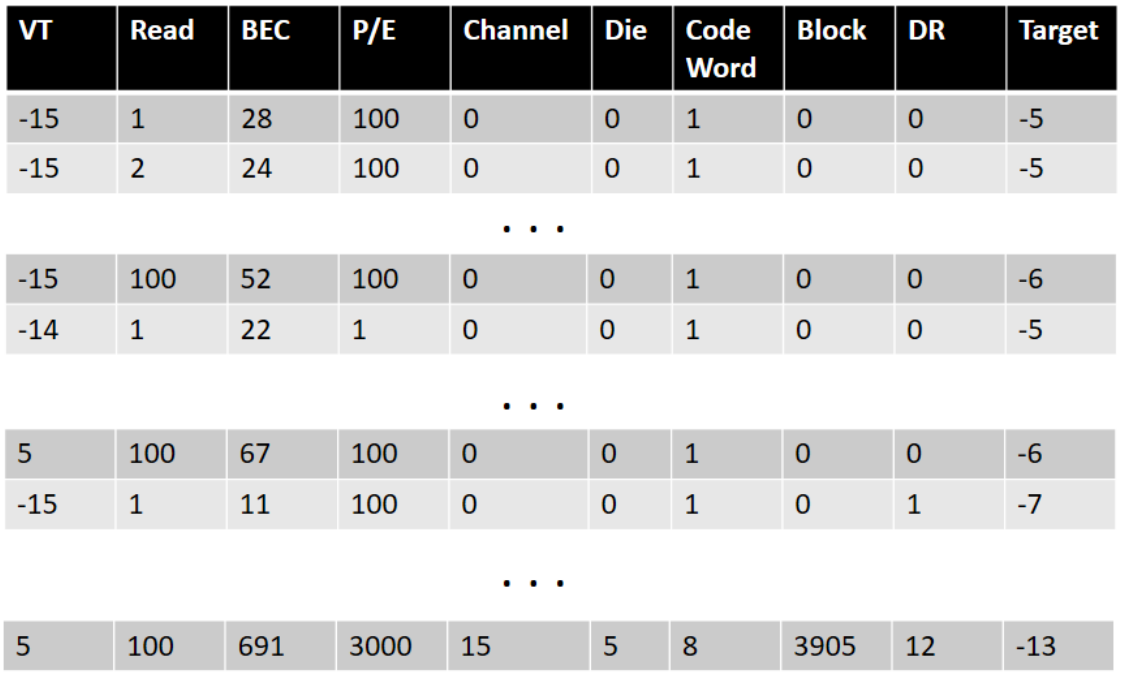


Figure 3.1: Data collection, model training and inference with a CSD.

Figure 3.1 shows the three stages of our approach to dealing with the problem. We start with data collection by using a tester to properly collect BEC metrics while emulating DR and cycling over P/E. Since we are treating the problem as supervised, we also collect our ground-truth targets at this stage. Afterward, we move on to offline training by preprocessing the dataset into features and training the ML model. Finally, we perform inference using the pre-trained model to adjust V_{th} dynamically.

3.3.1 Dataset Collection



VT	Read	BEC	P/E	Channel	Die	Code Word	Block	DR	Target
-15	1	28	100	0	0	1	0	0	-5
-15	2	24	100	0	0	1	0	0	-5
				• • •					
-15	100	52	100	0	0	1	0	0	-6
-14	1	22	1	0	0	1	0	0	-5
				• • •					
5	100	67	100	0	0	1	0	0	-6
-15	1	11	100	0	0	1	0	1	-7
				• • •					
5	100	691	3000	15	5	8	3905	12	-13

Figure 3.2: Tabular example of all the collected metrics available in our dataset.

Before we start describing our official dataset procedure, it is essential to describe the intuitive thought behind the collection procedure. The intended goal for generating a dataset is to provide similar enough data to what it will be found in the field. The term in-the-field relates to flash media being deployed in the real world and being utilized by a user. As we have discussed, several factors may change memory cell internal voltage distributions. Therefore, as for most learning applications, we would like to have a consistent dataset that replicates possible flash media behaviors or that we could derive such behaviors from the data collected. Another critical point is to provide enough data to perform our training, validation, and testing (inference) stage procedures.

With that in mind, we always looked back into what an SSD firmware log could inform us whenever a read was performed to keep our dataset in line with the in-the-field case. As we previously discussed (Section 2.1.2), one of the primary metrics for knowing how far away we are from the correct V_{th} is the BEC which is known by the firmware via the return of its ECC engine. Therefore, BEC is to be our primary source of knowledge as to the current state of flash. To properly collect BEC, we performed 100 consecutive reads at specific flash locations and collected their BEC. By performing these reads, we can also find the physical channel, die, block, and codeword, all of which the firmware also needs to know to perform a read, therefore reachable in the field. Another essential metric we can collect on a read is the current P/E of the block as the firmware keeps track of it to perform several functions such as Wear-Leveling.

Another critical control measure we must have is DR. As we previously discussed, it is a major cause of negative shifts in a memory cell. As we will be emulating DR, it is easy to know how much time has passed and include it in the dataset. Even though the SSD firmware might have access to a time-clock, it may be unclear if their time notion is correct, as drives may be turned off and calendars may be wrong. Therefore even though we have it in the dataset, we will not be using it as model input, as we may not be able to find reliable DR values [30]. However, we are getting ahead of ourselves, model input features will be better discussed in the next section.

Since we are dealing with a supervised regression problem, we must find the optimal V_{th}. Similar to what was described in some previous work [66], one way to find the correct threshold is to execute continuous re-reads with different V_{th}. Therefore, we took our 100 reads and repeated them at the exact location within a fixed range with the normalized values of -15 to +5. Organizing the data in this fashion allows for searching what V_{th} is currently minimizing the read or the die or block at the current P/E and DR.

Figure 3.2 shows an example of our dataset in a tabular format. The final dataset consists of several code words from blocks, four dies, and 16 channels subjected to successive phases of P/E cycles characterized as Beginning-of-Life (BOL) and Middle-of-Life (MOL), followed by weekly data retention for up to 12 weeks of retention time. We selected a normalized V_{th} range, iterating values one by one from -15 up to +5. Every block was read 100 times for every week, P/E cycle range and V_{th} which returned a maximum correspondent page BEC. All data was analyzed, and any significant discrepancies or errors were cleaned before performing any specific model procedure. Note that the presented target is representative of the optimal V_{th}.

Raw 3D CT TLC NAND Flash chips were characterized using a JEDEC-based cycling test up to the 3D TLC NAND endurance rating and the enterprise data retention requirement of up to 12 weeks at 40C. We used a 3D CT TLC NAND Flash memory chip with 96 stacking storage layers, 18 MB block size, containing 1152 logical pages with 16 KB per page. To perform P/E cycling, a script of block program and block erase using random data was performed. The blocks were subjected to successive phases of P/E cycles and DR of up to 12 weeks. That of which totaled approximately 45 GB of data.

3.3.2 Machine Learning Implementation

When first evaluating the problem, we came across the decision of choosing between classification and regression algorithms. The primary source of doubt came as voltage values are continuous, but when applied in this scenario, they are used as discrete integer values, as we can see by our set targets in 3.2. Still, It is known that the closer the voltage value we choose is to the optimal V_{th}, the lower the BEC is likely to be. Therefore, there is a correlation between closer BEC values in the dynamic threshold adjustment problem, which goes more in line with a regression problem than dividing it into discrete uncorrelated classes. When applying the voltage, we can truncate or round the algorithm's predicted values to force a discrete V_{th}.

As we detailed in the background Section 2.1.1.3 one of the leading causes of V_{th} shifts is DR. To recap, DR is related to the amount of charge the floating-gate can hold over time. Electrons may tunnel outside their intended location and shift voltage distributions around. There is a clear correlation between time and DR, as we even define DR by the amount of time passed.

Although P/E may not be directly represented in a time format, it is clear that it does hold some time-related events as the continued usage of the device will increase the number of programs and erases its components. P/E is constantly associated with a flash life cycle as a way of indicating the life stage of the flash [36].

With that in mind, the passage of time must be incorporated by the model inputs in some capacity. Therefore, we will navigate through the dataset and use consecutive reads metric information to create our model input time series as it should be of great help in noticing how BEC behaves as we perform reads with different or equal V_{th} values.

To properly evaluate a solution to our dynamic V_{th} adjustment problem, we implemented three different ML regression algorithm classes with different characteristics, previously detailed in Section 2.2. The three algorithms are XGBoost (XGB) [51], Regression WiSARD (ReW) [48] and DNNs using LSTMs as a way to better support time-series data. Many processing steps in creating the proper model input are similar to every model. We will detail them and their specifics in the next subsections.

3.3.2.1 Pre-processing & Features

We want to start this section by providing a discussion in which we bring the matter of physical levels to the reader's attention. From what we previously explained, our dataset consists of reads being performed at around the page level, which provides several reads for each block, die, and channel. When deploying a V_{th} adjustment, we must carefully consider in which of these levels we would likely be applying the algorithm. Changing V_{th} for every individual cell is infeasible due to their sheer amount. Therefore, a lot of discussion time was spent choosing an appropriate granularity to apply our algorithm. We believe a Channel granularity to be too coarse, as we may encompass the whole chip or more depending on the SSD architecture. Block granularity is in the thousands range, which is still too much to keep track of. At last, we came across the Die granularity, which generally is around 4-16 dies per chip and seems to be a nice number to predict upon.

Now that we have defined the physical level of flash media we will be acting upon, our thought process goes back to *how would the firmware feed me die-level information?* To answer that question, we got to probe flash and firmware and learned that SSD firmware usually measures BEC by passing up their maximum value across different levels. In practice, when reading n pages in a block, we would get the maximum BEC of the n pages from the firmware. Therefore, instead of multiple n BECs, the firmware will return one value, the highest (or worst) BEC, that comes from one of the n pages read. This is an essential finding as it allows for a realistic creation of a die-level dataset.

Algorithm 1 Algorithmic procedure to filtering the dataset into die-level

```
procedure DIE-LEVEL DATASET($dset$)
    $die\_dset \leftarrow Empty$
    for all DR, PE, Channels and Dies do
        $f1 \leftarrow$ Filter $dset$ by current DR,PE,Channel and Die
        for all Reads and $V_{th}$ in $f1$ do
            $f2 \leftarrow$ Filter $f1$ by current Read and $V_{th}$
            $row \leftarrow$ row of $f2$ where BEC == max(BEC)
            $die\_dset$ appends $row$
    return $die\_dset$
```

The algorithm 1 displays how to properly transform our dataset into a die-level dataset. We must first select the current PE, DR, channel, and die we will work on. We also filter by channel because channels are viewed as a level above the die in the hierarchic flash architecture. Therefore, if we go straight for die filtering only, we will have a die number representing multiple dies from different flash chips in different channels, which is not ideal as they may not have significant correlation. The next filtering step is to select the current read and V_{th} we are currently going to maximize. After this last step, we should have the ith read performed with a specific V_{th} for all blocks and codewords available at the current PE, DR, channel, and die. We will now pick the maximum BEC of all the blocks and codewords and put that in the new die-level dataset. After finding the max, the block and codeword dimensions can work to show from which place did the maximum come from for that read.

After correctly processing our data into a die-level dataset, we can start to figure out which features to use. From the collected data, we know that every row in our set can be obtained after a read occurs. Our two main essential features are the V_{th} used for reading and BEC, both of which will reflect on each other and show an indication of how the current flash state may look like. Another essential feature is the P/E count since we know it is an important factor for learning about flash life stages. Although DR would be fascinating to use, for reasons we previously explained (Section 2.1.1.3), it might be hard to collect at run-time properly; therefore, we will not be feeding it into the model.

The other features left, such as die and channel number, are categorical and will not be used explicitly as input features. The reasoning behind this choice is that we would be inserting input bias into the training process. For example, suppose that in most of our chip data measured, channel 15 die 1 has a peculiar behavior due to its manufacturing procedure which deviates from the others. When training the model, it might derive that all SSDs with channel 15 and die 1 will have erratic behavior in that location, which is unrealistic. Therefore, learning a generic representation without full location knowledge helps avoid biased learning.

Due to time-related reliability issues, we must remember that we will deploy our models as a time-series regression algorithm. Therefore, every sample consists of s consecutive read metrics collected from our die-level dataset, where s is the time-series length. The metrics are the V_{th}, BEC, and P/E.

Instead of directly using the raw P/E cycle count integer value, we applied a bucketing technique to transform it into a 3-bit one-hot vector. Each bit correlates to a different flash-life stage: BOL, MOL, and End-of-life (EOL). The goal is to help avoid model overfitting and allow for support prediction for different memory cell technologies as they greatly vary in order of magnitude for maximum P/E count support, as presented in the background Table 2.1.1.3.

Besides these common steps for feature selection and processing, every model has its peculiarities that must be attended to.

One standard procedure that greatly helps DNN convergence is to normalize feature inputs as it helps with numerical stability. After initial tryouts with different techniques, we settled for a Standard Scaler to limit values between 1 and -1 while keeping the standard deviation as close to 1.0 as possible. The algorithm subtracts the value of the current sample from the mean acquired from the training data, followed by the division of the training set standard deviation. The output is the scaled feature sample. Due to P/E being three binary values, it does not require any scaling. On the other hand, BEC and V_{th} will be normalized by a separate scaler with their own mean and standard deviation based only on their own BEC or V_{th} respectively. We did not apply normalization procedures on XGBoost and ReW.

It is important to mention that XGBoost does not have natural support for time series, which requires us to modify data dimensions before feeding them into the model. Every input sample we generate is a matrix with size $seq \times f$, where f is the number of features. Therefore, we must flatten it to a 1-D array of size $seq * f$.

We implemented a circular thermometer with a size of 128 bits for each of the features in the time series for ReW since it requires inputs to be binarized. A circular thermometer uses a min-max threshold while avoiding sparsity by keeping the word number of zeroes and ones equal. We also apply array flattening for ReW due to its pseudo-random mapping.

3.3.2.2 Training

The training process is done offline with data collected beforehand, as seen in Figure 3.1. Since we are dealing with a supervised problem, we must find the optimal voltage shift for every fed sample during training. To achieve this goal, we will look into every read performed within a location, i.e., chip and channel, and find from our V_{th} range of -15 to +5, which V_{th} returned the lowest BEC. This approach allows us to find the ground truth target V_{th} for every read. However, looking only at the BEC generated from a read to find the optimal might create targets constantly changing in a short read spawn. When performing a read, natural noises might cause different V_{th} values to intercalate between minimal BECs constantly. To provide a more constant target voltage value, we apply a simple moving average with a window of 10 over all the BECs to smooth out potential noises. Since a time series consists of several samples from different reads, we will use the last read as a reference for the target value fed into the model.

Both LSTM and XGBoost will use a Mean Squared Error (MSE) loss with the goal of increasing network penalty as the error between prediction and target increases squared. As explained earlier, the closer our V_{th} value is to the optimal, the lower the number of bit errors. ReW does not require a loss function to optimize.

Now that we have defined the specifics and how each feature looks after the preprocessing, we are left over with how to concatenate different read metrics to build a sample for the time series. Whenever we train the networks, we should always aim to use data that will look like the data available at inference time. Therefore, it is imperative to describe two different scenarios and how they might influence our training input generation.

First, we have the scenario where V_{th} are not always requiring change. For example, a write-heavy workload where data is constantly being freshly written, erasing DR, or the case where the dynamic threshold algorithm is barely used, rarely switching V_{th}. The SSD performs several reads with the same reference voltage value in these examples. For this case, our time series of *seq* reads would mostly have the same V_{th}. This training set is relatively straightforward to collect from our dataset as we need to build samples with the same voltage from consecutive read values, similar to how the Figure 3.2 shows the sorted data. We will refer to this set as the *EQUALVT* training set.

In our second scenario, we have a more dynamic system, where a dynamic threshold algorithm is changing V_{th} and the model inputs for inference may be performing reads with different V_{th}s at different DR. We must first choose our starting V_{th} and read to generate these samples from our dataset. Next, we randomly select a DR equal or older for the next read from the same time series. Our implementation draws from a triangular probability distribution allowing a higher chance of selecting DR equal to or a few weeks close to the previous sample DR. Read operations with closer DR values are more likely to occur instead of dies not being read within several weeks. After choosing the DR for our subsequent read, we randomly choose the next V_{th} to be equal or smaller. The choice of a smaller V_{th} is due to P/E and DR shifting voltage values to be more negative. Therefore if a dynamic threshold algorithm changes V_{th} as we progress through older DR, it is more likely to be for equal or smaller values. By knowing where we are in terms of physical location, DR and V_{th} are easy to filter in our dataset and extract the other features such as BEC. We will refer to this scenario as *DRFWD* set, as the read inputs from the same time series navigate to older DR values.

It is important to mention that the reads of the same time series will never have data from different dies or channels in any of the described scenarios. We will evaluate how each set changes the model capabilities to predict the appropriate V_{th} in Section 3.4.

3.3.2.3 Inference

As we try to nail the point, emulating data that is collectible in real-time is essential and one of the key distinct contributions of our approach. Still, it is required that the SSD firmware exposes our required features (V_{th}, BEC, and P/E) to the model. Unfortunately, firmware code is proprietary, so we cannot modify it to perform experiments in the field properly. Therefore, all the values used for testing are from our die-level dataset, which still consists of actual data from flash media. Note that all preprocessing steps remain the same for each model during inference, and we ensure training data does not leak into the testing data.

With that in mind, to properly emulate a prediction study case with our collected dataset, we developed a *convergence* testing set. The idea is to emulate how we would effectively test our trained ML models in an SSD. Note that we will be referring as if we were performing read operations on the drive, but we are just filtering the tables of our dataset to collect the data correctly.

We start by fixing our working test zone defined by the physical location (channel and die) we will probe. This strategy allows us to exclude these samples from the training set, avoiding contamination of the testing set. We then select our initial V_{th} to perform the first read as if the data was freshly written (DR of 0 weeks). The baseline usually defines the initial voltage threshold value. We collect seq reads from the current physical location we probed. This sequence of reads will create our first time-series input to infer upon which the model will receive and predict a V'_{th} in the process. Therefore, the next read is performed with V'_{th}, which generates our next input and so forth until we reach the maximum amount of available reads and DR.

During our inference stage, we will proceed to perform a K-fold-like training and testing procedure using the steps described above. Whenever we test at a specific channel and die, we train a model with all the data except the tested channel and die. The values presented in our experiments are measurements done for all available channel/die combinations.

The convergence testing approach allows us to visualize how the model will behave at different stages and, more importantly, how it predicts on top of its previous predictions, allowing for some room and self-adjustment in case of making a mistake in the previous read. We believe this testing set provides a more realistic case of study than just independently feeding randomly sampled inputs without any correlation.

3.4 Experiments & Results

We propose a set of different experiments divided by P/E and DR to evaluate the proposed ML models' behavior in different flash life stages. We aim to evaluate reliability metrics pertinent to our SSD environment and find where the models may fit best.

Besides the previously described models, we also measured a simple linear regression algorithm and a baseline approach that keeps the standard starting V_{th} throughout all P/E and DR stages. Moreover, we also plot the perfect dynamic threshold adjustment scenario where the optimal V_{th} extracted from the ground-truth targets is always applied. The plots are divided between BOL and MOL P/E stages and DR. Average plots will provide the average of all available reads in P/E, DR, die, and channel. In contrast, maximum plots represent the worst prediction of the performed reads. The starting point of the ML models at freshly written data (1st read of 0 weeks) will always be the baseline of $V_{th} = -5$.

Experiments were conducted on an Intel Core i7-6700 at 3.40 GHz, 32 GB of DRAM, an Nvidia GeForce GTX960-4GB, and a Ubuntu 16.04. We also deployed the Newport CSD to measure time and memory as to provide a discussion on how each model may me applied. We have described Newport architecture in detail in background Section 2.1.3. We did extensive hyperparameter optimization on the three proposed models. Appendix Section A.1 provides Tables A.1.1, A.1.3 and A.1.2 with the selected parameters.

It is essential to mention that we will also be evaluating different time series lengths of 1,3, and 9. Note that a size of 1 is not a time series per se, which causes some complications for our DNN with the LSTM ML model as LSTM layers require a 2D matrix with more than one row. Therefore, to test DNNs on time series of length 1, we deployed a standard fully connected DNN with Relu activation. Moreover, we also look at the difference between using the two previously mentioned training datasets in Section 3.3.2.2.

The rest of this section will be divided into different discussions and experiments. We first will present and justify the reliability metrics that will be used throughout the experiments. Afterward, we will get into our first experiments, which discuss model behavior on different time-series lengths. Our second evaluation will compare models at their best time series lengths and discuss which scenario they might fit best. We later also provide an insightful discussion on the different execution times and memory consumption of the proposed ML approaches.

3.4.1 Evaluation Metrics

To correctly measure the efficiency of the proposed models in finding the optimal V_{th}, we introduce two metrics: Failure Rate % and RBER %.

RBER is a standard metric deployed in prior work to study reliability issues [33, 36]. To calculate RBER, we must divide BEC by the number of bits read that is dependent on flash and SSD architecture. In our case, we use the value of 18336, which is correlated to the chips and tester used to collect the data from the dataset. The BEC value is the number of bit errors when reading the specified location with threshold V_{th}. We will evaluate the average RBER and the maximum RBER case. A reminder that the maximum RBER case displays the worst single prediction performed by the model at the current data retention week for all available dies and channels.

Ideally, we would like our model to always find values very close to the optimal reference voltage threshold. However, there are instances where a corner case or outliers may come by and throw the model's prediction off. In the problem we are dealing with, a wrong prediction may cause a high enough BEC that the ECC

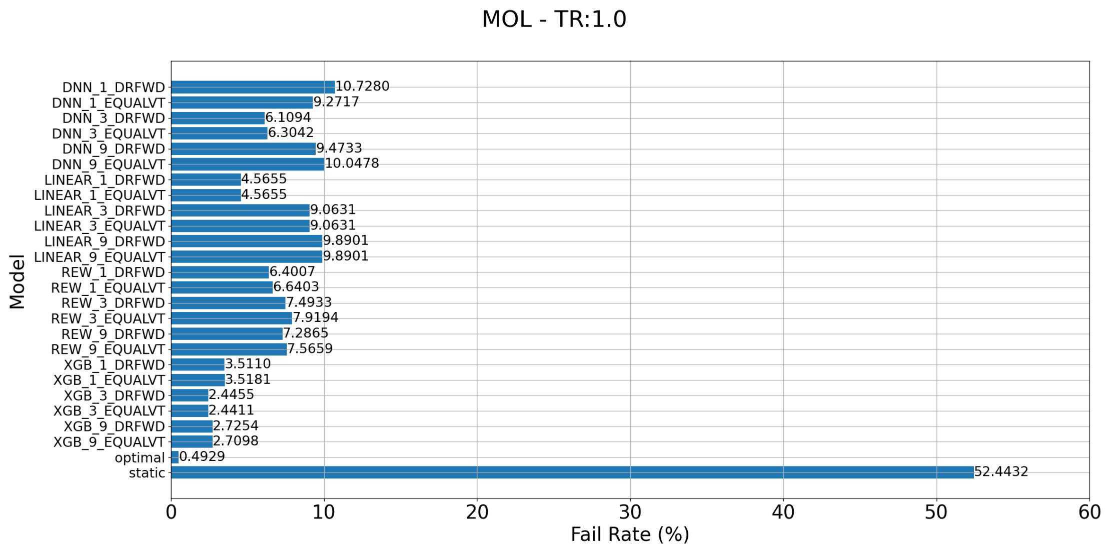


Figure 3.3: Failure Rate for all model types, time-series lengths and training sets with using TR of 1.0 and MOL Data.

cannot correct, generating an uncorrectable, increasing read latency or, even worse, complete data loss. Therefore, studying how often a miss prediction causes this scenario is essential when evaluating performance. In fact, that is the main reason we also evaluate the maximum RBER case explained in the last paragraph. Our Failure Rate metric gives us significant insight into how often the model fails to perform a successful read due to a miss prediction in our testing set.

Failure Rate % is calculated by selecting an RBER threshold TR and counting the number of predictions going over said value. The TR represents different ECC capabilities in RBER format. The higher the Failure Rate threshold, the stronger the ECC. Therefore, if a prediction causes an RBER value higher than TR, it is a sign of an unsuccessfully read. The tables will show the percentage of times every model made a prediction that went over a TR.

3.4.2 Training Set Evaluation

In this section, we will be evaluating how each of the proposed training sets *EQUALVT* and *DRFWD* impact Failure Rate %. Our selection of failure rate as an initial filter before diving into a DR-based analysis is due to the importance of worst cases and uncorrectable reads. In essence, if the model is more likely to generate a prediction over the ECC threshold, it will cause an uncorrectable, which will trigger different SSD routines such as WL, GC, or read-retries which result in extra overhead. There could even be an argument that uncorrectable reads may cause blocks to be retired early, diminishing SSD lifespan.

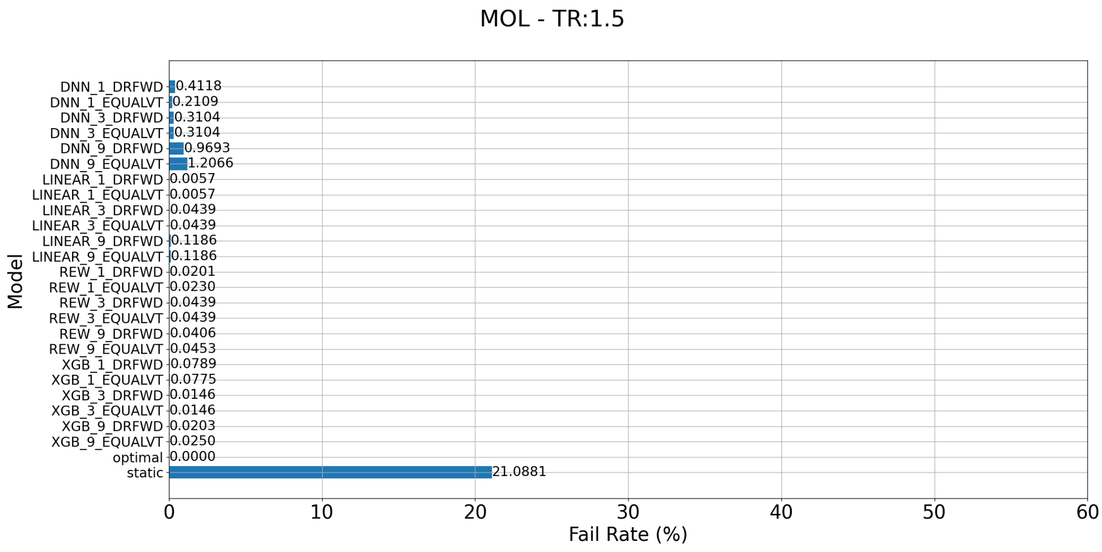


Figure 3.4: Failure Rate for all model types, time-series lengths and training sets with using TR of 1.5 and MOL Data.

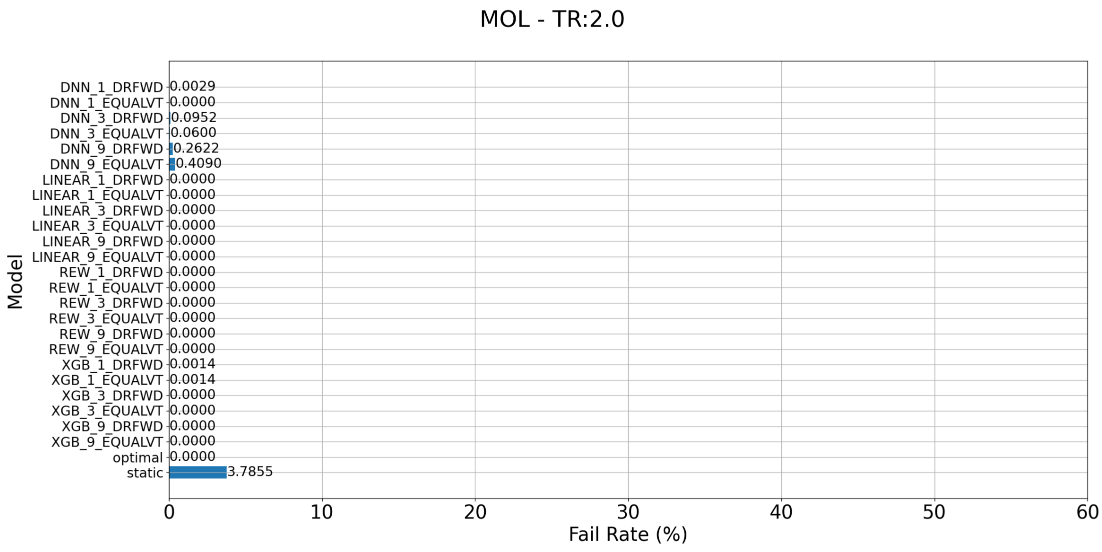


Figure 3.5: Failure Rate for all model types, time-series lengths and training sets with using TR of 2.0 and MOL Data.

Figures 3.3, 3.4 and 3.5 present the failure rate for different threshold values in a MOL case. These values are for all predictions made within every DR, channel, and die available. The optimal predictions have the lowest failure rate for every TR, but notice it is not zero for $TR = 1.0$. This rate higher than zero means that even by setting optimal V_{th} , an ECC with these capabilities might never be able to fix these bits. These can be referred to as unavoidable uncorrectable errors. The static approach is also the worst case for every TR, which is expected. As DR moves forward, the initial threshold becomes obsolete.

It is interesting to see that the training set of $DRFWD$ is achieving a better percentage for most of the time-series and model variations. Due to a more heterogeneous formation of samples with different V_{th} values, it might be helping models learn more robust internal representations of the problem. However, there are still a few instances, like in XGB-3 and XGB-9, where the $EQUALVT$ training approach performs slightly better. Note that the simplistic LINEAR classifier does not change independently of the training set since it has a lower learning capability.

For BOL Fail Rate (%), all model predictions achieved 0% Fail Rate except for the *static* baseline. This result is due to the flash memory cells being mostly new, therefore not showing meaningful read-level threshold voltage shifts that would generate noisy samples that may cause mispredictions.

Based on these findings, in the following sections, we will compare the models and time-series variations by using the training set, which achieved the lowest Fail Rate % in the evaluated model type and time series length.

3.4.3 Time-series Comparison

To study how time-series impact model prediction, we will look at each proposed model individually by varying the time series lengths to 1, 3, and 9. Appendix A.2 also provides additional plots discussed during this section using the best training sets based on the previous section discussion.

For DNN approaches, we can see more clearly in Figure 3.3 that a time series of length 3 provides better failure rates. Moreover, DNN-1, which uses a different architecture than the LSTM, seems to be having a more challenging time for $TR = 1.0$ while performing better at the other two thresholds. When devoting our attention to RBER per DR in Figure A.1 the average case seems to reassure us that the length of 3 is the best approach, seen more clearly for MOL. However, time-series of 1 showed more stable results, presenting fewer outlier predictions on the worst-case plots. This result indicates that the DNN model can provide better predictions using time series as input. However, it may also partially hinder its ability to deal with outliers as the worst predictions for DNN-3 are causing higher

RBER than DNN-1.

The Linear models are just simple linear equations tuned for the dataset. Therefore they show the least amount of variance over training sets. Its main advantage is being fast and providing transparency and predictability to the user. In Figure A.2 there is no discernable difference between time series lengths for BOL. For MOL, where there is a bigger variance within the data, LINEAR-1 is performing better, which aligns with our findings in Failure Rate %.

Figure A.3 shows the max and average plots for its tested time series sizes. There is a small difference between the models in the average plots, especially for lengths 3 and 9. The worst cases also provide similar results as the same lengths provide somewhat robustness by not making wildly wrong predictions. However, Failure Rate % seems to give an edge over XGB-3. It is interesting to point out that even though XGBoost has no natural support for time series, it still was able to achieve better predictions by using more feature information.

REW also provides highly stable results as per Figure A.4 it barely perform wild V_{th} mispredictions that cause a significant RBER. Similar to the LINEAR model, we can better see the differences when looking at the MOL instead of the BOL case. This difference is due to the binarization of inputs required by the model, which causes information loss. When dealing with BOL, the differences between BEC are way smaller than for older flashes such as MOL and EOL. Therefore the inputs may look more similar due to information loss, which generates similar predictions. However, looking at Failure Rate % and the average case of MOL, we can see that not using time series seems to provide the better as the data retention weeks rage on.

3.4.4 Model Evaluation

Based on our findings discussed in the previous sections, we selected the best time series and training set length for each model type and compared them with the baseline and optimal in Figures 3.7 and 3.6. Note the different y-axis RBER % ranges between the plots.

The baseline approach achieves the worse average RBER as DR moves forward. Moreover, it is way worse for MOL, as seen by the 10th-week baseline value in Figure 3.6. Every model achieved a better RBER than the baseline when looking at the average results, except for the first few weeks. The poor baseline result is due to V_{th} usually being selected with the intention of optimizing the read-level thresholds for the initial weeks of DR, as seen by its close proximity with the average optimal in Week 0. Arguably, since DR effects are pretty low at this early stage, this is the one place mispredictions may not cause much harm as they would not achieve high

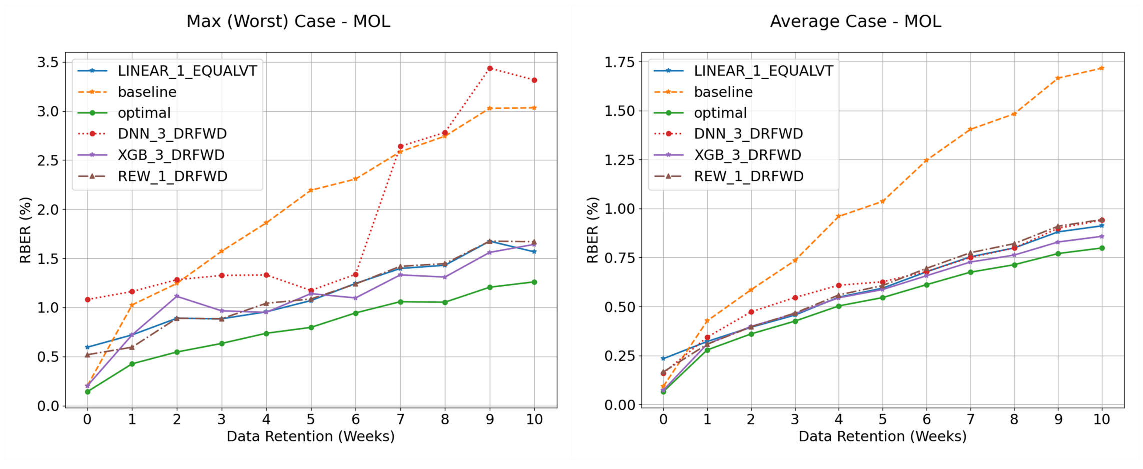


Figure 3.6: Model comparison in MOL flash data based on best performing time-series and training sets by model.

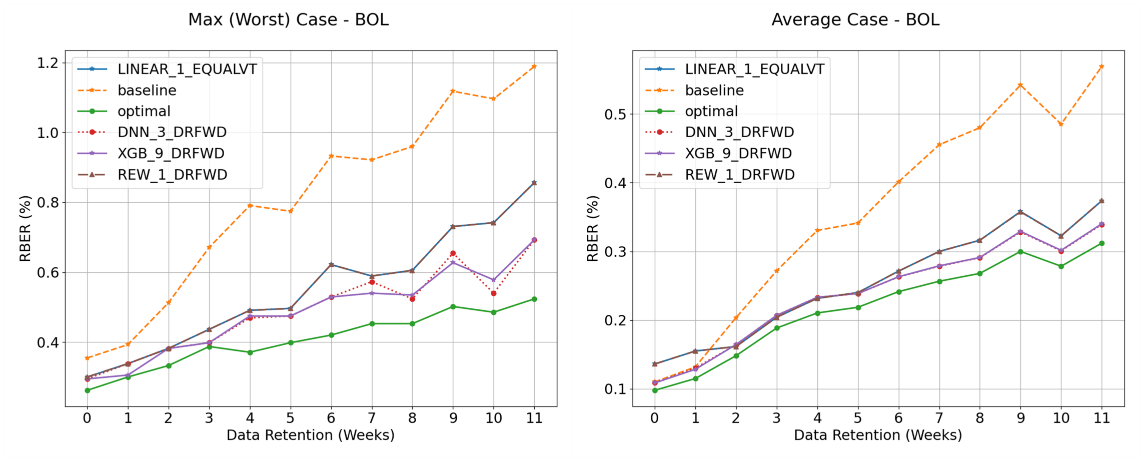


Figure 3.7: Model comparison in BOL flash data based on best performing time-series and training sets by model.

Model	Pred. Time	Memory
XGB-1	0.1536 ms	367 MB
REW-9	0.2885 ms	342 MB
DNN-3	2.0317 ms	537 MB

Table 3.1: Prediction time of one sample in milliseconds (ms) and approximate memory consumption in Megabytes (MB).

RBER.

As the weeks go by, most models keep close to the average optimal, achieving significantly better performance than the baseline. Moreover, XGB-3 seems to excel over the other models in MOL, as seen in the final progression of weeks. Interestingly, when looking at the max still in Figure 3.6, it is also the only model to always be behind or equal to the baseline approach, even at 0 weeks, where most models seem to struggle at. In the max case, DNN-3 is the one that struggles the most, achieving worst cases worse than the baseline for the initial weeks and the final four weeks as well. On the other hand, REW-1 and LINEAR-1 achieve a more predictable prediction pattern, which goes in line with the previous visualized plots. REW-1 seems to have a slighter edge as it performs better in the initial weeks.

In the average case for BOL in Figure 3.7, both DNN-3 and XGB-9 achieved better RBER with their predictions. Even though the average baseline case is performing better than a few model models, the Max plot shows that the worst predictions made by the baseline are achieving way higher RBER than the worst cases for every model throughout all available DR weeks.

Even though XGBoost seems to be the primary option, different time series sizes were required. Therefore a model for each flash life stage is ideal. Moreover, for more stability and faster training, both REW and LINEAR models showed promising results. Since flash media's life stage is usually indicated by P/E, which is actively tracked by the firmware, it should be feasible and straightforward to use different models at different stages.

3.4.5 Execution Time & Memory Footprint

Table 3.1 shows the inference execution time per sample in milliseconds (ms) and the approximate memory consumption for performing inference in megabytes (MB). All measurements were performed in the Newport CSD (Section 2.1.3) and are an average of 10 executions. Even though we are using the information from our dataset files, getting a grasp of memory consumption and execution time is important to learn more about environment required to perform dynamic threshold adjustment using the proposed models.

To correctly measure memory, we used a python *memory profiler* library [2] which measures memory throughout the execution of the program. To avoid interference, we had a pre-trained model that was then loaded into the main memory while deleting and forcing python's garbage collection to clean up unnecessary lingering objects. When we stop to evaluate each measurement, we notice the DNN-3 using LSTM layers is consuming more memory than the others. When we stop evaluating the architecture, we can see that the amount of parameters on the structure is significantly larger, which justifies the higher consumption from the model. XGB-1 comes in second place very close as we can limit the maximum amount of trees during training. REW-9 requires a bit less memory than other approaches because it greatly improves the WiSARD original architecture by only requiring one discriminator.

It is essential to mention that all these models were optimized for improving RBER. Therefore other implementations and techniques may provide a better trade-off between an acceptable RBER while decreasing memory footprint. For example, in DNNs, techniques like quantization and pruning can be implemented to decrease their memory footprint with a low impact on prediction accuracy [72, 73]. Hashing techniques have been previously applied in WiSARD to decrease memory consumption as well [74].

Still at Table 3.1, we have the prediction time per sample in milliseconds (ms). The measurement includes the preprocessing time of a sample, following the described process in Section 3.3.2.1. It is important to analyze the execution time by including the preprocessing because as data comes in from the firmware, all the procedures to shape the input properly, be it binarization, flattening, or normalization, are obligatory for proper model functionality. Moreover, by evaluating it on a per-sample basis, we can evaluate the values without worrying about batch sizes or the number of available inputs the firmware may provide simultaneously for the model to predict. XGB-1 has the lowest execution time out of the three proposed models. However, it should be pointed out that most of the execution time is clouded by their input preprocessing procedures. For example, REW suffers from its binarization procedure and DNN from the normalization requirement.

Note that each model may be benefited from better parallelization tailored for the Newport ARM architecture. For example, compiling TensorFlow from source with support to ARM special instructions or through *ARM NN SDK*[3] could significantly boost DNN-3 execution speed.

[2]https://pypi.org/project/memory-profiler/

[3]https://www.arm.com/products/silicon-ip-cpu/ethos/arm-nn

3.5 Concluding Remarks

This chapter introduced an ML time-series approach to improving flash media reliability by dynamically adjusting read-level voltage thresholds. Through flash characterization collected in SSD-graded flash media chips, we produced a compatible dataset with different opportunities and realistic data framing according to firmware behavior and data availability.

Our evaluation consisted of choosing different ML models and implementing them in a time series environment according to how the data could be achieved in a real SSD. Our findings show that using ML to find the appropriate V_{th} can significantly aid in decreasing RBER in different P/E Cycles and DR scenarios over the standard baseline approach. Specifically, XGBoost achieved the lowest average RBER values. At the same time, faster approaches such as REW showed good resiliency to outliers, achieving better stability when looking into the worst predictions made by the models.

Chapter 4

Object Tracking in Computational Storage

4.1 Motivation

In recent times, the amount of data being produced has increased dramatically. Evaluating and processing such data efficiently to a meaningful end has become quite the challenge [75]. There is a critical need to maximize the capabilities of a system by developing scalable and robust techniques. The problem is especially prominent in handling visual data as companies like Google and Meta have been experiencing unprecedented growth in the production of photos and videos for years [76, 77] which is expected to increase with the introduction of the Metaverse.

Due to their recent success in learning about visual data, ML has significantly advanced a way of interpreting the visual data by understanding its context with high accuracy. Therefore, the area of Computer Vision involves processing large volumes of data in the form of videos and images. These applications are well known for having outstanding performance in many areas, such as security, health, and entertainment [78–81].

Object Tracking is a process of tracking one or more *regions of interest* (ROIs) through a video stream. Regions are typically selected a priori and are then followed throughout a set number of frames. This requirement makes object tracking less computationally intensive than object detection, which needs to detect one or more objects every frame without using positional knowledge from previous frames [82]. Depending on the purpose of the final application leveraging object tracking, basic requirements can vary. For example, the exact contour and shape of the object might need to be tracked (i.e., object segmentation). At the same time, in other cases, only rectangular bounding boxes containing the ROI are required. The performance, measured in frames per second (FPS), is a crucial feature that shows how fast the algorithm can process a set number of frames in one second. Unfortunately, developing general-purpose tracking systems could be complicated due to noisy frames, object obstruction, and other errors. These factors, coupled with noisy frames, partial or complete object obstruction, and other errors, can complicate the development of general-purpose tracking algorithms.

According to the IDC [83], the sum of the world's data will exceed 175 zettabytes by 2025. Therefore, storage I/O bandwidth limitation could create a big problem for the AI applications that must interact continuously with storage systems. Furthermore, this problem gets accentuated with Next-Generation Networks (NextG), which are rumored to rely heavily on IA as well as IoT and Edge deployments [84, 85]. Computational Storage [86, 87] comes in as an important player to allow flexible deployment of extra computational power for both data center and edge devices. By providing in-storage computational power, preprocessing steps and entire applications could be digested before being fed to the main CPU. This computation enables the host resources to be better utilized and scale across a larger number of workloads. In fact, taking advantage of CSD capabilities has been studied in different applications before [11–15, 88–93]. Considering the trend toward larger video frames with 4K or 8K resolutions, we may not be able to fully store a set of frames into host memory, making in-storage processing an attractive alternative for computer vision problems.

In the work presented in this chapter, we provide a general study of different performance metrics for several object-tracking ML applications. We carefully selected three distinct classes of algorithms to evaluate how they would perform in an in-storage-prone scenario and what are the trade-offs at play. The algorithm classes are as follows: Correlation Filters (CFs) [80, 94], DNNs [79, 95] or WNNs [81, 96, 97]. Many of which have already been deployed successfully in an object tracking environment. We aim to explore how computational storage can be effectively integrated with conventional setups to improve the overall system gains.

The reader can find our contributions highlighted below:

- Algorithm modifications for processing datasets in-storage;
- Evaluation of throughput for several object tracking algorithms in a CSD;
- Study and Discussion on Energy-Consumption for different devices executing object tracking.

The rest of this chapter is organized as follows: Section 4.2 will briefly discuss some prior work conducted with CSD. Section 4.3 discusses the implemented algorithms and our choices behind their deployment. Section 4.4 provides experimental results for FPS, object tracking equivalent metric for accuracy and power consumption. Section 4.5 concludes the chapter. Some of the results of this chapter were published in Do *et al.* [1].

4.2 Related Work

In this section, we will gloss over some prior work that used the processing capabilities of computational storage to improve parallelism and lessen data movement.

Mahdi *et al.* [92] work introduces Newport's predecessor, Catalina, which was the base for the current Newport architecture. It evaluates 1D, 2D, and 3D FFT operations on several drives by looking at their power measurements through Joules and execution time in seconds. Our work performs similar performance measurements but focuses on a more intensive application while taking into account accuracy metrics.

It has been shown that deploying hardware accelerators, like FPGAs, is an up-and-coming solution to gain both performance and energy efficiency [98, 99]. Some CSDs have enabled their computational capabilities through FPGA devices [100]. The downside of FPGA-based CSDs is the deep knowledge required for developing applications for such devices, usually requiring a different set of skills related to hardware. Moreover, there are instances where general-purpose CPUs perform better due to their higher frequency (GHz) when compared to FPGA devices (Mhz) [101].

Cao *et al.* [102] deploys CSDs into Alibaba Cloud to offload computation from the host CPU. The cloud has its relational database system called POLARDB, which is modified to perform table scan tasks inside the CSD. The developed support inside POLARDB's entire software stack enables an efficient distributed table scan pushdown. Their approach included optimizing their storage engine, the distributed file system, and the CSDs as the host-side manages them. Since their CSD supports computation via FPGA, they also provided a hardware implementation of their task that utilizes pipelining, ECC, and a memcopy module.

Kwon *et al.* [103] proposed a virtualization mechanism called FlexCSV. The mechanism aims to improve the performance and cost-effectiveness of CSD virtualization. FlexCSV engine is an FPGA card tailored made to achieve the desired goal. Moreover, it supports hardware-assisted virtualization and resource orchestration, later supported by an extended NVME protocol for in-storage processing. Their experiments show that the hardware virtualization approach achieves a speed-up at some points 3 times higher than full software virtualization. They also provide a few other experiments studying multiple CSDs and Virtual Machines and how that correlates to higher aggregated bandwidth.

Other works have gone a different route and focus on proposing better programming models for computational storage. Seshadri *et al.* [14] presents a PCIe-based generic RPC mechanism allowing developers to augment and extend the SSD semantics with application-specific functions easily. On the other hand, Gu *et al.* [89] proposed a dataflow-based programming model where tasks and data pipes can be used to construct an application to be executed in storage dynamically. Although these programming models offer great flexibility of programmability, they are still far from being genuinely general-purpose.

We also refer the readers to Barbalace *et al.* [104] work, which gives an overview of the current state of CSD at the time it was conceived (2021) as well as future questions and challenges in the road ahead.

4.3 Object Tracking for CSD

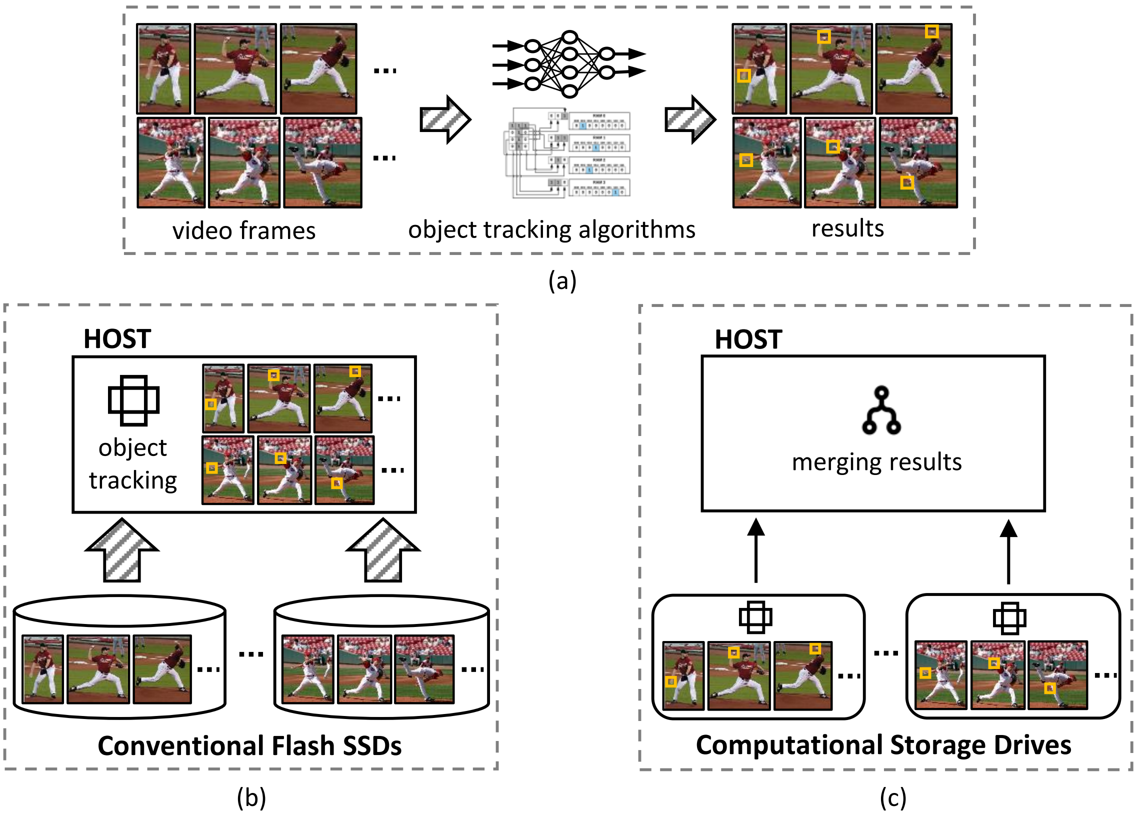


Figure 4.1: Object tracking general flow: (a) Generic view of how object tracking is performed via a black-box algorithm. Its result can be see by the yellow bounding boxes. (b) Usual flow of the object tracking algorithm in conventional storage media. Data is loaded from said media and the host executes the algorithm. (c) An example of a Flow for a CSD device. Data is loaded and processed in-storage, the host only merges the result and results data preprocessed. (Image from [1])

This section will discuss all implemented object tracking algorithms divided by their correspondent algorithm approaches in CFs, DNNs, and WNNs. In Figure 4.3.a there is an exemplification of how most tracking algorithms go about during execution. One or more frames are fed into a model which provides the coordinates indicating the position of the chosen object, known as a bounding-box.

Figure 4.3.b and 4.3.c shows the two different flows when using conventional storage means versus CSDs. Conventional Storage, in this case, pictures as a flash SSD, lacks any computation regarding the object tracking application. Therefore, they will only load and save frames while the host uses its devices to execute the model. CSDs are the opposite, where the algorithm runs inside the drives, processing different files or frames, and the host is responsible for merging their results. All algorithms were implemented and executed on different devices according to their characteristics to provide a better evaluation later (Section 4.4).

Before moving into the details and algorithms, we must briefly discuss our reasoning for choosing the different ML approaches. DNNs are a natural candidate due to their recent success in several computer vision applications. Unfortunately, object tracking specific datasets with the required data for training these networks is scarce. A common practice is to rely on pre-trained CNNs that learned from big classification datasets such as Imagenet. However, we must consider that to provide such high accuracy and robustness, the network must endure computationally expensive operations, which require potent devices, usually GPUs, to provide any online experience.

Other techniques, such as CFs and WNNs, follow different approaches requiring smaller datasets and allow for a higher frame per second (FPS) rate due to simplistic operations and structure. WNN provides online training by using previous frames to infer positions on the current frame. The network decides whether a frame should be trained if it realizes the object properties are changing. For example, a frontal view of an object may be highly different from a side view. CFs use pre-trained filters combined with a correlation operation on the Fourier domain instead of the spatial one to increase overall performance. The CFs and the WNNs perform a window search algorithm to look over the frame and find the object.

4.3.1 DNN Trackers

Both DNNs used in this work take advantage of CNN layers and the usual DNN architecture discussed in the Background Section 2.2.1. We chose three different networks with different attributes to properly evaluate their performance in a CSD environment.

When installing and deploying DNNs specifically, we encountered a few challenges due to a bug regarding Newport's Tunneling algorithm. The bug would not allow a large amount of data to be installed or transferred over from the host to CSD via tunnel. Therefore we had to develop ways of circumventing it by installing libraries required for DNN executions via other means, such as customized python wheels and source builds with special flags that were supported by the ARM architecture and did not trigger the problem. Besides these mishaps, we also had to adapt the network code flow to support our dataset, which holds for all other approaches as well.

GOTURN [79] is an object-tracking DNN with a CNN architecture followed by a few hidden FC layers totaling eight total layers. The network can only track one object at a time and rely on an offline training scheme with a subset of labeled videos. Besides videos, it also uses still images during training to learn a more extensive range of objects.

It will use previous knowledge from the last frame to track the object by cropping the past frame around the object's position. This position is widened by two times its size to allow for possible object movement in the current frame. This value can be optimized for faster moving objects as it might go over that region. The network is biased over a small motion of objects to provide a more smooth bounding box transition. It uses Laplace Distribution to randomly select cropped object samples with minor movement between frames as input during training.

As we previously mentioned, it is a common practice to use layers trained in Imagenet due to the small dataset site. GOTURN utilizes the same strategy for its CNN layers, based on CaffeNet [105] Architecture. Note that training only proceeds for the FC layers.

We first expected to have GOTURN using multi-cores since it is compiled with the CaffeNet framework where the multi-thread execution is enabled. However, that was not the case due to incompatibility issues. Therefore we implemented a multi-process execution. This strategy deploys several trackers in parallel as processes. Each is assigned a core to perform the tracking on a different video stream. Therefore fully utilizing CPU resources. For GPU, GOTURN's implementation executes as expected using GPU cores.

YOLO: To study the behavior of a different algorithm that could also be used for tracking, we decided to deploy a DNN of object detection. As we hinted earlier, the critical difference between detection and tracking is that tracking uses the object's previous location from past frames to infer its current location. In object detection, every frame is a new image that is swept through to find one or more objects while classifying them for some algorithms.

YOLO [95] is a DNN used for multi-object detection in real-time systems that can detect up to 9000 objects in a single frame. One of its key features is passing over a frame a single time to detect captured objects. We deployed YOLOv3, the most recent version of YOLO, at the time of the experiments. The network consists of a 53-layer deep CNN called Darknet responsible for predicting the bounding box of the objects. The input is split into grid cells before feeding it into the network to help with multi-object detection. The network may use the ground truth bounding boxes to give a confidence score related to its prediction location. The network also supports object classification by using a set of pre-trained layers. At the time of the experiments, YOLO was considered state-of-the-art and outperformed a set of other techniques regarding inference time.

As expected from its architecture, YOLOv3 is quite heavy computationally. Moreover, since it deals with a more intensive task in object detection, it was built and optimized for execution in GPU systems. With that in mind, we deployed a separate lightweight version of YOLO, called YOLO-Lite [95]. The network targets real-time systems with limited computational resources, like autonomous vehicles and edge devices, which is more characteristic of the CSD environment we portray. YOLO-Lite decreases the number of layers of the original network from 53 to 8, which sacrifices some of its accuracies, but allows for higher FPS and compatibilities with other systems.

Both YOLO implementations provide native support for multi-thread execution. Instead of disabling it to do similar multi-process execution as we did with GOTURN, we opted for keeping the native support as it is optimized internally and should provide better performance for the algorithm.

4.3.2 Correlation Filters

Correlation Filters are an online training algorithm that uses image templates as filters to discriminate through images [106]. Even though it might not produce the most accurate results, they provide high FPS. They perform correlation operations in the Fourier domain using a filter. It generally provides online functionalities and some online learning by updating said filter as the frames are processed.

Both CFs used the same multi-process approach as GOTURN to better utilize CPU resources through parallelism.

The Minimum Output Sum of Squared Error (MOSSE) Filter [80] is an object tracker based on adaptive correlation filters. As is usual for many online learning algorithms, it uses the first frame ground-truth bounding box as the label to initialize training. It applies random affine transformations that generate eight small perturbations inside the tracking window to generate our training set for that frame. Tracking is executed by correlating the filter with a search window on the subsequent frames. It minimizes the squared error sum between the algorithm output and the ground truth to find the best filter to apply. To speed up training, data is converted into the Fourier domain via Fast Fourier Transform to decrease the complexity of convolution operations as they become an element-wise multiplication. Another interesting feature of MOSSE is that it automatically detects if an object has left the frame and picks it back on in case it returns.

Kernelized Correlation Filters (KCF) [94] expands on the original KCF by allowing the algorithm to process multiple image channels. It applies a Gaussian Kernel on Histogram of Oriented Gradients features instead of raw pixels values to perform object tracking. Like MOSSE, it will take advantage of the Fourier Transform to reduce computation complexity significantly. Moreover, it also deploys a 'kernel trick' by representing inputs of a linear problem into a non-linear space via dot-products. Different from MOSSE, KCF does not provide any support for out-of-frame object detection.

4.3.3 Weightless Neural Network Tracker

WNNs [107] have been explored on object tracking tasks in several previous works [81, 96, 97]. We have introduced the WiSARD architecture and concept in the Background Section 2.2.2. Here we found similar complications regarding the OpenCV, which required a specific version and building from source to enable the WNN tracker. We also implemented WiSARD with a multi-process similar to the other trackers.

Unlike other trackers explored in this work, the WNN object tracker was built from the ground up. The implementation was inspired and based on Nascimento *et al.* [97], which proposes a tracker based on short and medium-term memories to store the WiSARD Discriminators. Such implementation does not require any previous training and supports online training updates. In the first stage of the algorithm, the first frame's ground truth regions of interest (ROI) are used as the training base for the first discriminator. A user can select these regions or be considered the label provided together with the dataset. For each subsequent frame, a search is performed around the last object position by using a slide window technique. All windows are sent to all Discriminators stored in a queue to classify the corresponding object. The position with the highest response is presumed to be the current object location. If the highest response is equal to or greater than a threshold, it is assumed that the object features changed, and a new discriminator is trained and sorted into the queue. The queue is constantly re-ordered, with the last discriminator that gave the highest response being the first. If a new discriminator was created for that frame, it is the one on top of the queue. When the queue is full, the last discriminator is discarded.

	YOLOv3	**YOLO-Lite**	**GOTURN**	**KCF**	**MOSSE**	**WiSARD**
GPU	✓	-	✓	-	-	-
CPU	-	✓	✓	✓	✓	✓
CSD	-	✓	✓	✓	✓	✓

Table 4.1: Object Tracker implementation availability per device tested.

4.4 Experiments & Results

This section presents experimental results to describe the potential benefits of using computational storage devices for object tracking applications. The following subsections carry out a thorough analysis of the models and their power performance.

The six object trackers presented in Section 4.3 were tested on the available devices. Figure 4.4 shows in which device each tracker was tested on. To recap, YOLOv3 is the heavy/high accuracy object-detection algorithm that works well for GPUs. Therefore we will only use it on the correspondent device. YOLO-Lite will be the iteration of YOLO to be executed in CPUs and CSDs. We will also study their accuracies separately in the subsection 4.4.3. GOTURN is fully supported on all available devices. WNN and CFs are lightweighted algorithms and do not require powerful GPU devices to provide high FPS. Moreover, their operations are not best suited for GPU architecture to justify the data transfer. Note that the trackers that required previous training information and did not support online learning were deployed with pre-trained models provided by their respective authors.

Regarding the dataset, we select the OTB-100 [78] for all experiments. Previous work [108–110] have used the same dataset to evaluate their tracking performance. It is usually not deployed in offline training, as these networks usually require much more data. OTB-100 is made of 100 videos with different resolutions and lengths, implying a different total number of frames per video. Every frame comes with coordinates representing the ground-truth bounding box with the corresponding position of the object. The videos also come with a tagged series of 11 distinct categories describing relevant characteristics from its content. For example, some of the tags we can find are illumination variation, partial occlusion, and motion blur. Although it is interesting to have knowledge about such categories to infer conclusions, they are not used in practice by any models.

We implemented a greedy load balancing algorithm to balance the workload and execution on multiple CSDs and have them busy together for most of the time. The balancing algorithm is statically performed before we start the measurements and rely on the video duration to try and keep CSDs processing the same amount of minutes. Note that we base our idea on the fact that extensive videos will take longer to perform object tracking as they contain more frames. However, when

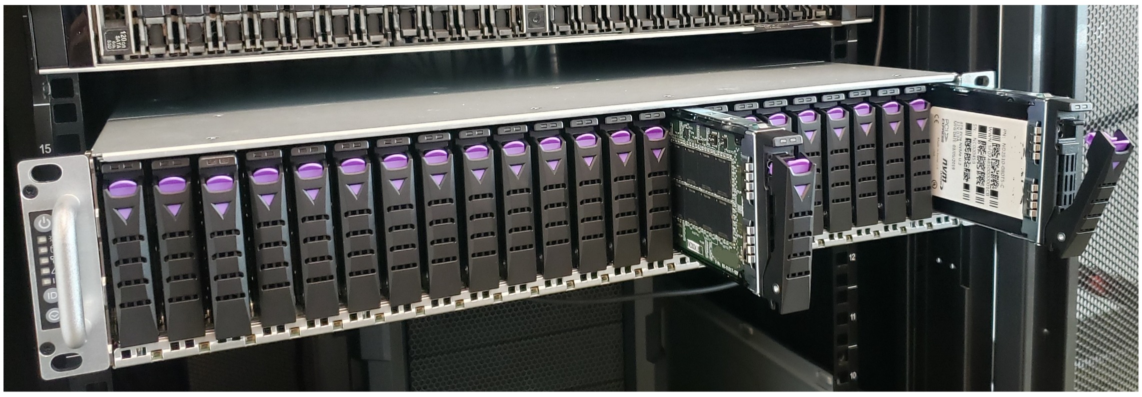


Figure 4.2: Experimental server blade with high capacity in-storage processing Newport CSDs. (Image from [1])

dealing with other datasets with more discrepancies between videos, it is important to consider different qualities and frame sizes as that may be the main characteristic to balance.

All experiments were carried over the server shown in Figure 4.4. The machine has a dual Intel Xeon Silver 4108 with 8 cores at 1.8GHz, and 64GB of DRAM. The server supports up to 24 NVMe U.2 SSD bays enabling a maximum of 23 Newport drives. The Newports used in the experiment use the architecture described in the Background Section 2.1.3. Although theoretically, the server could support 23 drives, each with 32TB and 4 ARM cores, totaling a whopping 96 extra cores and 768TB of storage, due to a limitation in supply, we were only able to test on 12 simultaneous drives. The power consumption of the platform is measured with a Wattman HPM-100A power meter [1] with the accuracy of 0.4% and a logging interval of 1 second.

It is important to mention that since the GPU is too big to fit into the server, we required a different setup when executing GOTURN and YOLOv3 with GPU enabled. The GPU machine had an RTX 2080 Ti 11GB GDDR6, an Intel Xeon CPU E5-2620 v4 2.10GHz, and 32 GB of DRAM.

4.4.1 Throughput Analysis

Table 4.4 shows different performance metrics for the different tracker application executing on different devices. We will focus on the two FPS performance metrics presented in the table to evaluate throughput. *Overall Throughput* is taking into account the total elapsed time to complete all object tracking available in the dataset. However, most of the algorithms deploy a multi-process approach to different videos. Our FPS for overall throughput is the amount of frames processed per second for

[1] http://shop2.adpower21com.cafe24.com

Computing Resource	**Tracker Model**	**Overall Throughput (FPS)**	**Avr. Per-Video Troughput (FPS)**	**Energy Consumption (J/Frame)**
GPU	YOLOv3	34.79	34.66	6.03
	GOTURN	194.83	199.26	2.13×10^{-1}
CPU	KCF	1145.85	220.69	4.38×10^{-2}
	MOSSE	3310.22	493.70	1.48×10^{-2}
	WiSARD	1568.00	72.59	3.95×10^{-2}
	YOLO-Lite	2.72	2.76	15.26
	GOTURN	21.24	2.03	2.42
CSD	KCF	186.78	32.09	1.51×10^{-2}
	MOSSE	1295.30	143.87	5.54×10^{-3}
	WiSARD	199.28	11.96	1.35×10^{-2}
	YOLO-Lite	2.36	0.23	2.00
	GOTURN	1.83	0.12	6.26×10^{-1}

Table 4.2: Several performance metrics for each model executed on their appropriate available devices.

several different videos in parallel. Usually, FPS is used concerning a single video, so we also introduce the *Average Per-Video Throughput*. The per-video throughput metric is related to the time to process the frames for the same single video, not affected by any multi-process execution. The FPS value of each video is averaged over all the datasets and displayed in its corresponding row.

Most tracking applications decrease computational power consumption as they go down from GPU to CPU to CSD. This result is expected to happen as cache and bus bandwidth decreases and processing units weaken. However, parallelization of some workloads helps mitigate some of these effects mitigating of their effects. Some of which maintained a real-time performance of 30 FPS or more.

The MOSSE filter had the fastest time and the highest overall per-video throughput. However, as we will see, it did not provide enough robustness for the entire dataset, which we will get into when discussing accuracy in Section 4.4.3. WiSARD provided a good trade-off between its execution time and accuracy, but their per-video FPS fell below the 30 FPS threshold for CSD, while both the filters kept good performance.

Our DNN models took a big hit, decreasing their throughput but orders of magnitude once we moved from GPUs to CPUs or CSDs. GOTURN had the lowest FPS compared to other trackers running on CPU or CSD. As we explained in previous sections, the network seems highly optimized for GPU execution, which explains why their throughput is at its highest in the device.

In both of the cases of GOTURN and WiSARD, it is likely that a parallel implementation through multi-thread instead of multi-process could greatly benefit the network by devoting its resource to a single video we could achieve a real-time per-video throughput, especially for WiSARD. Although, it would unlikely increase our overall throughput as we would be shifting the focus of working on multiple videos in parallel from working on a video in parallel.

For both YOLO networks, the FPS was very low on CPU and CSD, even though it had multi-thread execution enabled. YOLO variations are expected to perform worse than the other applications as it is foremost an object-detection algorithm. Therefore, it does not depend on the previous frames since it treats each frame as a still image. Paralyzing the workload by having each CSD working on the same video but in different frames might provide itself helpful in increasing fps performance.

4.4.2 Energy Consumption Analysis

Our methodology for measuring the energy consumption was as follows: First, we collect the power consumption output when the machine is powered on in idle mode, with only OS functionalities being executed. This initial measurement is our baseline. Afterward, we execute one of the tracker applications in a device and measure its active power in Watts during the execution. We derive total energy consumption in Joules per Frame (J/Frame) by calculating the differential power consumption between the tracker power consumption and the baseline taking into account the total execution time of the application. Note that a better energy efficiency implies lower energy consumption.

The last column of Table 4.4 presents our measurements for every device and tracker. Most of the trackers had a smaller energy consumption when executing in-storage computation, even though several CSDs work in parallel. Moreover, MOSSE had an order of magnitude decrease in J/Frame while keeping a good throughput. WiSARD had similar measurements to the CFs, which were low consumption but still achieved reasonable FPS. More computational heavy trackers such as DNNs required much more resources and time to complete their tasks, resulting in around an order of magnitude higher energy consumption for the CPU as well.

GOTURN is the only tracker with smaller energy consumption for GPU execution. This is an interesting finding as it shows that the algorithm is not only better optimized for GPUs, but it uses it so much more efficiently that it saves energy by taking less time to execute while better utilizing the resources.

The biggest and more complex network in YOLO, we find the worst J/frame out of the applications. Moreover, we still can see that YOLOv3 has a better performance in GPU regarding memory and throughput, for the same reason GOTURN does. Still, it shows an improvement in energy when moved into a multi-CSD environment.

	Average IoU	**Standard Deviation**
MOSSE	0.1985	0.2555
KCF	0.1832	0.2183
YOLO-Lite	0.1673	0.1978
YOLOv3	0.3588	0.2980
GOTURN	0.4140	0.2414
WiSARD	0.3543	0.2439

Table 4.3: Accuracy-related Metrics for the Object Tracking algorithms.

4.4.3 Accuracy Discussion

In this section, we will discuss the accuracy of the models while comparing their performance and energy consumption. Unlike standard classification or regression problems, F-measurements and pure accuracy are not directly correlated to the bounding box area. Therefore, we use Intersect over Union (IoU) as it is one of the most popular metrics for evaluating object tracking environments [111]. To calculate the IoU, we must divide the intersection and the union of the ground truth bounding box and the predicted bounding box areas. This function outputs a value between one and zero, where one represents a 100% match. For reference, values above 0.5 are considered a great match.

Table 4.4.3 displays the average IoU and its respective standard deviation with respect to all measured videos in OTB-100. A individual table with separate IoU measurements for each video can be seen in Appendix B at Table B.1. When evaluating the average IoU we can see that more complex and computationally heavy trackers provide the best overall accuracy. GOTURN achieves the highest IoU with 0.41. Even though YOLO is essentially an object-detection algorithm, it also provides good overall accuracy. Interestingly, when moving into the YOLO-Lite implementation, the IoU greatly decreases, being the worst of all trackers. An IoU decrease was expected as the network does not contain several layers from the original YOLO, but it is still interesting to see that the impact was huge in this case.

MOSSE and KCF also have low overall IoU even though they are pretty fast, per our throughput results previously discussed. This result shows that the tested CFs do not provide enough robustness to infer over the whole dataset. Still, there are several instances of videos in which MOSSE and KCF achieve a very good IoU, as evidenced in Table B.1 inside the Appendix. For example, in the video Coupon, MOSSE achieves an IoU of 0.8841. Moreover, MOSSE specifically has many zeroes due to objects being considered out-of-frame, which dramatically decreases the average IoU value, evidenced by a higher standard deviation.

When we look over, most of the standard deviation values a relatively large. One of the main reasons this is the case is due to OTB-100 being comprised of a diverse

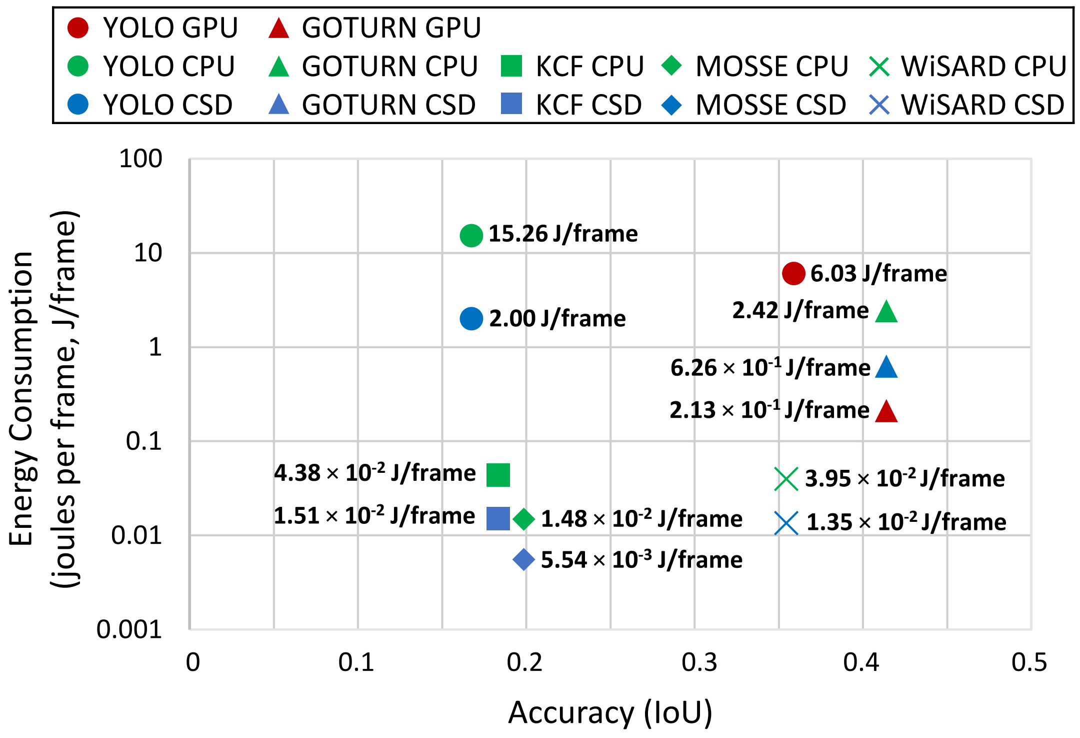


Figure 4.3: Energy consumption (Joule/frame) versus accuracy metric (IoU) of the tested models in different devices.

pool of videos with different environments, which provoke different frame noises and target sizes. Therefore, unless the tracker is robust enough to support all available varieties, it will achieve poor accuracy in some sets of videos.

WiSARD performs well, being a close third in IoU to GOTURN. Because it can execute on CPUs with a good FPS and throughput, it does seem a good candidate for in-storage object-tracking.

To further evaluate and provide a comparison between the available trade-offs for energy consumption and accuracy, we introduce Figure 4.3. The figure displays a plot for energy consumption per average IoU for every device and tracker combination tested. The results are presented in log scale on the y-axis since some energy-consumption values are too close to be well displayed. The devices are separated by color, and the tracker is differentiated by shape. The further to the right and lowest consumption we have, the better the object tracker is.

GOTURN's highest overall average IoU and a more or less in line standard deviation coupled with its energy efficiency and throughput on the RTX 2080 Ti makes it the clear best target for a GPU device. The WiSARD-based tracker had a great IoU closer to the desired 0.5, coupled with being orders of magnitude faster than DNNs on the host and CSD while providing excellent energy efficiency, especially on the CSD devices making a great target for general-purpose CPUs. The only tracker which provided better energy efficiency was the MOSSE Filter, but its average accuracy was too low. Still, it could be the case that the filter can perform well for some applications, as we pointed out previously.

It is also important to remember that by using computational storage, we are not only decreasing data movement but also enabling the host CPU or even a GPU to compute other tasks, which is a view not present in the data shown.

4.5 Concluding Remarks

The work in this chapter evaluates several object tracking algorithms concerning their energy efficiency and in-storage processing capabilities by idolizing the Newport computational storage environment. We introduce modifications to the tracker and a multi-process approach based on a static greedy algorithm to utilize the available resources per applied device fully. Our study case considers several metrics and how power consumption and accuracy may be key trade-offs to analyze when selecting the location an application will be executing on.

Our evaluation shows GOTURN could be an interesting case for GPU devices, while WiSARD might be best for executing in the edge or multi-CSD environments. Moreover, CFs may also be applicable depending on video noise and properties. More importantly, it shows an example of what a future scenario where CSDs are widespread may look like as far as object tracking goes.

Chapter 5

Admission Control on a NextG Dynamic Environment

5.1 Motivation

The increase of IoT devices through different levels of society pushes Next-Generation Networks (NextG) to provide a vast range of new services unavailable in the current generation [112]. Machine Learning (ML) is expected to be responsible for managing vital virtual resources and automating massive data contents as well as communication-intensive networks [84]. Moreover, 6G and further will deliver a fully connected and integrated intelligent infrastructure following the same trend of ML [19]

Network Function Virtualization (NFV) [113], and Software Defined Networks (SDN) [114] will provide the technological foundation for the introduction of new services. One of such services is called network slicing, where the physical infrastructure splits into virtual instances with different resource characteristics and functional attributes for specific services. This paradigm enables the infrastructure provider (InP) to facilitate its network infrastructure-as-a-service [115] and target specific services with unique requirements. Unfortunately, the provider has a limited pool of resources [116] and therefore must define an automated slice admission control policy to decide which resource requests to accept.

Ideally, the InP would like a decision algorithm that finds the optimal policy that maximizes its long-term profit. Given the uncertainty and variations of future requests and resource availability, reinforcement learning (RL) algorithms have been proposed towards this goal [115, 117, 118]. However, many of these works provide a limited environment with static request attributes. Their goal is to maximize revenue, but the revenue per request is fixed, lacking a pricing model.

RL is a machine learning algorithm that acts through an agent by observing an environment. The agent learns as it explores the environment space, receiving rewards (or penalties) for every action taken. Its final goal is to maximize the total reward received. Due to their recent success, such algorithms have attracted attention from industry, and academia [119]. We have provided background on characteristics and algorithms in RL in Section 2.2.4.

In this work, we address environment and pricing problems by developing a dynamic environment that allows for concurrent request arrivals. Requests from the same service class are generated to consider different resource needs, a waiting list queue, and a dynamic pricing model inspired by ride-hailing platforms, which considers supply and demand factors of the environment. We speculate that services in NextG will feature many such attributes for competitive resource access. We deploy standard and dueling architectures for Deep Q-Networks [120] with the goal of maximizing the InP's profit.

The work contributions for this body of work can be summarized as the following:

- Dynamic admission control with varying request attributes (e.g., urgency, duration, amount of resources) for conventional service classes;
- Dynamic pricing model and decision ordering for demand shaping and profit maximization ;
- Novel situational modeling of joint multi-slice request and profit-based reward using a vision-inspired approach;
- Modified Deep Q-Learning experience replay structure to sample based on actions;
- Analysis of reinforcement learning algorithms with varying request arrival rates and duration.

This chapter of the thesis is organized as follows: Section 5.2 briefly presents related works that used RL in the admission control problem as well as dynamic pricing models in earlier literature. Section 5.3 provides an overview of the overall system, defines our Markov-Decision Process, the specifics of our implementation of DQN structures, and our dynamic pricing model. Section 5.4 discusses the experiments and results of the RL algorithms applied to the network slicing problem. Lastly, we conclude this chapter with final remarks in Section 5.5. The main results of this chapter where incorporated in Ferreira *et al.* [121].

5.2 Related Work

In this section, we present some related and prior works that pertain to the use of Deep Neural Networks (DNNs) in an admission control environment and dynamic pricing related to NextG networks.

As we will highlight in the following paragraphs, prior work do not consider (*i*) a *dynamic environment* where requests attributes from a same service may vary and are dynamically generated, (*ii*) a *dynamic pricing* for demand control in the admission control scenario, and (*iii*) a request queue structure with a timeout option. All of which we define and model in our proposed system.

The admission control problem can be summarized as a decision process to accept or reject resource requests in a way that we can optimize something of interest. In general, previous works deal with revenue maximization. After acceptance, the provider should allow for some Network Slicing procedure to execute the accepted request jobs.

Van *et al.* [115] proposes that resource requests should come with well-defined consumption requirements similar to current data center resource allocation. Van's work classifies them as storage, bandwidth, and computation requirements. Each resource is derived by its request class (URLLC, eMBB, or mMTC). It is essential to clarify that there is no difference between requests from the same class. Van presents a Dueling DQN algorithm as a solution for the admission control of the requests. The primary goal of the neural network is to maximize revenue by accepting better requests.

Esmat *et al.* [117] expands Van's work by broadening the action space to include the possibility of accepting the request by leasing subscribers' resources. Esmat introduces Dueling DQ-EFNS, which receives a fixed reward based on the chosen action and the accepted request class. If the requests are accepted to use the leased subscriber resources, there is a penalty in the network reward.

These works consider that each request would always require the same amount of resources based on their class. Moreover, their model aims to maximize INP revenue, yet the reward value is fixed purely based on the accepted request class. In our work, we provide a more complex scenario with dynamism for both the request attributes as well as the reward. We have also remodeled the problem to support more actions and give the algorithm a broader view of the environment.

Thantharate *et al.* [122] introduces DeepSlice, a CNN-based deep learning algorithm that predicts between the regular service classes and allocates it into the proper slice. Unlike the previous two works, DeepSlice does not apply RL but a classification algorithm that chooses which network slice the application will execute. Besides the essential service request class slices, it provides a Master Slice for managing new slices in case of reliability issues, therefore maintaining Quality-of-Service (QoS).

Chounos *et al.* [123] provides a type of dynamic pricing through a Stackelberg game into the 5G scenario. In their scenario, Mobile Network Operators (MNOs)

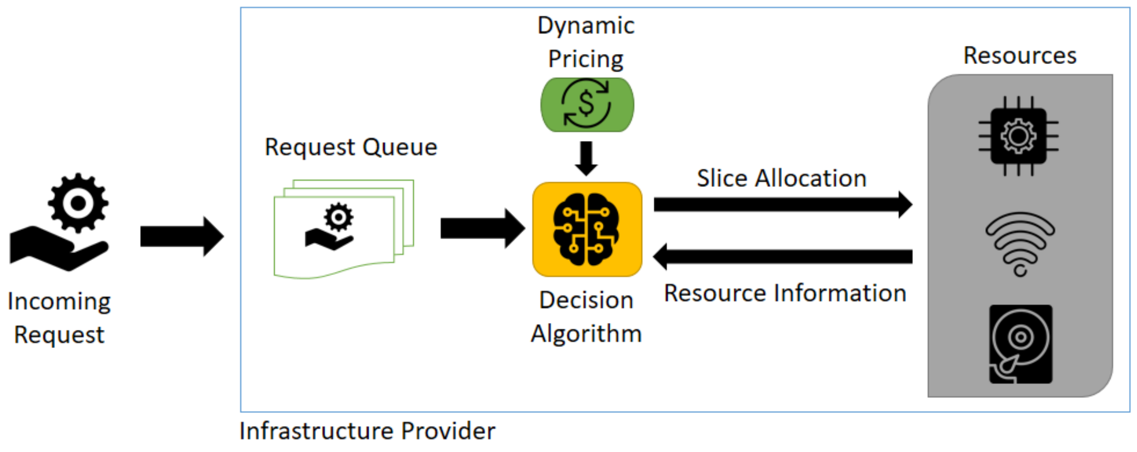


Figure 5.1: Admission control system components and flow overview.

may need to lease additional resources to continue providing user support if the demand is high. A Service Level Agreement (SLA) between the MNO and INP is agreed upon. Next, the MNO charges an optimal pricing value based on the 5G Policy Charging Function. Umut *et al.* [124] introduces a techno-economic market model for radio access networks to allow negotiation between MNOs and INPs, which dynamically prices resources based on multiple factors such as supply and demand and expenditure budget. These dynamic pricing schemes and complex market modeling differ from the admission control scenario and RL algorithms we are applying in this work as they look into other 5G policy scenarios.

5.3 Reinforcement Learning Framework for Slice Admission Control

This section will give the reader an overview of our proposed work by describing the environment. Our System Model section gives a thorough description of our system's several components that are connected with each other. The other three sections give an insight into the components and architecture of our decision algorithm.

5.3.1 System Model

Our multi-attribute slice admission control system is exemplified through the block diagram in Figure 5.1. The system can be viewed as a generic admission control environment containing a *infrastructure provider* (InP) and its three main resources, *service requests*, a *request queue*, *dynamic pricing model*, and a *decision algorithm*. To properly model our environment and RL decision algorithm, we will define an incremental passage of time referred to as the time horizon. Our time horizon is defined as a set of time steps t, where $t \in \{0, 1, ...\}$.

The InP is the central piece of the system. It holds most of the components and is responsible for dictating the price and deciding which request to accept. After request acceptance, it must provide and allocate the resources by performing network slicing. The InP's resources are divided into storage, computation, and bandwidth, similar to previous work in the area [115, 117].

A service request $R_i \in \mathcal{I}$, where $\mathcal{I}$ is a set of all possible requests, arrives with a set of attributes that define its structure. Many of these attributes are used by the decision algorithm as features and the dynamic pricing model to correctly calculate profit.

Request-id i and the Request service class, i.e., Ultra-Reliability Low-Latency Communications (URLLC), enhanced-Mobile BroadBand (eMBB), or massive Machine Type Communications (mMTC) are some of the control attributes a request possesses. The service class attribute is randomly generated uniformly at the point of request creation. Its primary purpose is to be used as a template for defining R_i resource requirements.

It is well known that each service class will demand different amounts from each resource [117]. Therefore, we used a base value multiplied by a randomly chosen variable to generate each of the available three resource requirements. This multiplying factor is a value between 0.8 and 1.2 and greatly diversifies the available amount of request attributes, even though they are from the same service class. This approach allows us to model different application requests within the same class. It is important to highlight that this random factor needs to be bounded by a small margin. Otherwise, we might change the blueprint of the service class and have, for example, a URLLC request with similar resource needs as an eMBB or mMTC request, which beats the whole purpose of having service class templates. The resource requirement from a request will be referred to as θ, where $\theta \in computation_i, bandwidth_i, storage_i$ The provider's total amount of resources and service class base resource values are derived from Van *et al.* [115].

Other attributes present in R_i are slice duration d_i representing how much time the resources need to be allocated. We applied an exponential distribution with parameter μ to randomly generate d_i for each individual request. This value is varied during experiments to evaluate different allocation times. The attribute patience p_i symbolizes the number of time steps R_i is willing to be inside the request queue before giving up and leaving the queue. Currently, p_i is set o half of d_i and starts to be tracked when R_i is inserted into the queue. In case it is accepted or rejected before we reach the patience limit, nothing happens. Lastly, a request contains an urgency u_i attribute, representing how urgent R_i might need the requested resources.

u_i is randomly selected between 1,2, and 3, implying three levels of urgency. Many of these attributes play a significant role in our dynamic pricing strategy and are chosen to draw a parallel with ride-hailing platforms, which we will better discuss in Section 5.3.4.

The decision algorithm decides if the current R_i is accepted or rejected. It considers several request attributes, as well as information on available InP resources and price. Even though our decision algorithm is based on RL DNNs, our admission control system supports any algorithm similarly performing a decision process. In our described system, we are considering one-side markets, which means only one player is bidding a price. As seen in Figure 5.1, the dynamic pricing scheme is a part of the infrastructure provider; therefore, the bidder is the InP. In practice, this implies that request owners are willing to pay whatever the InP set price is, and no negotiation is required.

The request queue is a simple list that stores incoming requests in a FIFO manner. At any given time t several scenarios might happen concurrently and may change the contents of the queue. Request arrivals follow a Poisson distribution with a mean λ and will append a new request to the queue. The value of λ will be varied during the experiments to evaluate different arrival rates. Residing requests have an internal counter that logs the number of time steps it has been sitting inside the queue. If the counter reaches the request patience limit, it will cause a timeout that removes it from the queue. The only other way to remove requests is if the decision algorithm accepts or rejects them. In case of acceptance of a R_j, the InP calculates its profit from the transaction, which will later help with the DNN training, and it updates the list of executing jobs by time-stamping the time of acceptance. After $t + d_j$ time steps, the running job corresponding to R_j finishes executing and is moved into a completion list while deallocating the resources it originally requested.

5.3.2 MDP Formulation

As explained in Section 2.2.4, when solving problems in the RL sphere, we formally model it through an MDP [54]. To recap, our MDP is a tuple of $M =< T, S, A, Pr((s'|s), a), r(s, a) >$, where $T = \{1, 2, ...\}$, note that the previously defined $t \in T$. S is the set of all available states and $s \in S$, A is the set of all available actions and $a \in A$, $r(s, a)$ is our reward and $Pr((s'|s), a)$ is a probability that taking a we move from s to s' in the next time step. Our algorithm aims to find the best set of actions for the available states at any available t that maximizes our cumulative reward. This is known as optimal policy ρ^*

Let us now describe the main elements of M in our system:

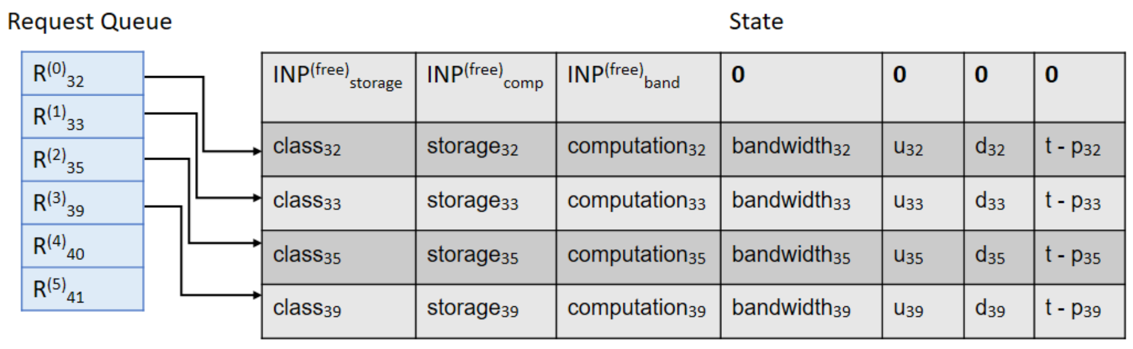


Figure 5.2: Our matrix representation of the state and how the features are correlated with the current queue requests a a given time step t.

State: The state is the input of our RL algorithm. We must carefully craft and select the features that will provide the model with enough view of the environment to make proper decisions. Still, it is crucial to consider that in some cases, there will be hidden values in the environment that can not be accessed by the agent and should not be included in the state.

Our defined state s consists of a matrix representing a gray-scale image $(width{,}height{,}1)$ containing data from requests currently residing in the request queue as well as some internal InP information. The first image row has the amount of free resources for each θ the InP has at the current time step t. Every subsequent row consists of a different R_i attributes such as: request class, each one of their three requested resources, u_i, d_i, and time left inside the queue $t - p_i$. Figure 5.2 exemplifies a possible state at t. $R^{(}_{i}k)$ represent the requests, where k is the current request position inside the queue.

The requests inside s are taken from the top of the request queue in a first-come-first-served manner. We shall have at most $height - 1$ requests for any given s, even if the queue is holding more requests. In case there are not enough requests inside the queue to fill the whole image at the current t, the image will be zero-padded. All θ values are normalized to be between the range of 0 and 255, emulating a gray-scale image.

Action: The available actions an agent can take in this environment are defined by the state height, as in $A = \{0, 1, 2, ..., height - 1\}$. When $a = 0$, the decision is to reject the current top request from the queue, which will remove it from the queue structure. Any other action between 1 and $height - 1$ represents a decision to accept request $R^{(}_{i}k)$, where k is correlated to the request pictured in the ath row of s. This approach allows the model to select requests based on how they are displayed in the input. In case it chooses a $R^{(}_{i}k)$ that is zero-padded into s, due to insufficient requests available at the queue, the agent is interpreted as choosing an action of waiting, neither accepting nor rejecting any request. By following this approach, the algorithm can effectively choose which request from the queue within the current state it wants to accept at t, allowing for a more flexible decision process.

It is important to mention that if the model chooses an action a that results in an overbooking of resource θ, we will force a reject action $a = 0$ without consulting the decision algorithm. Overbooking resources in the described environment means that the current amount of θ required from R_k is higher than the currently available amount of free θ from the InP. **Reward**: Our reward function is defined as follows:

$$r(a, s) = \begin{cases} 0 & \text{if } a = 0 \\ \pi_i & \text{otherwise} \end{cases} \tag{5.1}$$

In case a request is rejected, the reward is zero; otherwise, it receives a profit π_i from accepting request i following our dynamic pricing model explained in Section 5.3.4. Note that the action of waiting, that is, selecting a zero-padded request, would yield a reward of zero as well. The waiting reward is derived implicitly from our profit calculation, as it would be the same as accepting a request without any requirements.

5.3.3 CNN-based Slice Admission Control

Given the previous success of DQNs in RL problems [62] and also in similar, yet more constrained, scenarios than the one presented [115, 117, 118], we decided to apply both Dueling DQN and the standard DQN to our environment. As detailed in Section 2.2.4, Deep Q-Learning uses Neural Networks to provide a good approximation of Q-values by training a network that learns the optimal policy as it explores the environment. At every time step, unless the request queue is empty, the DQN model receives the current state and produces a decision of which request it will accept. Furthermore, since we model our state as a matrix, we can deploy both our DQNs using CNN layers.

We will be using the ϵ-greedy strategy to explore the environment with an initial value of 0.9 and a final value of 0.1. The decaying factor is calculated by $(initial - final)/explore\_rate$, where the explore rate is a hyperparameter.

One of the important structures present in DQNs discussed in Section 2.2.4 is the experience replay. For this work, we have also implemented another famous iteration of the replay buffer, called *prioritized experience replay* [125]. Unlike the standard buffer, it will attach new weight values to the saved tuple, which are then used during the sampling procedure to prioritize some experience tuples over the others. The idea is to select the tuples the model is performing worse to force it to better learn the overall latent characteristics of the data.

Besides these two replay buffers, we developed a particular structure we named *balanced replay buffer*. Similar to classification problems, when class distribution is unbalanced, the trained model is more likely to develop some type of bias. Therefore if our admission control problem's actions are too similar, our experience replay buffer might be filled with similar entries, returning mostly samples from a specific action to train the network with. To minimize such occurrence, our *balanced replay buffer* has one separate buffer for each available action. When performing sampling, each of them will be sampled uniformly, and the return of the balanced replay will have a similar number of samples from each available action.

Lastly, to improve our network convergence, we applied a reward scaling technique using a factor of 10^{-4}. This procedure was previously used with success for a similar purpose in other works [126, 127].

5.3.4 Dynamic Pricing

Even though previous works on admission control have considered maximizing revenue as its main optimization goal, they lack a proper pricing modeling scheme that reflects a realistic environment. We chose to maximize *profit* (π) instead of revenue as we can model it as a function of revenue (*Rev*) minus the cost (C) [128]. By allowing the equation to include cost, the INP can model both fixed and variable cost scenarios, including the original revenue-only cases ($C(R_i) = 0$). Equation 5.2 below displays our a profit π_i for accepting request i:

$$\pi_i = Rev(R_i, t) - C(R_i) \tag{5.2}$$

Our revenue function was partially inspired by ride-sharing platforms such as Uber. These platforms deploy proprietary algorithms for pricing calculations which take into account several different factors [1]. For example, we are aware that a fair base price and travel distance are required and are probably the basis of the calculation. In our case, we will have a similar base fair, but instead of distance, we will use the amount of resource requested by R_i. It is also known that several factors outside ride-hailing platforms control impact pricing, such as supply, demand, and how much a user is willing to pay [129]. Equation 5.3 displays our Revenue calculation as a function of attributes from R_i and the current time step t presented below. Note that we R_i as a way to summarize a given request i attributes previously defined in Section 5.3.1.

[1]`https://help.uber.com/riders/article/how-are-fares-calculated/?nodeId=d2d43bbc-f4bb-4882-b8bb-4bd8acf03a9d`

$$Rev(R_i, t) = u_i * d_i * \sum_{\theta \in \{c_i, b_i, s_i\}} P(\theta, n_i^{(\theta)}) * SD(\theta, t) \tag{5.3}$$

$Rev(R_i, t)$ is comprised of several multiplying factors and a price fair ($P(\theta, t)$), making a parallel on how in demand-shaping several factors impact on the final overall pricing. We defined our factor as the current request urgency u_i, how long will the resources be allocated for d_i and a *supply and demand* (SD) function $SD(\theta, t)$.

Remember that our SD factor and price fair are a summation of each available resource request by R_i. These are a set of computation (c), bandwidth (b) and storage (s). By defining a different price, supply, and demand for each resource individually, we can better model the different service aspects of all available service classes and any other future ones. We must also consider it as a function of t to properly model the current supply and demand, as jobs may release or consume new resources as old ones finish and new ones are accepted.

Our SD function is defined by the Equation 5.4 below:

$$0.8 \leq SD(\theta, t) = (\theta_{req}^{(t)})^{\alpha} * (\theta_{free}^{(t)})^{-\beta} \leq 5.0 \tag{5.4}$$

$SD(\theta, t)$ is comprised of a value representing the InP current supply $\theta_{free}^{(t)}$ and the current demand $\theta_{req}^{(t)}$ for a resource θ, where $\theta = c, b, s$.

$\theta_{req}^{(t)}$ is calculated by computing the average of all θ from requests sitting in the queue at the current time step t. This strategy allows us to have the average demand at t for request θ. $\theta_{free}^{(t)}$ represents the current amount of free resources the InP has available given the previously accepted request currently executing at the same time step t. Knowing how much of the resource θ we have left, we can use it as our supply. Both α and β are regulator parameters to control how much should demand or supply influence the SD function. For our experiments, we set a value of $\alpha = 1.2$ and $\beta = 1.0$, giving a slight edge over demand to decrease the amount of generated discounts ($SD(\theta, t) < 1.0$) in high supply and low demand scenarios.

After some initial testing, we noticed the SD factor would increase and decrease rapidly in a minimal amount of timesteps because of cases where INP resources were mostly depleted or as the queue changed in size. Therefore, it would be unrealistic for price change without any control by spiking and decreasing orders of magnitude with each time step. To provide limitations to the equation, we bound it to the values of 0.8 and 5.0. It should be pointed out that these values can be freely customized to adequate the user case.

$P(\theta, n)$ represents our resource price fair which encompasses a straightforward multiplication between a fixed base fair for resource θ and the amount of the requested resource $n_i^{(\theta)}$, similar to Kim *et al.* [130]. Note that neither n_i nor θ vary with time, and the request price of resource θ is independent of the request service class. Equation 5.5 below displays the calculation.

$$P(\theta, n) = p_\theta * n_i^{(\theta)} \tag{5.5}$$

Now that we have dealt with revenue, we can move into Cost presented in Equation 5.6. The $C(R_i)$ represents the amount of cost it takes to execute R_i, given its attributes. It follows a similar structure to our revenue calculation, but it does not consider urgency and SD factors. Variable δ works as a regulator to control the base pricing cost of maintaining the requested resource θ. In our experiments we chose a value of $\delta = 0.1$.

$$C(R_i) = d_i * \delta * \sum_{\theta \in \{c_i, b_i, s_i\}} P(\theta, n_i^{(\theta)}) \tag{5.6}$$

It is important to mention that even though we are defining a variable cost by using these variables and calculating demand by averaging queue values, these equations should allow minor modifications to support several other calculations. For example, cost values could be fixed or found through a predictive algorithm. Moreover, demand might be generated through a more complex forecasting algorithm based on past events to predict the future. As long as it makes sense for the profit user case calculation, we can deploy it as a reward in our RL admission control algorithm.

5.4 Experiments & Results

To evaluate our dynamic environment, we try to emulate different situations while looking at different metrics to compare different models and make a decision. We will be evaluating the system on a fixed 1K time steps gap. To enable different environmental situations, we varied our slice arrival rate λ and the exponential mean μ related to the slice allocation duration. We selected λ values of 1, 5, and 10 to study cases where the request queue may be empty, partially empty/full, and mostly full. μ values are either 5 or 100 to explore the impact of different allocation durations. Every request attribute upon arrival is generated according to Section 5.3.1. We will refer to possible combinations between λ and μ as datasets. All experiments were carried out on an Intel Core i7-6700 3.40 GHz server, 32GB of DRAM, and an Nvidia GeForce GTX960-4G on a Ubuntu 16.04.

We have implemented both Dueling DQN and standard DQN using our vision-inspired approach, which uses a CNN architecture followed by FC layers with RELU activation. We refer to them in the experiments as *cnn_ ddqn* and *cnn_ dqn*, respec-

Dataset	**cnn_ddqn**	**cnn_dqn**	**ddqn**	**fcfs**
$\lambda = 1$ & $\mu = 100$	3.09	3.12	**2.70**	**2.70**
$\lambda = 1$ & $\mu = 5$	1.15	1.14	**1.02**	**1.02**
$\lambda = 10$ & $\mu = 100$	43.66	43.66	43.66	43.66
$\lambda = 10$ & $\mu = 5$	3.05	3.05	3.05	3.05
$\lambda = 5$ & $\mu = 100$	39.16	**38.17**	38.19	38.19
$\lambda = 5$ & $\mu = 5$	2.83	2.82	**2.78**	**2.78**

Table 5.1: Average Waiting times in time steps for the final training epoch of each model and dataset scenario.

tively. Moreover, we also adapted Van *et al.* [115] Dueling DQN proposal (*ddqn*) to see how it would fair a more dynamic environment setting and serve as a comparison for our implementation. The baseline approach is a first-come-first-served (*fcfs*) algorithm, which will always accept the first request at the top of the queue as long as it does not overbook the InP resources. All DNN models were implemented in Pytorch 1.9, and a hyperparameter exploration with Random Search was done through RayTune 1.8.

5.4.1 Model Comparison

Figure 5.3 shows the accumulated reward of the 1K timesteps after the final training epoch. These values are an average of 10 executions of the models, including retraining of the network for the same dataset. Annotated values on top of the bars were truncated to integers to improve Figure visualization.

It is important to remember that since our reward per time step is the profit, the cumulative reward is a synonym for cumulative profit and that the presented values were scaled down according to Section 5.3.3.

Looking at overarching model behavior on the different tested datasets, we can conclude that the DDQN model based on Van *et al.* [115] admission control fixed reward scenario is having many difficulties in finding a good enough ρ. In fact, in most instances, it achieves a slightly better result than our *fcfs* baseline approach, with the scenario of $\lambda = 10\&\mu = 100$ having the worst performance of all models. Our proposed approaches using a CNN architecture with a vision-based state outperform the baseline and Van's DDQN in all available situations. However, the margin is small for a few datasets where we have a higher arrival rate and smaller job duration ($\lambda = 10\&\mu = 5$ and $\lambda = 5\&\mu = 5$).

These results provide evidence that allowing the model to have a broader system view by including several requests into the state and expanded action set by enabling it to choose and pick which requests to accept can greatly benefit policy learning, which in term generates a higher accumulated profit. Moreover, by using a vision-inspired approach, we can benefit from state-of-the-art CNN layers, which greatly

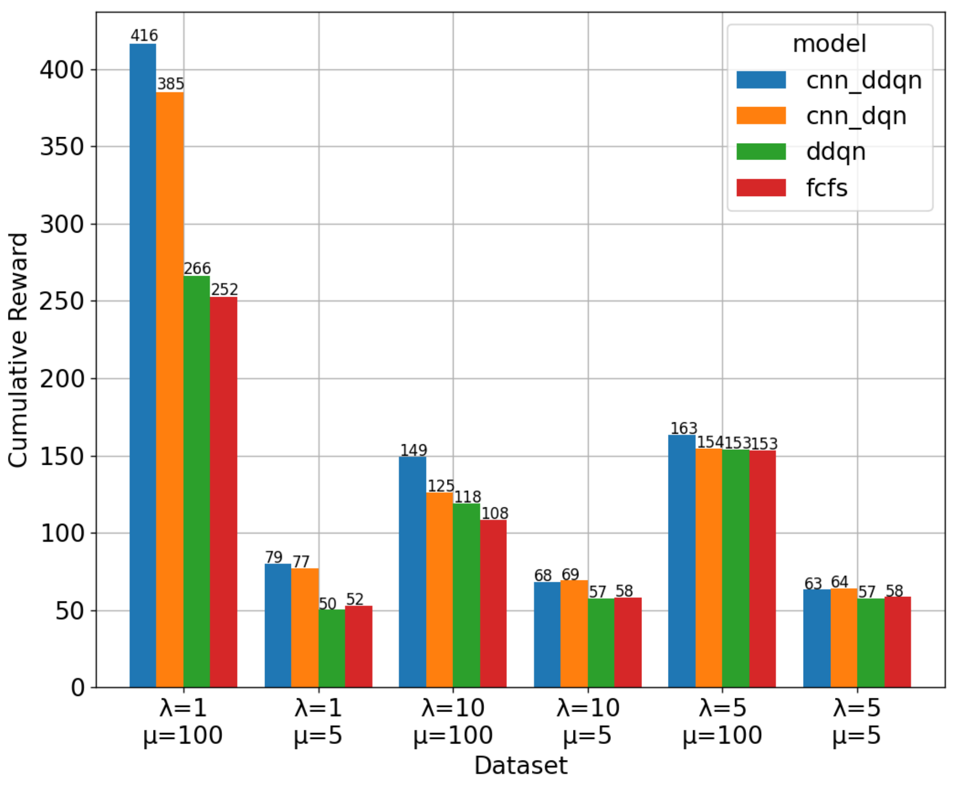


Figure 5.3: Amount of cumulative reward at the final training epoch for each model and each dataset scenario.

benefit feature learning and exploitation of the presented dynamic environment.

Another interesting point is that similar to previous work [57, 117], the usage of a dueling architecture through *cnn_ ddqn* also helped with network convergence. In most of the presented scenarios, it achieved better policies than using a standard *cnn_ dqn*.

Figure 5.4 shows a epoch vs reward plot for all the models. The *fcfs* approach is displayed as a continuous line since no training is involved. In most cases, the models converge pretty nicely just after a few epochs. The *ddqn* model based on Van's work does show somewhat of a convergence curve, but it mostly plateaus around the baseline. The CNN models converge pretty well, even finding a better policy than the baseline after the first epoch. There are a few instances where the model does not converge well. Previous research has noted that Deep Q-learning approaches may be challenging to converge in some scenarios [131]. Still, all models converge loss-wise, meaning the model is learning to generalize the data and optimize the objective.

Table 5.1 shows the average waiting time (in time steps) of the final training epoch on the 1K testing steps for our available datasets. Essentially, it shows how long a request stays in the queue until it is accepted, rejected, or times out. Note that higher arrival rates (λ) and longer resource allocation (μ) corroborate higher

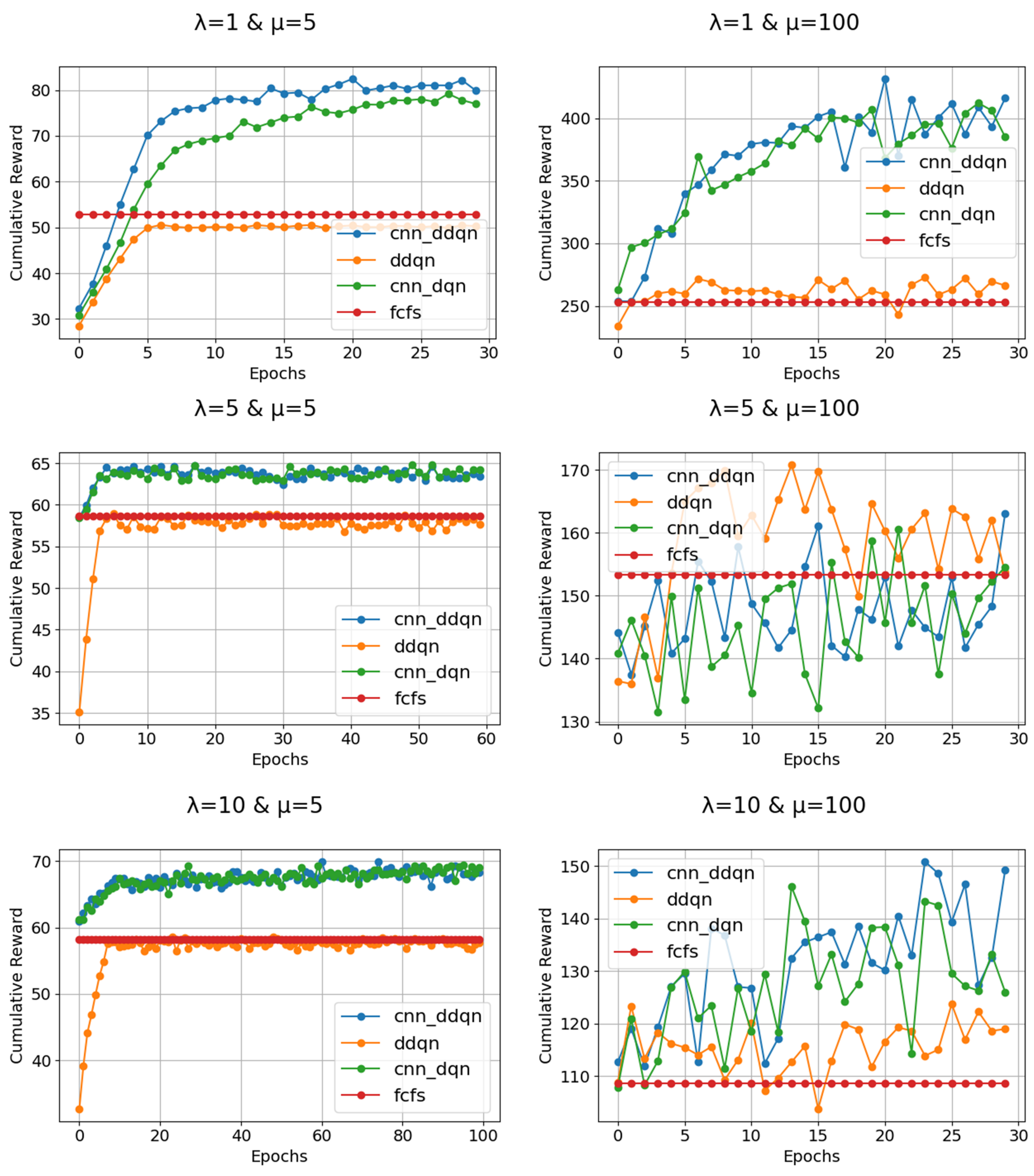


Figure 5.4: Model convergence as training progresses throughout our environment for the different models and different datasets

waiting times. These higher values happen because the InP resources are occupied for longer due to the higher μ, leading to resource starvation. Therefore, the InP is unable to accept any more requests, or it would overbook its resources, leading to requests waiting longer in the queue and more frequent time outs which increases the average waiting times.

Interestingly, the baseline and *ddqn* achieve lower waiting times by a small margin. There are a few conclusions that we can draw from here. The requests accepted by the CNN approaches have a longer duration, therefore occupying resources for longer, which can be expected as duration is a multiplying factor in a profit calculation. Another more obscure indication may be that the model learned to leave requests waiting longer to increase demand artificially. This last scenario is improbable as the difference between the average values is pretty low.

5.4.2 Arrival rate & Slice duration

Returning to Figure 5.3 we can see that a more considerable job duration benefits our CNN approach, as we can find more significant differences in accumulated profit, especially when coupled with a lower arrival rate such as in $\lambda = 1 \& \mu = 100$. When we look at this specific case, we notice that a bigger job duration allocates resources for longer, which means we will be blocked from accepting new requests as we do not support overbooking resources. Therefore, choosing a better quality request which yields better profit is essential. This is further exacerbated by the case where fewer requests come by (lower arrival rates). Moreover, a smaller arrival rate ($lambda = 1$) benefits our vision-inspired approach as, most likely, all requests within the queue will fit into our matrix state, allowing the model a complete view of the request queue. Higher λ values increase queue size enough to eventually obscure the model of other available options.

Shifting our focus to some QoS metrics, Figure 5.6 shows the total amount of accepted requests at the end of the 1K time steps. These are averaged over ten executions and measured at the end of the last training epoch. Interestingly, a smaller job duration provides a lower value of accepted requests. This finding is expected due to the similar case explained earlier in this section: jobs will hold onto resources for longer, disabling the InP of accepting new ones. The opposite happens for smaller μ values, and we have higher job rotation and request acceptance. There are several instances where the number of accepted requests for both CNN models is lower than the baseline and *ddqn*. Still, when we compare it with Figure 5.3 we still achieve higher accumulated profit. This profit is excellent evidence of the learned policy choosing better requests. Although there are cases where accepting the most amount of requests possible is the way to go, cases such as $\lambda = 1$ & $\mu = 5$ and

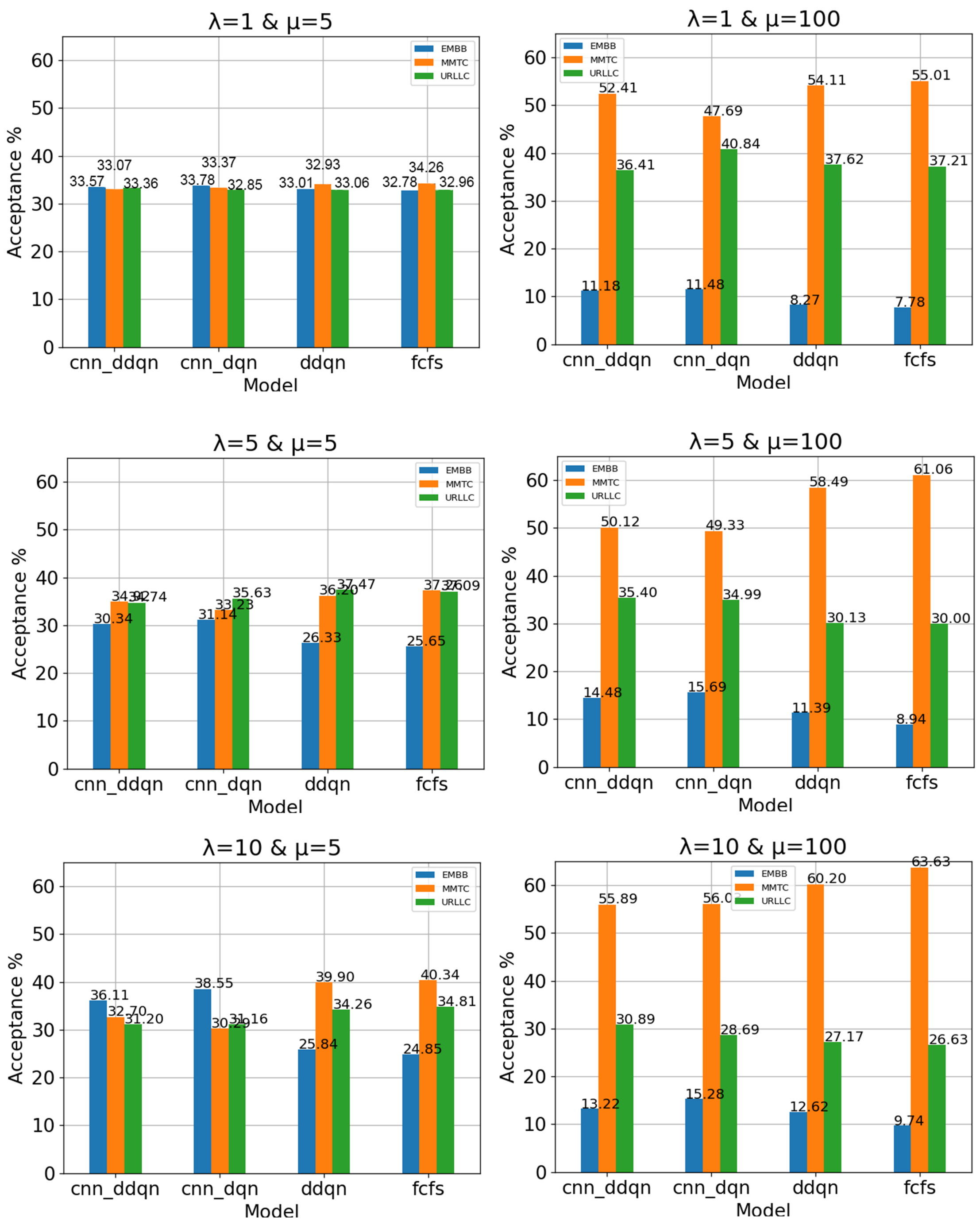


Figure 5.5: Overall average percentage of acceptance for each service class. The values are an average of the final epoch of the models for each dataset.

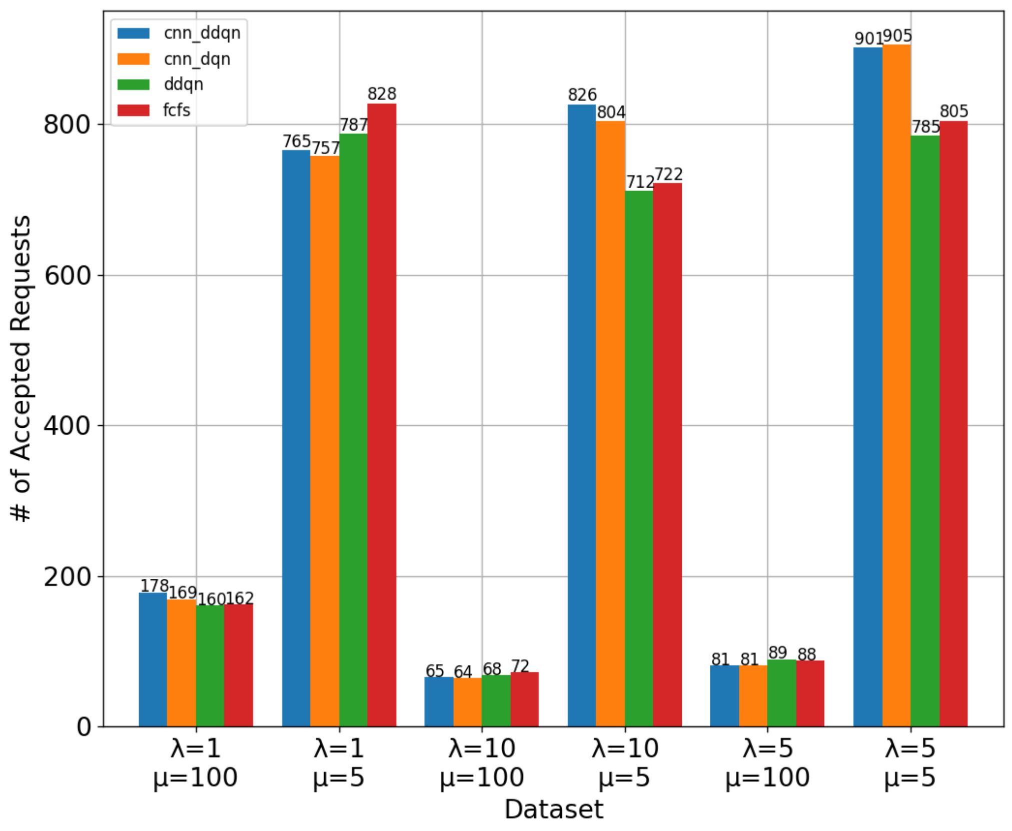


Figure 5.6: Total amount of accepted requests for the final training epoch for each model and dataset.

$\lambda = 10$ & $\mu = 100$ show that sometimes, quality is better than quantity.

To help us visualize that models are effectively learning different ρ we introduce Figure 5.6. It displays the proportion, in percentages, of the accepted requests for each service class. It is important to remind the viewer that since the request class is uniformly selected, all classes have a very similar total amount of requests (33% each). Even though many of the datasets follow a similar distribution, for example, accepting more MMTC requests than the others, they still accept at different rates. CNN's approaches seem to favor a more balanced approach. We want to highlight the scenario of $\lambda = 10$ & $\mu = 5$, where the learned policy percentages are quite different. While *ddqn* and *fcfs* selecting mostly MMTC and URLLCC services, the *cnn_ dqn* and *cnn_ ddqn* learned to provide higher profit through a policy that accepts more EMBB services.

5.4.3 Hyperparameter Discussion

Table 5.4.3 displays the RL main parameters after we performed a customized Random Search coupled with Grid Search (for a few parameters) through RayTune. We decided to apply different searches for each dataset scenario to visualize its differences and what they would take from each other. Looking closely, one might notice

Dataset	$\lambda = 10$ **&** $\mu = 5$	$\lambda = 10$ **&** $\mu = 100$	$\lambda = 1$ **&** $\mu = 5$	$\lambda = 1$ **&** $\mu = 100$	$\lambda = 5$ **&** $\mu = 5$	$\lambda = 5$ **&** $\mu = 100$
γ	0.1	0.1	0.1	0.1	0.4	0.1
Replay Type	priority	balance	balance	balance	priority	balance
Replay Size	50k	5K	5K	5K	50k	5K
Update Steps	73	40	40	40	22	40
Exploration	7211	5000	5000	5000	3301	5000
Learning Rate	0.0003	0.001	0.001	0.001	0.016	0.001
Max Norm	1.5	2.0	2.0	2.0	1.5	2.0
Batch Size	16	16	16	16	16	16
Epochs	100	30	30	30	60	30
Image Height	10	5	5	5	10	5

Table 5.2: Reinforcement Learning parameters at the end of Ray Tune Random Search.

several datasets with the same configuration. After finishing the exhaustive random search, we would compare the best model against the one we handcrafted the parameters through analysis and observation of behavior thought the development of the system. For several datasets, the handcrafted version supplied us with better reward and model convergence than the random search option. Therefore, we deployed them with such values, which is why the rows contain equal values. Still, there are a few models where a different configuration provided better results such is the case for $\lambda = 10$ & $\mu = 5$ and $\lambda = 5$ & $\mu = 5$.

The γ value was introduced before in Section 2.2.4 and indicates the weight the model should put in short-term vs. long-term reward. Interestingly, most models perform better on lower γ, indicating that short-term rewards should be weighted more than long-term.

The *Replay type* is related to one of the three implemented replay buffers discretized at Section 5.3.3. Our strategy for creating a *balanced experience replay* proves successful as most of the models achieved better results with it. Prioritized experience replay also seemed to be a good alternative for datasets. *Replay size* is the buffer size of our replay buffer fixed length.

Update Steps is correlated to the number of time steps we wait until updating the target network. *Exploration* refers to the decaying factor of the ϵ-greedy algorithm, also explained in Section 2.2.4. *Learning Rate* relates to normal DNN training and so does *Batch Size* and *Epochs*. *Max Norm* is related to a technique called gradient clipping [132] which limits our gradients to a minimum or maximum value. In this case, we only define a maximum border. After some initial testing, many models had trouble converging their losses. Therefore, the gradient clipping technique was used to stabilize training and help their convergence.

Image Height is the size of our state input, which is also related to the number of actions we can take at a given time t. One would expect that bigger image matrices would provide a better view of the system for the mode, which, coupled with a bigger action set, would mean better request selection. We can see that is not the case as we only have the sizes of 5 and 10 being selected as the better ones.

5.5 Concluding Remarks

The work proposed in this part of the thesis introduced a dynamic admission control environment with varying request attributes from service classes. We propose a solution by applying DNN in an RL setting with the goal of maximizing the overall InP profit. Proper modeling of profit is an essential piece missing from previous work, which we achieved by developing a dynamic pricing scheme inspired by ride-hailing platforms' pricing.

Our experiments with varying attributes from the environment showed that modeling the state via a vision-inspired approach and choosing a proper architecture benefited the DNN model in finding better policies than the previous work and baseline. Moreover, our generic dynamic pricing scheme allows for better reward evaluation and a realistic view of the respective scenario.

Chapter 6

Improving transparency on Fake News Classification

6.1 Motivation

With the explosion of social networks and widespread information access, news articles and posts can be found pretty much anywhere by the masses. As anyone can publish, read and share, the dissemination of fake news has become a considerable concern. It has been studied that malicious hand-made information can influence elections and sabotage public trust on various subjects [20]. Moreover, as online news outlets and social media deploy recommendation algorithms, it intensifies phenomena called *Echo Chamber Effect* [133, 134]. News feeds constantly learn from user behavior based on several characteristics and information. Based on these features, the algorithm returns several articles that the user is more likely to enjoy, creating the echo chamber effect. The effect happens because the recommendation system may generate a feedback loop of biased or malicious news articles from their current social bubble, enforcing a confirmation bias.

Several recent works [135–137] have tackled the problem of automatic fake news classification by applying different algorithms to learn the contents of the article. DNNs have received much attention and shown great promise in classification problems for Natural Language Processing (NLP) [138].

However, fake news provides unique challenges different from the usual NLP problems that not only create difficulties for classification algorithms but also for creating viable datasets. For example, fake news articles are sometimes purposely written to mislead the reader with the goal of spreading information that may benefit a group or someone. There are instances where the news might revolve around satires, outdated information, or rumors, which provides a complicated distinction between factual and fake content that makes it difficult to label and adequately provide a complete supervised dataset [139]. Furthermore, the massive stream of continuous data may be noisy and coupled with multi-modal articles with video, images, and text mixed together. All of these problems provide significant challenges for fake news classification.

When a system cannot explain why the prediction was made, it is said to be a black box. Most of the state-of-the-art DNN algorithms are usually worried only about classification accuracy metrics while providing zero transparency, therefore being a black box [135, 140, 141].

For the problem of fake news, increasing model transparency is of utmost importance as it deals with several sensitive matters such as censorship. Providing human-interpretable explanations to users may help increase trust in the algorithm. Furthermore, models that provide some explanation are usually preferred over models that do not due to their additional transparency [21]. For the developer, a transparent model can help understand its weakness and target specific components.

The field of explainable AI (XAI) and interpretability have been successfully applied in previous supervised problems [141]. It helps the user understand the inner workings of models by explaining the decisions made. There are to main lines of techniques we can deploy to increase model transparency, these are: *intrinsic* and *post-hoc* [142].

Intrinsic techniques are the ones where we have structures in place within the model that provide some explainability. For example, tree-based algorithms naturally support a feature importance metric. Attention units in DNNs are another essential structure that can derive feature importance. Unfortunately, these techniques require the model structure to be modified to improve transparency. However, This approach could harm classification accuracy. On the other hand, *post-hoc* techniques allow us to keep any model structure and apply algorithms that probe information from within the structures or evaluate predictions correlations to their inputs to infer explanations. For example, LIME [143] derives an interpretable model from the original classifier by using input perturbations to a set of original inputs to produce explanations via feature contributions to a given prediction.

This work aims to provide a methodology for applying a post-hoc analysis technique coupled with an interpretability technique into a state-of-the-art fake news classification DNN model. By increasing transparency, we can study the contributions of different data modalities and how their features interact so the model can arrive at a classification result. Moreover, as we will show in our experiments and results, we gathered insightful information that helped us take action and improve model performance in one of the datasets.

With that in mind, we selected a state-of-the-art fake news classifier called *SAFE* introduced by Zhou *et al.* [144], a post-hoc technique called *Layer-Wise Relevance*

Propagation (LRP) presented earlier by Montavon *et al.* [145] and, a interpretability technique called *representation erasure* [140]. SAFE was able to achieve the highest F1 score metrics on the FakeNewsNet datasets [146] which will also be using to provide a parallel to the model original results.

The main contributions of the work from this chapter are as follows:

- Better F1 score than earlier works based on improved preprocessing and architecture specification;
- Novel *representation erasure* strategy based on LRP scores;
- Analysis of classification logic of the black-box model by using post-hoc explainability techniques;
- Relating model prediction behavior to dataset distribution;
- A insightful discussion on future fake news dataset construction.

The remainder of this chapter starts with Section 6.2 presenting related work within the scope of fake news classification and a few prior works that worried about improving model transparency. Section 6.3 discusses our proposed methodology while giving a brief insight into our chosen techniques and model. Section 6.4 evaluates our approach by providing insightful discussion on feature scores and dataset distribution. Finally, Section 6.5 concludes the chapter with some final remarks. Part of the results of this chapter were published in Ferreira *et al.* [147].

6.2 Related Work

In this section, we will provide an overview of some related works on the fake news detection/classification problem. Some of which provide some explainability and interpretability analysis. The area of fake news classification is divided into two different classes of algorithms called *propagation-based* and *content-based.* Each algorithm's main difference is how they use article data to perform the job.

Propagation-based algorithms are usually deployed in a social media-like context and require control information from the post and the user responsible for trying and finding a connection between fake news spreaders. The goal is to learn about fake news through the information that is not precisely about the article's contents but its social context (retweets, likes, user followers, and others). Structures like propagation graphs and trees have been used together with ML to learn about social media propagation information [148, 149]. Unfortunately, since propagation-based algorithms greatly depend on the information usually available after the post has been out for some time, it fails to detect fake news in the early stages.

Content-based algorithms use the article's actual contents as inputs to learn or detect latent characteristics that may help discern the fake from real information. The content may come in various modalities such as image, audio, video, and others. Despite the fact that many models use only textual information [150–152], multi-modality content-based detection tends to result in higher accuracy [144, 153–155]. Differently than propagation-based, we can properly classify and take action earlier by learning ingrained information on how fake news is written.

Wang *et al.* [155] presents EANN, a multi-modal adversarial DNN fake news classifier that uses text and images from two different social media datasets and applies a content-based algorithm. The network uses TextCNN [156] to process the textual inputs and a pre-trained VGG-19 [157] to process image inputs from within the article content. It also uses an event discriminator to create a min-max game that enables adversarial learning between the multi-modal network portion and the event discriminator.

Jin *et al.* [154] proposes an RNN with LSTM called Att-RNN. The network uses the social media post content text and social context to classify the post correctly. Since it is also a multi-modal algorithm, the images are processed in a modified VGG-19 while the text is converted into an embedded representation. An Attention mechanism [158] in the network is used to learn a correlation between the image, text, and social context network representation.

Until now, we have explored binary classification as the only way to classify fake news correctly. However, a few works try a different approach where several classes are taken into account. For example, Ghosh *et al.* [159] defines it as a 3-class problem considering *fake, suspicious or legit* as the available ways of classifying an article. It deploys a neural network with two submodules. The first sub-module is known as the knowledge base as it has an FC hidden layer with Relu activation, which is responsible for classifying the news into one of the classes. The second sub-module uses a bidirectional LSTM DNN to learn the article's writing style.

Note that all of these works provide solutions for classifying the problem, but the models remain a black box. Therefore we are unable to identify how the models perceive such news. Jin *et al.* uses Attention units, which have been previously deployed to increase model transparency, but instead, it chooses to only focus on classification accuracy.

As an example of attention being used to improve transparency, Kai *et al.* [142] introduces the DEFEND framework. The framework uses a DNN with a co-attention mechanism. Attention is used to learn the news content and user comment information. The attention weights provide a way of ranking the top-k sentences and increase model transparency. However, relying on user comments to classify news articles falls into the propagation-based algorithm category. On the other hand, Reis *et al.* [136] applies clustering together with SHapley Additive exPlanations (SHAP) [160] to provide insightful explanations on fake news complexity. The work uses XGBoost as its classifier to study different features.

Unlike the last two works, we apply an LRP explainability technique merged with a representation erasure interpretability technique. We aim to provide a novel methodology for increasing black-box model transparency.

6.3 Increasing transparency of Fake News Model

This section introduces both techniques applied to the SAFE model and a brief review of the network itself. Furthermore, we elaborate on the changes we made to the model and how the techniques were applied to enable transparency.

6.3.1 Explainability & Interpretability Techniques

Measuring explanation quality is an exciting problem currently open as it can be highly subjective. It is greatly damped by the fact that many calculations to measure performance use an indicator or ground truth reference value to estimate the goodness of the solution, which is not straightforward to find when dealing with model explanations [141]. Therefore, a common practice is to provide and analyze human-interpretable data to the user is to enable the model with some feature importance algorithm which layer can be translated into color-coded highlighted words for user visualization.

Before introducing the techniques we applied, it is essential to clearly distinguish between explainability and interpretability as many works use such terms interchangeably. Usually, interpretability is associated with cause and effect. By looking at an input without knowing anything about the model, we can derive why we got the result it produced. Therefore, we did not have to learn or fully comprehend internal mechanics to derive an explanation. On the other hand, explainability is usually deployed inside black box models by using its internal structure to derive explanations of what a given input returns the specific output. Even though the distinction seems straightforward, there are instances where we fall into a blurred line where it is hard to define the technique. Nevertheless, for the purpose of this work, we are considering LRP to be an explainability technique as it deals with the internals of DNNs, while Representation Erasure alone is in the realm of interpretability as it associates cause and effect without examining the model specifics.

Layer-wise Relevance Propagation: LRP is a post-hoc explainability technique that takes advantage of feed-forward neural networks with support for back-propagation. The algorithm is based on Deep Taylor Decomposition [161] and uses the network prediction propagated backward by applying a rule that follows a conservation property analogous to electrical circuits [161]. The property requires that information received by a neuron at layer l is redistributed to $l-1$ until we arrive back into the input layer. Each neuron will calculate its own *relevance score*. Once we reach the initial layer, we can use their scores to infer the input features relevance scores associated with which neuron they were fed into.

Regarding the rules by which the scores are propagated, it will use neuron weights and connections as the basis for calculation. Moreover, recent research on LRP found that different rules for different DNN layers [135] might provide better input relevance scores.

To deploy LRP, we used the python INNvestigate framework [162] which contains a vast disposition of different LRP rules. Based on the insight in Kohlbrenner *et al.* [135] we use LRP-ϵ on dense layers and LRP-$\alpha\beta$ on CNN layers to render better final feature scores.

Representation Erasure: The Erasure technique is a generic procedure that can be deployed independently of the network. It was previously applied with success in Li *et al.* 's [140] work in a bi-direction LSTM DNN to provide transparency on the model decisions.

The algorithmic procedure consists of removing or masking features from the samples before inputting them into the model and analyzing how that would change the model output. For example, we could black out or remove a mouth of a face in an image input and see how the network reacts. If there is an increase in miss predictions, we can derive that the removed feature is essential for the network to make its decision. Notice that even though it is a different format for feature importance definition, it does not need any information from the black box, providing interpretability.

In theory, this procedure should be performed for all samples and all possible combinations of feature removals, as features may correlate with each other. This constraint will require a prohibitive amount of computation, making it impractical for problems with a high enough amount of features.

6.3.2 SAFE: Similarity-Aware Multi-modal Fake News Detection

The DNN SAFE presented in Zhou *et al.* [144] takes advantage of three main modules to produce state-of-the-art results on the FakeNewsNet dataset. Before we get into the details of the implementation, let us first define the fake news problem: Given a news article A with textual information (T) and a matching visual feature (V), let $A = \{T, V\}$. SAFE aims to predict whether A should be labeled (Y) as fake or real news.

SAFE mainly receives three inputs, two of which are text (news headline and body-text) and one from visual information (image attached to the news article). For dealing with the image, instead of learning it via a CNN layer as previous work did [154, 156], we use a separate pre-trained network called *image2sentence* [163] to generate a caption (text sentence) for the image. This preprocessing step allows the network to receive all three separate inputs (headline, body-text, image caption) as a text.

The original SAFE implementation uses a pre-trained *word2vector* model Glove [164] to generate the text embedding. These are fed to an extended TextCNN, comprised of a CNN using a ReLU activation followed by a Max Pooling and an FC layers. The extended comes from the fact that the authors decided to add the FC layer to the classic TextCNN as it helps with learning the feature extracted by the convolution layer. The output of these FC layers is concatenated and fed into the last FC layer with a *softmax* function that makes the class prediction.

The network minimizes a loss function (L) by using a weighted sum of two other losses. The first one (L_p) uses a cross-entropy-based loss to match the output of the softmax layer with the ground truth label. And is presented in Equation 6.1 below.

$$\begin{aligned} L_p = -E_{(a,y)\sim(A,Y)}(&ylogM_p(t,v) \\ &+(1-y)log(1-M_p(t,v))) \end{aligned} \tag{6.1}$$

Where $M_p(t,v)$ is the output of the softmax layer based on the textual (t) and visual (v) representations produced by previous layers, and y is the ground truth label.

Equation 6.1 displays the second loss. Its main idea is to use a slightly modified *cosine* similarity to match the textual features vector with the image captions feature vector. Therefore, if the image content of the news is different from the written content, this might provide some insight that the news is fake. We will denote the modified cosine similarity output as M_s. The second loss consists of applying the same cross-entropy-based function of L_p but on M_s, producing the equation below.

$$\begin{aligned} L_s = -E_{(a,y)\sim(A,Y)}(&ylogM_s(t,v) \\ &+(1-y)log(1-M_s(t,v))) \end{aligned} \tag{6.2}$$

The loss function the network minimizes uses weights κ and ι to control the influence of both L_p and L_s as show in Equation 6.3. These weights are different depending on which dataset from FakeNewsNet the network is deployed on.

$$L = \kappa L_p + \iota L_s \tag{6.3}$$

6.3.3 A Transparent SAFE$-\beta$ Model

To enable transparency into SAFE, we modified the neural network slightly and reimplemented all code into Tensorflow's Keras to enable support for the INNvestigate framework. As our version provides a few different points to the original SAFE, we will refer to it as SAFE-β. Moreover, SAFE-β achieved a slightly higher F1 score than SAFE during our experiments.

At this stage, it is crucial to clarify that since our inputs are mainly textual data, we will refer to different words, punctuation, and other special characters as *token* or *token words*. Just 'word' may be misleading to the reader as the textual data is not only comprised of it.

One of our objectives when changing SAFE and its preprocessing steps was to preserve the original input as close to the source as possible. Our decision was inspired by a few works [20, 165] which hinted that writing style could be an important factor in differentiating real from fake news.

It is a common practice in NLP to have an initial Embedding Layer to transform all text data into meaningful numbers that can be fed into the model. In the original SAFE, all the embedding procedure is done as a preprocessing step by using a pre-trained word vector in conjunction with SIF [166]. After further looking into SIF, we found that it skips embedding generation of tokens not in the training vocabulary[1]. Therefore, for SAFE-β we decided to apply Gensim's Word2Vec [167] to load the exact pre-trained Glove embeddings without the use of SIF. Moreover, we generated a random Gaussian distribution embedding for out-of-vocabulary words with a mean of 0.0 and a standard deviation of 0.6.

We apply word tokenization and data padding only for other usual text processing steps. It is commonplace to perform stemming and lower casing in NLP, but these are considered destructive techniques as they modify the words to provide a more standard text format.

SAFE-β network architecture diverges from the original SAFE as we removed an FC layer without activation that sits before the CNN pass. The layer's goal was to transform the input size before feeding them into their respective extended TextCNNs. In our modification, data is fed straight into the CNN layer.

We also produced a version called *SAFE-β Text-only* which does not take the image caption into consideration. We remove the whole network portion related to processing the image input for this version. This removal includes *image2sentence*

[1]https://github.com/PrincetonML/SIF/issues/5

preprocessing, its extended TextCNN, and the concatenation with the other two inputs. Furthermore, there will be no need to apply L_s loss as its sole purpose was to find a similarity between the image caption and the article's text and headline. Therefore our final L will only apply Equation 6.1. Our insight into a text-only version came about because of our initial evaluations performed because of the increased transparency we provided into the model, which we will get into in Section 6.4.

To enable transparency to the model during our experiments, we first performed training on a portion of the dataset while testing/validating it on another to ensure the network achieved the expected accuracy. Afterward, we run a second pass over the entire dataset with LRP enabled, but instead of performing a new training, we are now using our trained network from the previous step. With LRP enabled, we can collect relevance scores for all available token embeddings. Notice that embeddings for each token are comprised of several values, each of which has a correlated relevance score. To find the token's relevance score, we add the scores of the embedding values. Providing a more complex mechanism to derive explanations from embeddings to tokens is non-trivial and has been previously explored by Monti *et al.* [137]. In general, more simplistic approaches are adopted.

We further improve transparency by using representation erasure and basing our feature removal process on token relevance scores to decrease the computation requirements. For each article sample, we select N words to remove. To choose such words, we first look at the relevance scores produced by LRP on the experiments from Subsections 6.4.1 and 6.4.2. We proceed to remove the highest scored N words for that sentence/sample. We repeat the process on all the samples and evaluate their impact on the F1 score. By using the relevance scores to guide us through the best feature combination removals, we can significantly decrease and limit the computational cost of the algorithm, making it a viable interpretability option.

6.4 Experiments & Results

To evaluate our proposed methodology, we will analyze and discuss our findings on experiments related to the LRP relevance scores. Afterward, we will debate accuracy metrics and our representation erasure approach results. We also briefly discuss how the results found through transparency can help future dataset construction.

All experiments were carried out in the same FakeNewsNet [146] dataset version used in the original SAFE. As we explained earlier, creating comprehensive supervised datasets for fake news classification is not a trivial matter. FakeNewsNet was an initiative to help fake news-related research by collecting two distinct datasets that were labeled based on fact-checking websites. One dataset used the *Gossipcop* fact-checking website to retrieve both the news and labels., while the other was retrieved from *Politifact* fact-checking website. We will refer to the datasets as per the name of their respective website.

The Gossipcop dataset is the big one with more than 20K samples and a proportion of around 3:1 of real news to fake news. Politifact, on the other hand, is much smaller with around 1K samples but a better ratio of 1:1. Due to SAFE-β input layers, we use the article headline, body-text, and associated image to generate the features to our model. Still, both datasets also contain spatial-temporal information as well as social media context.

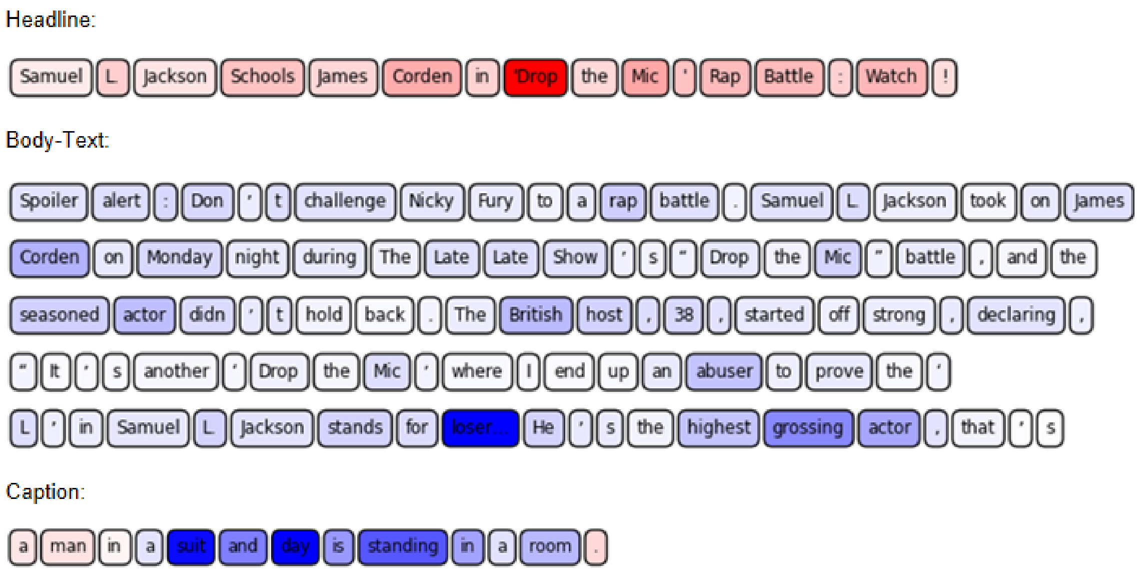
We performed a random split on the data to perform training and test. 80% of the data was used for training, while the 20 % was left for testing. The network was trained for ten epochs, with a learning rate of $5e^{-4}$ and a batch size of 64. Our procedures for relevance score calculation and representation erasure were explained in Section 6.3.3. All experiments were carried on a Intel(R) Xeon(R) Gold6246 CPUs @ 3.30GHz, 256GBRAM and two NVIDIA Quadro RTX 8000 GPUs.

As we dive deep into the discussion of relevance scores, we will present several tables with their value normalized. The normalization was done through a min-max approach for the scores within the same dataset. In this chapter, we will present the top 10 highest scores on the tables, but the reader can find the top-100 token sorted tables in Appendix C. We did not include a full sorted table because the total amount of tokens is in the order of thousands.

At this stage, we must distinguish the different classification options we will evaluate. True Positives (TP) represent real news classified correctly as real news by the model. True Negatives (TN) are fake news articles classified as fake news by the model. False Positives (FP) are real news articles miss classified by the model as fake news, while False Negatives (FN) represent the opposite, fake news articles miss classified as real news.

6.4.1 Explainability on Gossipcop

SAFE-β: We start by evaluating the more extensive dataset, Gossipcop. Table 6.1 presents top-ranked words by their relevance score for headlines, body-text, and image captions separately. As we look at the scores, we can clearly see a distortion with the highest values skewed over to the Headline input. On the other hand, body-text and caption seem to have a similar importance for making predictions. We can say it is expected to have headlines as a vital part of the article as it usually conveys the central concept and summary. It has been reported that fake news uses alarming and clickbaity headlines to gather more access and increase the spread of misinformation [165].


Headline:

Samuel L Jackson Schools James Corden in 'Drop the Mic ' Rap Battle : Watch !

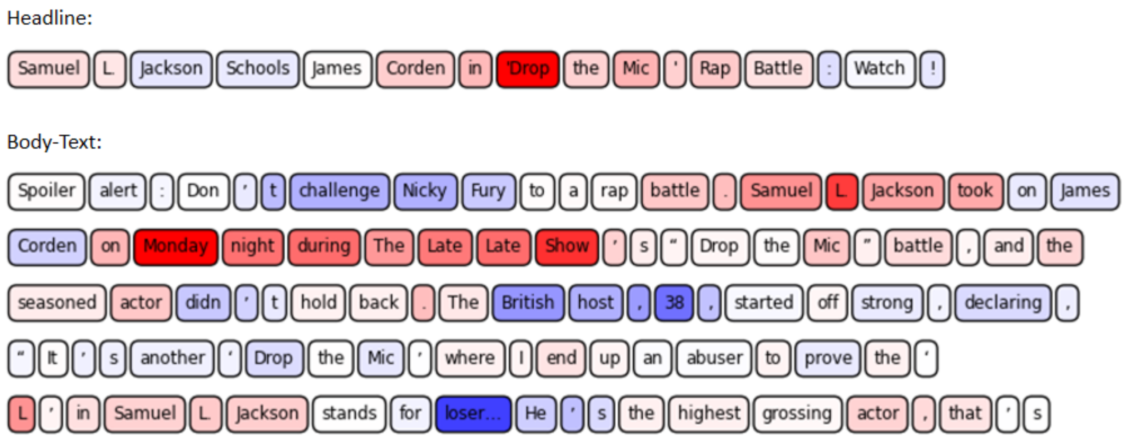
Body-Text:

Spoiler alert : Don ' t challenge Nicky Fury to a rap battle . Samuel L Jackson took on James Corden on Monday night during The Late Late Show ' s " Drop the Mic " battle , and the seasoned actor didn ' t hold back . The British host , 38 , started off strong , declaring , " It ' s another ' Drop the Mic ' where I end up an abuser to prove the ' L ' in Samuel L Jackson stands for loser... He ' s the highest grossing actor , that ' s

Caption:

a man in a suit and day is standing in a room .

Figure 6.1: Word Importance when using Gossipcop with SAFE-β


Headline:

Samuel L Jackson Schools James Corden in 'Drop the Mic ' Rap Battle : Watch !

Body-Text:

Spoiler alert : Don ' t challenge Nicky Fury to a rap battle . Samuel L Jackson took on James Corden on Monday night during The Late Late Show ' s " Drop the Mic " battle , and the seasoned actor didn ' t hold back . The British host , 38 , started off strong , declaring , " It ' s another ' Drop the Mic ' where I end up an abuser to prove the ' L ' in Samuel L Jackson stands for loser... He ' s the highest grossing actor , that ' s

Figure 6.2: Word Importance when using Gossipcop with SAFE-β text-only

Gossipcop (SAFE-β)					
Headline		Body-Text		Image Caption	
Word	Score	Word	Score	Word	Score
Wikinow	1.0000	notifications	0.2526	cell	0.2684
Walderdorff	0.9024	news	0.2392	phone	0.2651
Migos	0.8611	push	0.2380	wii	0.2503
Arrowverse	0.8603	Get	0.2377	laptop	0.2487
USweekly	0.8589	E	0.2355	woman	0.2481
timothee	0.8471	features	0.2354	tennis	0.2477
Acne-Prone	0.8438	with	0.2352	controller	0.2477
Dion-na	0.8393	,	0.2337	remote	0.2465
Stormi-Inspired	0.8283	!	0.2329	frisbee	0.2464
chalamet	0.8181	.	0.2328	baseball	0.2461

Table 6.1: Top-10 tokens and their normalized relevance scores for each of the three text inputs on Gossipcop Dataset using SAFE-β

The image caption is the second most important feature, even though it is reasonably close in score values to Body-text. One possible reason the caption does not carry more weight in the model decision is the generation procedure SAFE performs during preprocessing. As we pass the original image through the *image2sentence* network, we generate a caption text that describes the image. Unfortunately, the caption may come with noise or be wrong altogether as the network will never be 100% accurate. Moreover, there is a limitation on how complex sentences generated can be, which means we deal with several repeated words and short sentences.

An unexpected result is for the relevance scores for body-text as they seem to be the lowest out of the three. One would think that the actual text content of the article would be a critical feature for the network to discern between the binary classes. Figure 6.1 displays a random sample taken from the data and color-coded from red to white to blue. The more vivid the red is, the more positive the impact such a word had on the prediction, while blue represents the opposite. Looking at the image, we can clearly see how one-sided the features look in importance.

Based on this insight found through the increased transparency of the model, we decided to remove one of the inputs. As we just discussed, the caption generation procedure seems to be the one most prone to fault, which might be clouding the model in learning some different aspects of the news. Therefore, we deployed SAFE-β Text Only, previously explained in Section 6.3.3, which used only Headlines and Body-text as inputs.

SAFE-β Text-Only: The SAFE-β Text-Only network was retrained and tested on the same samples from SAFE-β. Figure 6.2 shows the heatmap of text-only for the same sample from Figure 6.1. Just by evaluating this one sample, we can already see a better distribution of importance between the tokens, showing a more expected relevance score mapping. Table 6.2 confirms our suspicions as the broad distribution of scores between body-text and headlines is more healthy. Note that headlines are still a more important feature for the same reasons as before, which, as we are going to show further, remains a trend for the other experiments. Moreover, removing caption increased our overall F1-score for this dataset as we will get into Section 6.4.3.

Another interesting conclusion to point out is that even though we achieved a higher F1 score, the SAFE-β with caption approach still yielded a high enough F1. Therefore, by comparing the different distributions of relevance scores produced by both models, we have strong evidence that there is more than one way of correctly classifying the same news article.

As we have found, SAFE-β Text-Only performs significantly better for Gossip-cop. We further dive deeper into the model decisions by analyzing specific prediction cases, such as those predicted correctly (TP & TN) and the ones where the classifier fails (FP & FN). Table 6.2 ranks the top 10 highest scoring tokens for all available prediction cases and both headline and body-text inputs. Evaluating the highest scoring words allows us to see what are the most important themes or patterns the algorithm is looking for to perform classification.

Interestingly, the highest-ranking body-text tokens for TN predictions consist mainly of website links or website names. Moreover, all these websites are from gossip-related news, likely containing a lot of click baits and rumor news that might have been proved to be fake. It is fair to assume that the network most likely learned this behavior through training as most of the website data within the dataset are fake, therefore creating a high association between the tokens and the fake news classification, increasing the overall relevance score produced by LRP.

Going over some of the other tokens also found in headline we can find many gossip-related names and words such as *Kardashian*, *Brad* and *Aniston*, and for Body-Text *exclusive*, *Gossip* and *insider*. Different token forms in upper case for *exclusive* appear high on the list, which we would probably not have if we decided to apply destructive techniques such as stemming and lower-casing as preprocessing options.

Below the top-10 in Table C.3 the trend continues for fake news as we see words like *rumor* and *divorce*. However, there are also some high placements for punctuation tokens like '*!*', '*?*' and '*:*' appearing for TP as well. It is interesting to see the different positions and scores punctuation possesses as it shows different writing styles the network is picking on to make a decision.

As we devote our attention to the top tokens for misclassifications, it is possible to notice that most of the tokens for both headline and body-text have lower overall score values, except for the first token in the FP column. That supports evidence

Gossipcop (SAFE-β Text-only)							
True Positive		True Negative		False Positive		False Negative	
Word	Score	Word	Score	Word	Score	Word	Score
Headline							
18-49	0.9414	Celebrities	0.9799	chalamet	1.0000	Pattinson	0.7434
'guilty	0.8613	Pattinson	0.8727	timothee	0.8507	Aniston	0.6282
Soundsational	0.8579	react	0.8282	'One	0.8190	'She	0.5277
Kufrin	0.8556	Aniston	0.7891	Tweeden	0.6969	Jennifer	0.5267
'Hollywood	0.8433	Kidman	0.7221	'callous	0.6867	Stefani	0.5172
'Reputation	0.8194	Stefani	0.7209	Jam-Packed	0.6711	Pitt	0.5165
Ankle-Wrap	0.8193	'World	0.6938	Makeup	0.6658	'So	0.5072
Boob-Baring	0.8130	'has	0.6893	Season	0.6472	Lopez	0.5064
Zendaya	0.8094	de	0.6891		0.5979	Divorce	0.5038
USweekly	0.7886	&	0.6288	Post-Pregnancy	0.5902	Beckham	0.5011
Body-Text							
notifications	0.9288	HollywoodLife.com	0.6962	low-vamp	0.5418	RadarOnline.com	0.4420
White/The	0.6091	RadarOnline.com	0.5771	Angelillo/UPI	0.5235	well—Us	0.4250
push	0.5654	HollywoodLife	0.4731	kimkardashian	0.5125	insider	0.3733
Aquazzura	0.5622	RadarOnline	0.4590	Vernoff	0.5119	source	0.3713
track-inspired	0.5616	Gossip	0.4582	Bendjima	0.5102	Rumors	0.3657
E	0.5492	Inc.	0.4559	_Andrea_Ant	0.4913	Aniston	0.3654
10-months-of	0.5452	resolved.	0.4504	him—at	0.4754	rumors	0.3634
Rodriguez/Getty	0.5438	insider	0.4267	Mazur/Getty	0.4745	Broadimage	0.3582
Nadinne	0.5419	EXCLUSIVELY	0.4156	Kardashian–Jenner	0.4588	Radar	0.3542
Scoopnest	0.5391	EXCLUSIVE	0.4155	background.2nd	0.4576	TMZ	0.3514

Table 6.2: Top 10 Words for each prediction type on Gossipcop using only Politifact using only Headline and Body-text.

that the network is not as confident when making these predictions. We can also find a lot of TN tokens appearing in the FN table, which may justify the miss prediction. Moreover, they appear with a lower score than in the TN, again showing the model's lower confidence in making these predictions.

In short, the Gossipcop dataset appears to be following a trend for the fake news articles as in a celebrity/gossip environment. The model seems to piggyback on this distribution of words and use it to its advantage by learning the most critical tokens and achieving high accuracy results in doing so. Most of the miss predictions are made with less overall confidence, highlighted by the lower score values on similar words used to make a correct prediction.

6.4.2 Explainability on Politifact

Before we evaluate our results with the Politifact dataset, it is important to clarify that over 60% of the images attached to news articles used in the original SAFE FakeNewsNet Politifact dataset were broken. Since this dataset already contains a significantly smaller amount of samples and our previous findings showed that image caption generation is causing noise for the model, we decided only to study the effects of LRP relevance scores for our SAFE-β Text Only model. We still include the SAFE-β accuracy metrics in Section 6.4.3 for proper discussion. Moreover, our F1 score improved by removing the image caption, supporting this decision even

Politifact (SAFE-β Text-only)							
True Positive		True Negative		False Positive		False Negative	
Word	Score	Word	Score	Word	Score	Word	Score
Headline							
Republican	0.8727	forgot...	1.0000	presidential	0.8065	'Regular	0.8816
presidential	0.8369	'just	0.9721	Democrats	0.7853	'Never	0.8062
debate	0.8248	Wiping	0.9612	election	0.5924	Teen-Ager	0.7663
Economic	0.7888	Admits	0.8958	2020	0.5718	Miami	0.7003
Congressional	0.7715	Attacked	0.8878	Senator	0.5391	Found	0.6937
Barack	0.7702	Fired	0.8761	in	0.4898	Shore	0.6645
President	0.7530	Accused	0.8717	Biden	0.4722	!	0.6606
Outlook	0.7430	Arrested	0.8654	government	0.4619	Of	0.6349
Transcript	0.7330	president... They	0.8624	States	0.4494	Jersey	0.6288
Sen.	0.7269	Her	0.8467	of	0.4482	Mercurial	0.6178
Body-Text							
678-8511	0.6596	Trendolizer™	0.4193	Senate	0.5218	tire-melting	0.4010
Rating	0.6276	al-Islam	0.4074	Act	0.4985	RadarOnline.com	0.4000
Outlook	0.6116	implanted	0.4046	actionnetwork.org/	0.4965	suspected	0.3966
Remove	0.6062	Accuser	0.4040	Judiciary	0.4910	southern-fried	0.3962
cqrollcall.com	0.5950	Trump	0.4027	Committee	0.4906	murder	0.3960
Fiscal	0.5865	Hayhoe	0.4024	intervening.	0.4813	Trump	0.3956
Economic	0.5762	diagnosed	0.4010	NVRA	0.4773	teen-ager	0.3950
Watch	0.5744	surgically	0.4006	source=	0.4697	Maj.	0.3945
browser	0.5633	blood	0.4006	nowrapper=true	0.4671	flock	0.3934
transcript	0.5343	Etemad	0.4003	Registration	0.4658	'	0.3927

Table 6.3: Top-10 tokens for each prediction type on Politifact using only Headline and Body-text.

further.

SAFE-β Text Only: Table 6.3 presents the top 10 token for the Politifact dataset for both inputs. Similar to our findings for the Gossipcop dataset, the score values are better distributed, but there is a clear edge over headlines. When comparing the table's TN and TP portions, we can see that TN has a bigger disparity between the two input types. This finding aligns with what we previously alluded to about fake news being more likely to provide sensationalist headlines.

When evaluating the tokens presented in the TN portion of the table, there is a constant trend of words associated with negative connotations, such as *Attacked*, *Fired* and *Arrested*. For TP, the headline scores a lower, and the body-text are higher when compared to FN. Although we can see some website links and names appearing, it does not seem to be such a heavy trend as for the Gossipcop dataset. Below the top-10 tokens, we can find punctuation tokens in ':' and ',' used to discern for both TP and TN, reinforcing the idea of writing style being a significant factor in classifying news articles.

The results should be taken with a grain of salt regarding miss classification in FP and FN. Since the model has over 80% of accuracy and the dataset is pretty tiny, our amount of misclassified samples is minimal. However, we can still draw some conclusions. For example, we see similar tokens again from FP in TP, which might be a justification for miss classification. As for FN, we can see a similar trend to the TN portion of the table due to negative connotation words like *murder* and *suspected* in the Body-Text input.

6.4.3 Accuracy and F1

Dataset	**Gossipcop**			**Politifact**		
Model	SAFE-β	SAFE-β Text-Only	SAFE*	SAFE-β	SAFE-β Text-Only	SAFE*
F1	0.8897	0.9153	0.895	0.8171	0.8505	0.896
Accuracy	0.8288	0.8652	0.838	0.8153	0.8566	0.874
Recall	0.9181	0.9533	0.937	0.8534	0.8438	0.93
Precision	0.8744	0.8822	0.857	0.8041	0.8621	0.889

Table 6.4: Different Metric results for both FakeNewsNet datasets. SAFE* values are the ones reported from the original paper, while SAFE-β is the tree input version and SAFE-β Text-Only represents our Headline and Body-text only version

When dealing with a binary classification problem, it is important to evaluate the accuracy, recall, and precision, especially if the dataset is unbalanced, as is the case for Gossipcop. F1-Score is our preferred metric as it considers recall and precision via a harmonic mean. Table 6.4 presents all measured accuracy-related metrics for our tested approaches and the original SAFE reported values. The measurements consist of a mean of 10 different executions.

As discussed in the previous experimental sections, image caption seems to be interfering with the learning process of the network due to noisy caption generation by *image2sentence*. We saw a meaningful increase in all metrics for both datasets to support this conclusion. Another interesting point is that for the Gossipcop dataset, we achieved higher accuracy metric values using our text-only version of the model. As for the Politifact, our results were lower, partly because we could not retrieve all images from the articles or because training is unstable to the small number of samples.

6.4.4 Representation Erasure

We implemented our representation erasure experiment to study further the different impacts of the most critical tokens and individual inputs, further increasing the model transparency. To properly evaluate the token erasure impact, we performed the training and inference without the token for all available samples and measured their correspondent F1-score for both datasets.

The experiments were executed separately by selecting N tokens to remove at each experiment, where $N = 0, 1, 2, ..15$. Notice that $N = 0$ is equivalent to the results presented in Table 6.4 as we are using all available tokens. We performed the experiment for each individual input and removed tokens of headline only, body-text

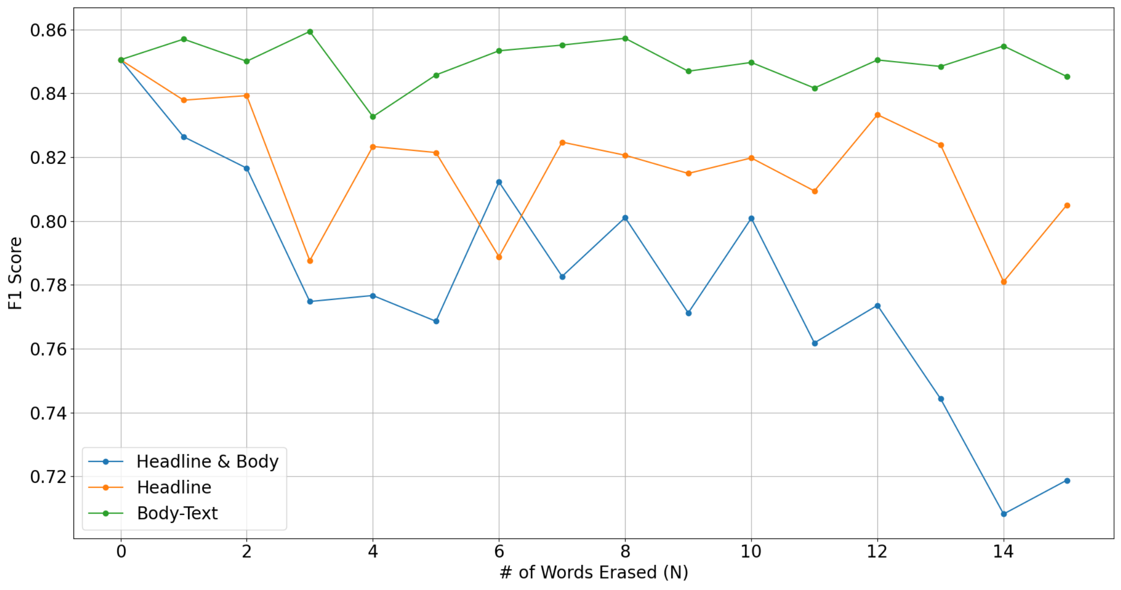


Figure 6.3: Representation Erasure for Politifact with SAFE-β text-only varying removal of 0 to 15 tokens. Orange line is for erasures on headline only, green is for body-text only, while blue represents erasing words on both inputs.

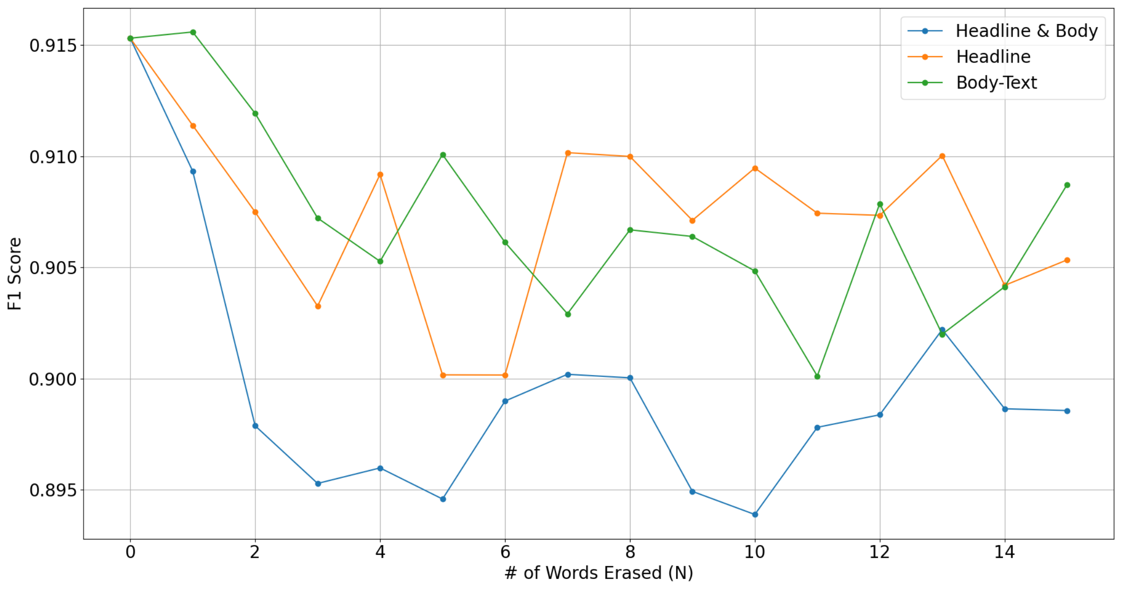


Figure 6.4: Representation Erasure for Gossipcop with SAFE-β text-only varying removal of 0 to 15 tokens. Orange line is for erasures on headline only, green is for body-text only, while blue represents erasing words on both inputs.

only, and then removal of tokens of both. The selection of token removal is based on the values of the highest relevance scores based on LRP as explained in Section 6.3.3. All F1 score results are an average of 5 executions with the correspondent number of removals N.

Figure 6.3 presents our results for the Politifact dataset. The results show that removing headline tokens seems to impact the model F1 score more than body-text removal. This score aligns with our previous findings that headlines are more crucial than the other features. Furthermore, the article's body-text contains a higher amount of tokens which may make the removal less impactful. This result is also more compelling evidence that there are multiple ways a classifier may learn about the article to make its prediction. As expected, the removal of tokens from both inputs simultaneously greatly hinders the model's capacity to continue predicting well as the token removal stacks up.

Figure 6.4 shows the same metrics for the Gossipcop dataset. Here the results are not as clear. For some initial removals up to $N = 3$ we follow the same pattern of headline token removal being more significant than body-text, but afterward, the removals seem less impactful. The pattern for removal of tokens does achieve the lowest scores than single removals, but it also seems to plateau after the same N. If we shift our scope and look to the y-axis range, we notice it is a much smaller scale than the Politifact plot. The smaller scale indicates that, even though there is a drop in F1, the model can still learn how to classify the samples of the dataset properly. One major factor could be the sheer size of Gossicop compared to the Politifact, posing 20 times more samples which greatly helps with network training.

6.4.5 Impacts of transparency for future Datasets construction

In this subsection, we will take our significant findings provided by the increased model transparency and discuss how some could be related to how the dataset is built. We hope our discussion could improve for future fake news dataset construction.

One noticeable takeaway from both datasets is that there is a common tendency for data to be labeled as fake news. Politifact had a bad-connotation word trend, while Gossipcop contained celebrity and gossip-related tokens. A way to mitigate this problem is to make sure the dataset contains a somewhat equal amount of different news topics. A procedure like this would allow for more diverse data that could also fit into the News Category Classification problem [2]. Categorizing the article could be applied as preprocessing step to cluster and build more specialized fake

[2]https://www.kaggle.com/rmisra/news-category-dataset

news detection models or be deployed as an individual feature that could improve performance.

Another interesting takeaway was that Headlines seem to be the primary source for the model to extract its knowledge. Punctuation tokens also appeared on both datasets to indicate different writing styles, contributing to both TP and TN predictions. To decrease some of the bias over headlines and the impact of writing styles, one might use asentiment analysis algorithm to study how different samples may be distributed for each class. This process can help create a better distribution of tokens throughout the dataset.

6.5 Concluding Remarks

The work presented in this chapter introduced a methodology for improving the transparency of a state-of-the-art Fake News Classifier by using explainability and interpretability techniques. With the increased transparency, we were able to study model decisions, including mispredictions. We found that headlines play a critical role in model predictions. Even though we could not properly evaluate image caption with Politifact, we demonstrated that there are multiple ways of learning and inferencing over the same sets while still predicting with reasonable accuracy.

We also briefly discussed how our findings show possible bias for the datasets we experimented on. We provided solutions to help mitigate these conditions for future dataset creation. We hope our findings stimulate future works on machine learning to provide transparency on their model, as it could provide perspective on the fake news problem.

Chapter 7

Conclusion

The introduction of new technological solutions to meet society's growing demand creates a slew of new challenges and concerns. These concerns create opportunities for new hardware and software solutions that must be optimized to fit the new environment. The increasing development and state-of-the-art solutions provided by various ML algorithms appear to provide significant opportunity to model rising complex problems and improve current existing solutions. In this thesis, we studied several emerging problems and introduced new techniques and approaches to tackle the new problems that arise from them involving various strategies to deal with increasingly complex data.

SSDs are a growing technology because they overcome several limitations of previous HDDs storage devices. Its main selling point is the faster throughput powered by NAND Flash Media. With the introduction of a different storage structure, several issues arose internally, which were dealt with clever algorithmic techniques such as ECCs and GC. However, some issues innate to Flash Media architecture persist and require optimization, such as reading-level threshold voltage calibration, which is an essential step in decreasing bit errors.

With that in mind, this thesis provided a viable solution for dynamically adjusting such thresholds using ML algorithms. We initially obtain a dataset via flash chip characterization, allowing us to model our ML algorithms with accurate data, which is different from previous solutions. Moreover, we introduce time-series modeling of the problem to better grasp different factors present in voltage shifting. The ML approach improved against the standard technique of keeping a static voltage threshold while achieving results very close to the optimal average. Moreover, XGBoost variations were performing slightly better as we moved into deeper DR regions.

Due to the various challenges within SSD management, the storage device requires more powerful hardware with extra computational capabilities. Initially, developers took advantage of the computational power to share the CPU load between SSD management and user application. A few years later, CSDs were introduced to the market to enable storage devices with dedicated computational capabilities to execute user applications. Its proximity to the device allows for extra parallelism within the entire system and an overall decrease in data movement if wholly taken advantage of. In this body of work, we provided a study of different classes of ML algorithms that perform object tracking. Coupling object tracking with in-storage processing can provide significant benefits as many of these applications execute at the edge and with power constraints. Therefore, by bringing the processing closer to the device, we can optimize power consumption because of less data movement. Our experiments showed that many of the models executing within *in situ* CPU can still maintain a real-time FPS while achieving lower power consumption on a per-frame basis. For example, WiSARD, a WNN, had good accuracy (IoU) for the tested dataset while having one of the lowest power consumption of 1.32×10^{-2} Joules per frame. Moreover, we also took advantage of a CSD to perform experiments involving our dynamic voltage adjustment solution to evaluate timing and memory constraints. The platform can provide a good window for collecting SSD firmware data on-the-fly similar to our offline characterization procedure.

Another technology growing in popularity and current in deployment is 5G Networks. However, both industry and researchers are already speculating how NextG networks may improve on the current problems 5G was unable to solve. It is greatly expected that ML will be even further integrated and essential for 6G and further. NFV and SDN are technologies that currently provide some level of Network Slicing, allowing providers to offer their resources as services. Nevertheless, to properly slice its resources, it must select which requests it will accept, as its resources are finite. The selection of such requests is considered an admission control problem and requires a policy and a goal to maximize something important to the provider.

In this thesis, we formalize a dynamic environment which differs greatly from previous work which made the assumptions that request attributes do not vary with time and revenue does not change with demand. We introduced two different CNN DQN architectures with novel modeling of the environment to serve as their input, as well as action set size linked with such state. Moreover, to maximize the provider's profit, we formalize a Dynamic Pricing scheme that considers supply and demand, as well as requests resource requirements and attributes. Our results showed that one of the models from previous work could not find suitable policies in the proposed dynamic environment. At the same time, our vision-inspired state and action set coupled with the CNN DQN architecture achieved higher profit and learned better policies for different arrival rates and job duration.

Fake News classification is another interesting topic that has been growing in our daily lives. Fake News can be created to mislead the reader, which may cause long-lasting damage in different areas of society. Therefore, many researchers continually look for a way to detect such articles properly. Again, ML algorithms have been successful in related areas linked with Fake News, such as NLP. However, purely detection and stopping the article deals with complicated issues such as censorship. Therefore, to increase trust in an ML model, it is vital to learn how it performs the classification and its relationship with the inputs.

We proposed a generic methodology for black-box feed forwarding ML models, which significantly increases transparency. Our solution deploys two different algorithms to learn which features are more likely to be important for the model to make its prediction. We were able to detect several biases and distribute information about the used dataset. Moreover, the newly enabled transparency allowed an insight for removing one of the inputs, which led to an overall increase in the accuracy metrics.

References

[1] J. Do, V. C. Ferreira, H. Bobarshad, M. Torabzadehkashi, S. Rezaei, A. Heydarigorji, D. Souza, B. F. Goldstein, L. Santiago, M. S. Kim, *et al.*, "Cost-effective, energy-efficient, and scalable storage computing for large-scale ai applications," *ACM Transactions on Storage (TOS)*, vol. 16, no. 4, pp. 1–37, 2020.

[2] "Kingston 3d nand memory cells." `https://www.kingston.com/br/blog/pc-performance/difference-between-slc-mlc-tlc-3d-nand`. Accessed: 2022-05-31.

[3] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, M. Hasan, B. C. Van Essen, A. A. Awwal, and V. K. Asari, "A state-of-the-art survey on deep learning theory and architectures," *Electronics*, vol. 8, no. 3, p. 292, 2019.

[4] E. Adamopoulou and L. Moussiades, "An overview of chatbot technology," in *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pp. 373–383, Springer, 2020.

[5] M.-C. Yang, Y.-M. Chang, C.-W. Tsao, P.-C. Huang, Y.-H. Chang, and T.-W. Kuo, "Garbage collection and wear leveling for flash memory: Past and future," in *2014 International Conference on Smart Computing*, pp. 66–73, IEEE, 2014.

[6] L.-P. Chang, "On efficient wear leveling for large-scale flash-memory storage systems," in *Proceedings of the 2007 ACM symposium on Applied computing*, pp. 1126–1130, 2007.

[7] K. Zhao, W. Zhao, H. Sun, X. Zhang, N. Zheng, and T. Zhang, "{LDPC-in-SSD}: Making advanced error correction codes work effectively in solid state drives," in *11th USENIX Conference on File and Storage Technologies (FAST 13)*, pp. 243–256, 2013.

[8] B. Peleato, R. Agarwal, J. Cioffi, M. Qin, and P. H. Siegel, "Towards minimizing read time for nand flash," in *2012 IEEE Global Communications Conference (GLOBECOM)*, pp. 3219–3224, IEEE, 2012.

[9] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Threshold voltage distribution in mlc nand flash memory: Characterization, analysis, and modeling," in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1285–1290, IEEE, 2013.

[10] N. Papandreou, T. Parnell, H. Pozidis, T. Mittelholzer, E. Eleftheriou, C. Camp, T. Griffin, G. Tressler, and A. Walls, "Using adaptive read voltage thresholds to enhance the reliability of mlc nand flash memory systems," in *Proceedings of the 24th edition of the great lakes symposium on VLSI*, pp. 151–156, 2014.

[11] J. Do, Y.-S. Kee, J. M. Patel, C. Park, K. Park, and D. J. DeWitt, "Query processing on smart ssds: Opportunities and challenges," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pp. 1221–1230, 2013.

[12] I. Jo, D.-H. Bae, A. S. Yoon, J.-U. Kang, S. Cho, D. D. Lee, and J. Jeong, "Yoursql: a high-performance database system leveraging in-storage computing," *Proceedings of the VLDB Endowment*, vol. 9, no. 12, pp. 924–935, 2016.

[13] Y. Kang, Y.-s. Kee, E. L. Miller, and C. Park, "Enabling cost-effective data processing with smart ssd," in *2013 IEEE 29th symposium on mass storage systems and technologies (MSST)*, pp. 1–12, IEEE, 2013.

[14] S. Seshadri, M. Gahagan, M. S. Bhaskaran, T. Bunker, A. De, Y. Jin, Y. Liu, and S. Swanson, "Willow: A user-programmable ssd.," in *OSDI*, pp. 67–80, 2014.

[15] D. Tiwari, S. Boboila, S. S. Vazhkudai, Y. Kim, X. Ma, P. Desnoyers, and Y. Solihin, "Active flash: towards energy-efficient, in-situ data analytics on extreme-scale machines.," in *FAST*, pp. 119–132, 2013.

[16] S. J. Nawaz, S. K. Sharma, S. Wyne, M. N. Patwary, and M. Asaduzzaman, "Quantum machine learning for 6g communication networks: State-of-the-art and vision for the future," *IEEE access*, vol. 7, pp. 46317–46350, 2019.

[17] S. Mumtaz, J. M. Jornet, J. Aulin, W. H. Gerstacker, X. Dong, and B. Ai, "Terahertz communication for vehicular networks," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 7, 2017.

[18] K. B. Letaief, W. Chen, Y. Shi, J. Zhang, and Y.-J. A. Zhang, "The roadmap to 6g: Ai empowered wireless networks," *IEEE communications magazine*, vol. 57, no. 8, pp. 84–90, 2019.

[19] M. Z. Chowdhury *et al.*, "6g wireless communication systems: Applications, requirements, technologies, challenges, and research directions," *IEEE Open Journal of the Communications Society*, vol. 1, 2020.

[20] X. Zhou and R. Zafarani, "A survey of fake news: Fundamental theories, detection methods, and opportunities," *ACM Computing Surveys (CSUR)*, vol. 53, no. 5, pp. 1–40, 2020.

[21] S. Gholizadeh and N. Zhou, "Model explainability in deep learning based natural language processing," *arXiv preprint arXiv:2106.07410*, 2021.

[22] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error characterization, mitigation, and recovery in flash-memory-based solid-state drives," *Proceedings of the IEEE*, vol. 105, no. 9, pp. 1666–1704, 2017.

[23] L. Shi, Y. Di, M. Zhao, C. J. Xue, K. Wu, and E. H.-M. Sha, "Exploiting process variation for write performance improvement on nand flash memory storage systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 1, pp. 334–337, 2015.

[24] V. Mohan *et al.*, *Modeling the physical characteristics of NAND flash memory.* PhD thesis, Citeseer, 2010.

[25] W. Wang, T. Xie, and D. Zhou, "Understanding the impact of threshold voltage on mlc flash memory performance and reliability," in *Proceedings of the 28th ACM international conference on Supercomputing*, pp. 201–210, 2014.

[26] N. Mielke, H. P. Belgal, A. Fazio, Q. Meng, and N. Righos, "Recovery effects in the distributed cycling of flash memories," in *2006 IEEE International Reliability Physics Symposium Proceedings*, pp. 29–35, IEEE, 2006.

[27] M. Calabrese, C. Miccoli, C. M. Compagnoni, L. Chiavarone, S. Beltrami, A. Parisi, S. Bartolone, A. L. Lacaita, A. S. Spinelli, and A. Visconti, "Accelerated reliability testing of flash memory: Accuracy and issues on

a 45nm nor technology," in *Proceedings of 2013 International Conference on IC Design & Technology (ICICDT)*, pp. 37–40, IEEE, 2013.

[28] V. Mohan, T. Siddiqua, S. Gurumurthi, and M. R. Stan, "How i learned to stop worrying and love flash endurance," in *2nd Workshop on Hot Topics in Storage and File Systems (HotStorage 10)*, 2010.

[29] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, "An experimental study of data retention behavior in modern dram devices: Implications for retention time profiling mechanisms," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 60–71, 2013.

[30] Y. Cai, Y. Luo, E. F. Haratsch, K. Mai, and O. Mutlu, "Data retention in mlc nand flash memory: Characterization, optimization, and recovery," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pp. 551–563, IEEE, 2015.

[31] L. Shi, K. Wu, M. Zhao, C. J. Xue, D. Liu, and E. H.-M. Sha, "Retention trimming for lifetime improvement of flash memory storage systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 1, pp. 58–71, 2015.

[32] Y. Cai, Y. Luo, S. Ghose, and O. Mutlu, "Read disturb errors in mlc nand flash memory: Characterization, mitigation, and recovery," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 438–449, IEEE, 2015.

[33] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "A large-scale study of flash memory failures in the field," *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, no. 1, pp. 177–190, 2015.

[34] K. Eshghi and R. Micheloni, "Ssd architecture and pci express interface," in *Inside solid state drives (SSDs)*, pp. 19–45, Springer, 2013.

[35] Y. Lee, H. Yoo, I. Yoo, and I.-C. Park, "6.4 gb/s multi-threaded bch encoder and decoder for multi-channel ssd controllers," in *2012 IEEE International Solid-State Circuits Conference*, pp. 426–428, IEEE, 2012.

[36] B. Schroeder, R. Lagisetty, and A. Merchant, "Flash reliability in production: The expected and the unexpected," in *14th USENIX Conference on File and Storage Technologies (FAST 16)*, pp. 67–80, 2016.

[37] I. Narayanan, D. Wang, M. Jeon, B. Sharma, L. Caulfield, A. Sivasubramaniam, B. Cutler, J. Liu, B. Khessib, and K. Vaid, "Ssd failures in datacenters:

What? when? and why?," in *Proceedings of the 9th ACM International on Systems and Storage Conference*, pp. 1–11, 2016.

[38] O. online, "Open nand flash interface specification," 2017. , Accessed: 03/29/2022.

[39] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[40] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[41] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[42] M. A. Nielsen, *Neural networks and deep learning*, vol. 25. Determination press San Francisco, CA, USA, 2015.

[43] R. C. Staudemeyer and E. R. Morris, "Understanding lstm–a tutorial into long short-term memory recurrent neural networks," *arXiv preprint arXiv:1909.09586*, 2019.

[44] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[45] I. Aleksander, M. De Gregorio, F. M. G. França, P. M. V. Lima, and H. Morton, "A brief introduction to weightless neural systems.," in *ESANN*, pp. 299–305, Citeseer, 2009.

[46] R. J. Mitchell, J. Bishop, and P. R. Minchinton, "Optimising memory usage in n-tuple neural networks," *Mathematics and computers in simulation*, vol. 40, no. 5-6, pp. 549–563, 1996.

[47] I. Aleksander, W. Thomas, and P. Bowden, "Wisard· a radical step forward in image recognition," *Sensor review*, 1984.

[48] L. A. Lusquino Filho, L. F. Oliveira, A. Lima Filho, G. P. Guarisa, L. M. Felix, P. M. Lima, and F. M. França, "Extending the weightless wisard classifier for regression," *Neurocomputing*, vol. 416, pp. 280–291, 2020.

[49] K. M. M. J. Khaki, *Weightless Neural Networks For Face and Pattern Recognition: An Evaluation Using Open-Source Databases*. PhD thesis, Citeseer, 2013.

[50] D. S. Carvalho, H. C. Carneiro, F. M. França, and P. M. Lima, "B-bleaching: Agile overtraining avoidance in the wisard weightless neural classifier.," in *ESANN*, 2013.

[51] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

[52] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[53] M. A. Wiering and M. Van Otterlo, "Reinforcement learning," *Adaptation, learning, and optimization*, vol. 12, no. 3, p. 729, 2012.

[54] M. L. Puterman, "Markov decision processes," *Handbooks in operations research and management science*, vol. 2, pp. 331–434, 1990.

[55] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.

[56] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[57] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*, pp. 1995–2003, PMLR, 2016.

[58] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2219–2225, IEEE, 2006.

[59] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[60] W. Fedus, P. Ramachandran, R. Agarwal, Y. Bengio, H. Larochelle, M. Rowland, and W. Dabney, "Revisiting fundamentals of experience replay," in *International Conference on Machine Learning*, pp. 3061–3071, PMLR, 2020.

[61] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, no. 3, pp. 293–321, 1992.

[62] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.

[63] M. Srinivasan, "Flashcache," 2014. , Accessed: 1/9/2020.

[64] J. Janukowicz, "How new qlc ssds will change the storage landscape," tech. rep., Micron, 5 Speen Street, Framingham, MA 01701, USA, October 2018. , Accessed: 12/7/2019.

[65] Z. Mei, K. Cai, and X. He, "Deep learning-aided dynamic read thresholds design for multi-level-cell flash memories," *IEEE Transactions on Communications*, vol. 68, no. 5, pp. 2850–2862, 2020.

[66] B. Peleato, R. Agarwal, J. M. Cioffi, M. Qin, and P. H. Siegel, "Adaptive read thresholds for nand flash," *IEEE Transactions on Communications*, vol. 63, no. 9, pp. 3069–3081, 2015.

[67] M. Rajab, J.-P. Thiers, and J. Freudenberger, "Read threshold calibration for non-volatile flash memories," in *2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin)*, pp. 109–113, IEEE, 2019.

[68] R. Ma, F. Wu, M. Zhang, Z. Lu, J. Wan, and C. Xie, "Rber-aware lifetime prediction scheme for 3d-tlc nand flash memory," *IEEE Access*, vol. 7, pp. 44696–44708, 2019.

[69] F. Mahdisoltani, I. Stefanovici, and B. Schroeder, "Improving storage system reliability with proactive error prediction," in *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference*, vol. 1, pp. 391–402, 2017.

[70] L. Zuolo and R. Micheloni, "Regression neural network for identifying threshold voltages to be used in reads of flash memory devices," Jan. 27 2022. US Patent App. 17/089,891.

[71] E. H. Wilson, M. Jung, and M. T. Kandemir, "Zombienand: Resurrecting dead nand flash for improved ssd longevity," in *2014 IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems*, pp. 229–238, IEEE, 2014.

[72] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A white paper on neural network quantization," *arXiv preprint arXiv:2106.08295*, 2021.

[73] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *arXiv preprint arXiv:1510.00149*, 2015.

[74] V. C. Ferreira, A. S. Nery, L. A. Marzulo, L. Santiago, D. Souza, B. F. Goldstein, F. M. França, and V. Alves, “A feasible fpga weightless neural accelerator,” in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2019.

[75] J. Gantz and D. Reinsel, “The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east,” *IDC iView: IDC Analyze the future*, vol. 2007, no. 2012, pp. 1–16, 2012.

[76] S. Abu-El-Haija, N. Kothari, J. Lee, P. Natsev, G. Toderici, B. Varadarajan, and S. Vijayanarasimhan, “Youtube-8m: A large-scale video classification benchmark,” *arXiv preprint arXiv:1609.08675*, 2016.

[77] D. Beaver, S. Kumar, H. C. Li, J. Sobel, P. Vajgel, *et al.*, “Finding a needle in haystack: Facebook’s photo storage.,” in *OSDI*, vol. 10, pp. 1–8, 2010.

[78] Y. Wu, J. Lim, and M.-H. Yang, “Object tracking benchmark,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 9, pp. 1834–1848, 2015.

[79] D. Held, S. Thrun, and S. Savarese, “Learning to track at 100 fps with deep regression networks,” in *European Conference on Computer Vision*, pp. 749–765, Springer, 2016.

[80] D. Bolme, “Visual object tracking using adaptive correlation filters,” in *The IEEE Conference on Computer Vision and Pattern Recognition CVPR) Year*, 2010.

[81] R. Lima De Carvalho, D. S. C. Carvalho, F. Mora-Camino, P. V. M. Lima, and F. M. G. França, “Online tracking of multiple objects using WiSARD,” in *ESANN 2014, 22st European Symposium on Artificial Neural Networks, Computational Intelligence And Machine Learning*, (Bruges, Belgium), pp. pp 541–546, Apr. 2014. http://www.i6doc.com/fr/livre/?GCOI=28001100432440.

[82] A. Borji, M.-M. Cheng, Q. Hou, H. Jiang, and J. Li, “Salient object detection: A survey,” *Computational visual media*, pp. 1–34, 2019.

[83] I. Report, "The digitization of the world from edge to core," *https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf*, 2018.

[84] W. Jiang, B. Han, *et al.*, "The road towards 6g: A comprehensive survey," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 334–366, 2021.

[85] T. F. Thomas Bittman, Neil MacDonald, "How to Overcome Four Major Challenges in Edge Computing." `https://www.gartner.com/doc/reprints?id=1-1XWDQ2PW&ct=191210&st=sb`, 2019. [Online; accessed 21-Jan-2020].

[86] SNIA, "Computational storage technical working group," 2019. , Accessed: 05/03/2019.

[87] J. Do, S. Sengupta, and S. Swanson, "Programmable solid-state storage in future cloud datacenters," *Communications of the ACM*, vol. 62, no. 6, pp. 54–62, 2019.

[88] M. Torabzadehkashi, S. Rezaei, V. Alves, and N. Bagherzadeh, "Compstor: An in-storage computation platform for scalable distributed processing," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 1260–1267, IEEE, 2018.

[89] B. Gu, A. S. Yoon, D.-H. Bae, I. Jo, J. Lee, J. Yoon, J.-U. Kang, M. Kwon, C. Yoon, S. Cho, *et al.*, "Biscuit: A framework for near-data processing of big data workloads," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 153–165, 2016.

[90] G. Koo, K. K. Matam, H. Narra, J. Li, H.-W. Tseng, S. Swanson, M. Annavaram, *et al.*, "Summarizer: trading communication with computing near storage," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 219–231, ACM, 2017.

[91] M. Torabzadehkashi, S. Rezaei, A. Heydarigorji, H. Bobarshad, V. Alves, and N. Bagherzadeh, "Catalina: In-storage processing acceleration for scalable big data analytics," in *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pp. 430–437, IEEE, 2019.

[92] M. Torabzadehkashi, A. Heydarigorji, S. Rezaei, H. Bobarshad, V. Alves, and N. Bagherzadeh, "Accelerating hpc applications using computational storage devices," in *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Confer-*

ence on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), pp. 1878–1885, IEEE, 2019.

[93] M. Torabzadehkashi, S. Rezaei, A. HeydariGorji, H. Bobarshad, V. Alves, and N. Bagherzadeh, "Computational storage: an efficient and scalable platform for big data and hpc applications," *Journal of Big Data*, vol. 6, no. 1, p. 100, 2019.

[94] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 3, pp. 583–596, 2014.

[95] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[96] H. L. França, J. C. P. da Silva, O. Lengerke, M. S. Dutra, M. D. Gregorio, and F. M. G. França, "Movement persuit control of an offshore automated platform via a ram-based neural network," *2010 11th International Conference on Control Automation Robotics & Vision*, pp. 2437–2441, 2010.

[97] D. N. Do Nascimiento, R. Lima De Carvalho, F. Mora-Camino, P. V. M. Lima, and F. M. G. Franca, "A WiSARD-based multi-term memory framework for online tracking of objects," in *ESANN 2015, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, (Bruges, Belgium), pp. 978–287587014–8, Apr. 2015.

[98] S. Rezaei, K. Kim, and E. Bozorgzadeh, "Scalable multi-queue data transfer scheme for fpga-based multi-accelerators," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*, pp. 374–380, Oct 2018.

[99] S. Rezaei, E. Bozorgzadeh, and K. Kim, "Ultrashare: Fpga-based dynamic accelerator sharing and allocation," in *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pp. 1–5, IEEE, 2019.

[100] S.-W. Jun, M. Liu, S. Lee, J. Hicks, J. Ankcorn, M. King, S. Xu, and Arvind, "Bluedbm: An appliance for big data analytics," in *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ISCA '15, (New York, NY, USA), pp. 1–13, ACM, 2015.

[101] S. Rezaei, C. Hernandez-Calderon, S. Mirzamohammadi, E. Bozorgzadeh, A. Veidenbaum, A. Nicolau, and M. J. Prather, "Data-rate-aware fpga-based acceleration framework for streaming applications," in *2016 Inter-*

national Conference on ReConFigurable Computing and FPGAs (ReConFig), pp. 1–6, Nov 2016.

[102] W. Cao, Y. Liu, Z. Cheng, N. Zheng, W. Li, W. Wu, L. Ouyang, P. Wang, Y. Wang, R. Kuan, *et al.*, "{POLARDB} meets computational storage: Efficiently support analytical workloads in {Cloud-Native} relational database," in *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pp. 29–41, 2020.

[103] D. Kwon, D. Kim, J. Boo, W. Lee, and J. Kim, "A fast and flexible hardware-based virtualization mechanism for computational storage devices," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pp. 729–743, 2021.

[104] A. Barbalace and J. Do, "Computational storage: Where are we today?," in *CIDR*, 2021.

[105] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[106] J. Valmadre, L. Bertinetto, J. Henriques, A. Vedaldi, and P. H. Torr, "End-to-end representation learning for correlation filter based tracking," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2805–2813, 2017.

[107] I. Aleksander, M. D. Gregorio, F. M. G. França, P. M. V. Lima, and H. Morton, "A brief introduction to weightless neural systems," in *ESANN 2009, 17th European Symposium on Artificial Neural Networks*, 2009.

[108] G. Ning, Z. Zhang, C. Huang, X. Ren, H. Wang, C. Cai, and Z. He, "Spatially supervised recurrent convolutional neural networks for visual object tracking," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4, IEEE, 2017.

[109] B. Li, J. Yan, W. Wu, Z. Zhu, and X. Hu, "High performance visual tracking with siamese region proposal network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8971–8980, 2018.

[110] H. Nam and B. Han, "Learning multi-domain convolutional neural networks for visual tracking," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4293–4302, 2016.

[111] M. Kristan, J. Matas, A. Leonardis, T. Vojir, R. Pflugfelder, G. Fernandez, G. Nebehay, F. Porikli, and L. Čehovin, "A novel performance evaluation methodology for single-target trackers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, pp. 2137–2155, Nov 2016.

[112] M. Rasti, S. K. Taskou, H. Tabassum, and E. Hossain, "Evolution toward 6g wireless networks: A resource management perspective," *arXiv preprint arXiv:2108.06527*, 2021.

[113] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications surveys & tutorials*, vol. 18, no. 1, pp. 236–262, 2015.

[114] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2014.

[115] N. Van Huynh, D. T. Hoang, D. N. Nguyen, and E. Dutkiewicz, "Optimal and fast real-time resource slicing with deep dueling neural networks," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1455–1470, 2019.

[116] S. Bakri, P. A. Frangoudis, and A. Ksentini, "Dynamic slicing of ran resources for heterogeneous coexisting 5g services," in *2019 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, 2019.

[117] H. Esmat and B. Lorenzo, "Deep reinforcement learning based dynamic edge/fog network slicing," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*, pp. 1–6, IEEE, 2020.

[118] S. Bakri, B. Brik, and A. Ksentini, "On using reinforcement learning for network slice admission control in 5g: Offline vs. online," *International Journal of Communication Systems*, vol. 34, no. 7, p. e4757, 2021.

[119] P. Henderson, R. Islam, *et al.*, "Deep reinforcement learning that matters," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[120] V. Mnih, K. Kavukcuoglu, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[121] V. C. Ferreira, H. Esmat, B. Lorenzo, S. Kundu, and F. F. MG, "Reinforcement learning based multi-attribute slice admission control for next-generation

networks in a dynamic pricing environment," in *2022 IEEE 95th Vehicular Technology Conference:(VTC2022-Spring)*, pp. 1–5, IEEE, 2022.

[122] A. Thantharate, R. Paropkari, V. Walunj, and C. Beard, "Deepslice: A deep learning approach towards an efficient and reliable network slicing in 5g networks," in *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, pp. 0762–0767, IEEE, 2019.

[123] K. Chounos, A. Apostolaras, and T. Korakis, "A dynamic pricing and leasing module for 5g networks," in *2020 IEEE 3rd 5G World Forum (5GWF)*, pp. 343–348, IEEE, 2020.

[124] Ö. U. Akgül, I. Malanchini, V. Suryaprakash, and A. Capone, "Dynamic resource allocation and pricing for shared radio access infrastructure," in *2017 IEEE International Conference on Communications (ICC)*, pp. 1–7, IEEE, 2017.

[125] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[126] S. Gu, T. Lillicrap, *et al.*, "Q-prop: Sample-efficient policy gradient with an off-policy critic," *arXiv preprint arXiv:1611.02247*, 2016.

[127] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International conference on machine learning*, pp. 1329–1338, PMLR, 2016.

[128] N. C. Luong, P. Wang, *et al.*, "Resource management in cloud networking using economic analysis and pricing models: A survey," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 2, pp. 954–1001, 2017.

[129] "Uber dynamic pricing." `https://www.uber.com/en-GB/blog/uber-dynamic-pricing/`. Accessed: 2022-03-14.

[130] Y. Kim, S. Kim, and H. Lim, "Reinforcement learning based resource management for network slicing," *Applied Sciences*, vol. 9, no. 11, p. 2361, 2019.

[131] H. Maei, C. Szepesvari, S. Bhatnagar, D. Precup, D. Silver, and R. S. Sutton, "Convergent temporal-difference learning with arbitrary smooth function approximation," *Advances in neural information processing systems*, vol. 22, 2009.

[132] T. Mikolov *et al.*, “Statistical language models based on neural networks,” *Presentation at Google, Mountain View, 2nd April*, vol. 80, p. 26, 2012.

[133] S. Mohseni, E. Ragan, and X. Hu, “Open issues in combating fake news: Interpretability as an opportunity,” *arXiv preprint arXiv:1904.03016*, 2019.

[134] M. R. Hershey, “Echo chamber: Rush limbaugh and the conservative media establishment. by kathleen hall jamieson and joseph n. cappella. new york: Oxford university press, 2008. 320p. 22.99 paper.,” *Perspectives on Politics*, vol. 7, no. 2, pp. 415–417, 2009.

[135] M. Kohlbrenner, A. Bauer, S. Nakajima, A. Binder, W. Samek, and S. Lapuschkin, “Towards best practice in explaining neural network decisions with lrp,” in *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7, IEEE, 2020.

[136] J. C. Reis, A. Correia, F. Murai, A. Veloso, and F. Benevenuto, “Explainable machine learning for fake news detection,” in *Proceedings of the 10th ACM conference on web science*, pp. 17–26, 2019.

[137] F. Monti, F. Frasca, D. Eynard, D. Mannion, and M. M. Bronstein, “Fake news detection on social media using geometric deep learning,” *arXiv preprint arXiv:1902.06673*, 2019.

[138] W. Khan, A. Daud, J. A. Nasir, and T. Amjad, “A survey on the state-of-the-art machine learning models in the context of nlp,” *Kuwait journal of Science*, vol. 43, no. 4, 2016.

[139] W. Y. Wang, “" liar, liar pants on fire": A new benchmark dataset for fake news detection,” *arXiv preprint arXiv:1705.00648*, 2017.

[140] J. Li, W. Monroe, and D. Jurafsky, “Understanding neural networks through representation erasure,” *arXiv preprint arXiv:1612.08220*, 2016.

[141] M. Danilevsky, K. Qian, R. Aharonov, Y. Katsis, B. Kawas, and P. Sen, “A survey of the state of explainable ai for natural language processing,” *arXiv preprint arXiv:2010.00711*, 2020.

[142] K. Shu, L. Cui, S. Wang, D. Lee, and H. Liu, “defend: Explainable fake news detection,” in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 395–405, 2019.

[143] M. T. Ribeiro, S. Singh, and C. Guestrin, “" why should i trust you?" explaining the predictions of any classifier,” in *Proceedings of the 22nd ACM*

SIGKDD international conference on knowledge discovery and data mining, pp. 1135–1144, 2016.

[144] X. Zhou, J. Wu, and R. Zafarani, “Safe: Similarity-aware multi-modal fake news detection,” *Advances in Knowledge Discovery and Data Mining*, vol. 12085, p. 354, 2020.

[145] G. Montavon, A. Binder, S. Lapuschkin, W. Samek, and K.-R. Müller, “Layer-wise relevance propagation: an overview,” *Explainable AI: interpreting, explaining and visualizing deep learning*, pp. 193–209, 2019.

[146] K. Shu, D. Mahudeswaran, S. Wang, D. Lee, and H. Liu, “Fakenewsnet: A data repository with news content, social context, and spatiotemporal information for studying fake news on social media,” *Big data*, vol. 8, no. 3, pp. 171–188, 2020.

[147] V. C. Ferreira, S. Kundu, and F. M. França, “Analysis of fake news classification for insight into the roles of different data types,” in *2022 IEEE 16th International Conference on Semantic Computing (ICSC)*, pp. 75–82, IEEE, 2022.

[148] C. Castillo, M. Mendoza, and B. Poblete, “Information credibility on twitter,” in *Proceedings of the 20th international conference on World wide web*, pp. 675–684, 2011.

[149] K. Wu, S. Yang, and K. Q. Zhu, “False rumors detection on sina weibo by propagation structures,” in *2015 IEEE 31st international conference on data engineering*, pp. 651–662, IEEE, 2015.

[150] V. Pérez-Rosas, B. Kleinberg, A. Lefevre, and R. Mihalcea, “Automatic detection of fake news,” *arXiv preprint arXiv:1708.07104*, 2017.

[151] S. Hakak, M. Alazab, S. Khan, T. R. Gadekallu, P. K. R. Maddikunta, and W. Z. Khan, “An ensemble machine learning approach through effective feature extraction to classify fake news,” *Future Generation Computer Systems*, vol. 117, pp. 47–58, 2021.

[152] S. Singhania, N. Fernandez, and S. Rao, “3han: A deep neural network for fake news detection,” in *International conference on neural information processing*, pp. 572–581, Springer, 2017.

[153] D. Khattar, J. S. Goud, M. Gupta, and V. Varma, “Mvae: Multimodal variational autoencoder for fake news detection,” in *The world wide web conference*, pp. 2915–2921, 2019.

[154] Z. Jin, J. Cao, H. Guo, Y. Zhang, and J. Luo, "Multimodal fusion with recurrent neural networks for rumor detection on microblogs," in *Proceedings of the 25th ACM international conference on Multimedia*, pp. 795–816, 2017.

[155] Y. Wang, F. Ma, Z. Jin, Y. Yuan, G. Xun, K. Jha, L. Su, and J. Gao, "Eann: Event adversarial neural networks for multi-modal fake news detection," in *Proceedings of the 24th acm sigkdd international conference on knowledge discovery & data mining*, pp. 849–857, 2018.

[156] Y. Kim, "Convolutional neural networks for sentence classification," 2014.

[157] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[158] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[159] S. Ghosh and C. Shah, "Towards automatic fake news classification," *Proceedings of the Association for Information Science and Technology*, vol. 55, no. 1, pp. 805–807, 2018.

[160] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proceedings of the 31st international conference on neural information processing systems*, pp. 4768–4777, 2017.

[161] G. Montavon, S. Lapuschkin, A. Binder, W. Samek, and K.-R. Müller, "Explaining nonlinear classification decisions with deep taylor decomposition," *Pattern Recognition*, vol. 65, pp. 211–222, 2017.

[162] M. Alber, S. Lapuschkin, P. Seegerer, M. Hägele, K. T. Schütt, G. Montavon, W. Samek, K.-R. Müller, S. Dähne, and P.-J. Kindermans, "innvestigate neural networks!," *Journal of Machine Learning Research*, vol. 20, no. 93, pp. 1–8, 2019.

[163] T. Lan, W. Yang, Y. Wang, and G. Mori, "Image retrieval with structured object queries using latent ranking svm," in *European conference on computer vision*, pp. 129–142, Springer, 2012.

[164] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014.

[165] Y. Chen, N. J. Conroy, and V. L. Rubin, “Misleading online content: recognizing clickbait as" false news",” in *Proceedings of the 2015 ACM on workshop on multimodal deception detection*, pp. 15–19, 2015.

[166] S. Arora, Y. Liang, and T. Ma, “A simple but tough-to-beat baseline for sentence embeddings,” 2017.

[167] R. Řehůřek and P. Sojka, “Software Framework for Topic Modelling with Large Corpora,” in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, (Valletta, Malta), pp. 45–50, ELRA, May 2010. `http://is.muni.cz/publication/884893/en`.

Appendices

Appendix A

Dynamic Read-level Voltage Adjustment Additional Experiments

A.1 Model Parameters

A.1.1 DNN Parameters

DNN Parameters	
dense 1	48
dense 2	96
dense 3	64
dense dropout 1	0.1
dense dropout 2	0.4
dense dropout 3	0.4
lstm dropout 1	0.3
hidden num	3
learning rate	0.0001
lstm 2	208
lstm dropout 2	0.3
lstm 1	256
lstm num	2
epochs	46
batch size	64

Table A.1: DNN with LSTM parameters.

A.1.2 XGBoost Parameters

XGBoost Parameters	
SubSample	0.5
# Estimators	1000
min child weight	5
max depth	10
learning rate	0.01
gamma	2
colsample bytree	0.8
num rounds	995

Table A.2: XGBoost parameters

A.1.3 WiSARD Parameters

ReW Parameters	
Thermometer	128
Address Size	30

Table A.3: Regression WiSARD parameters

A.2 Supplementary Time-series Comparison Plots

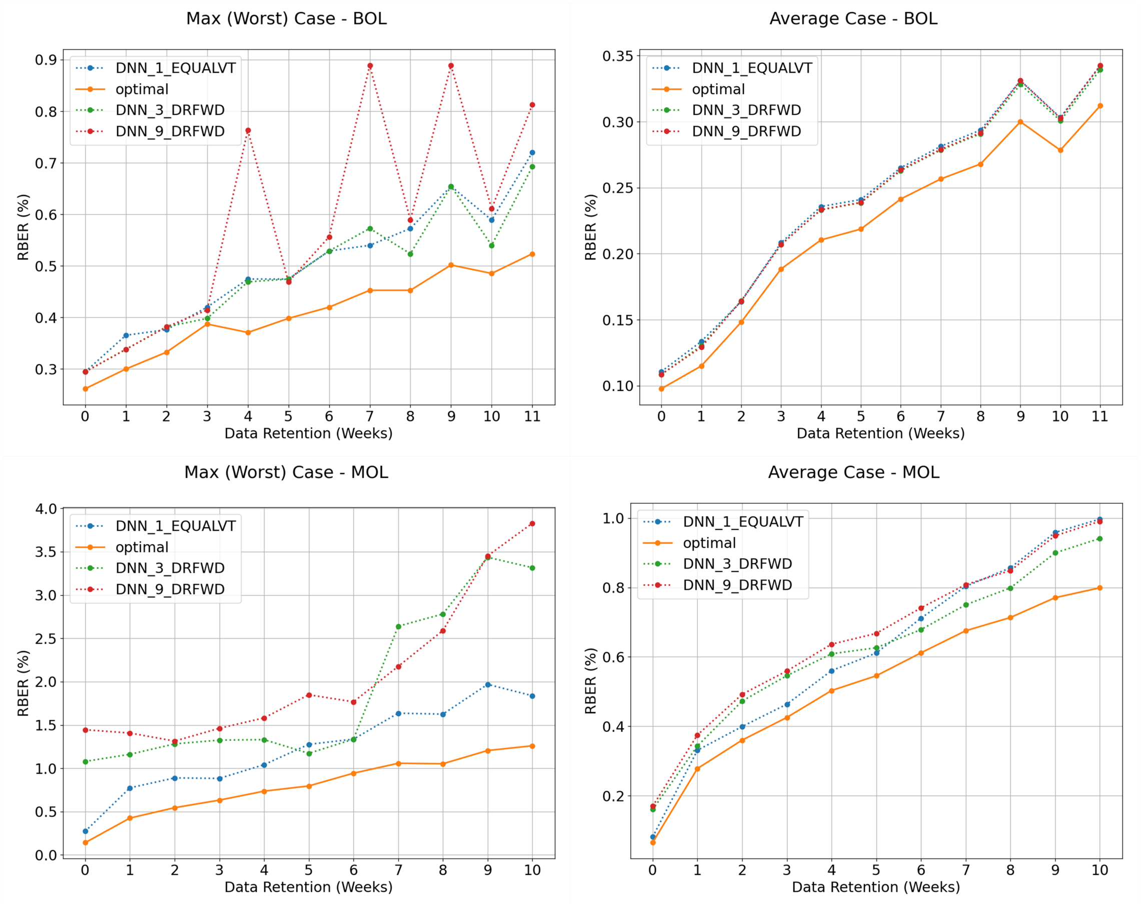


Figure A.1: Different DNN time-series variation comparisons on RBER % vs DR for both BOL and MOL datasets.

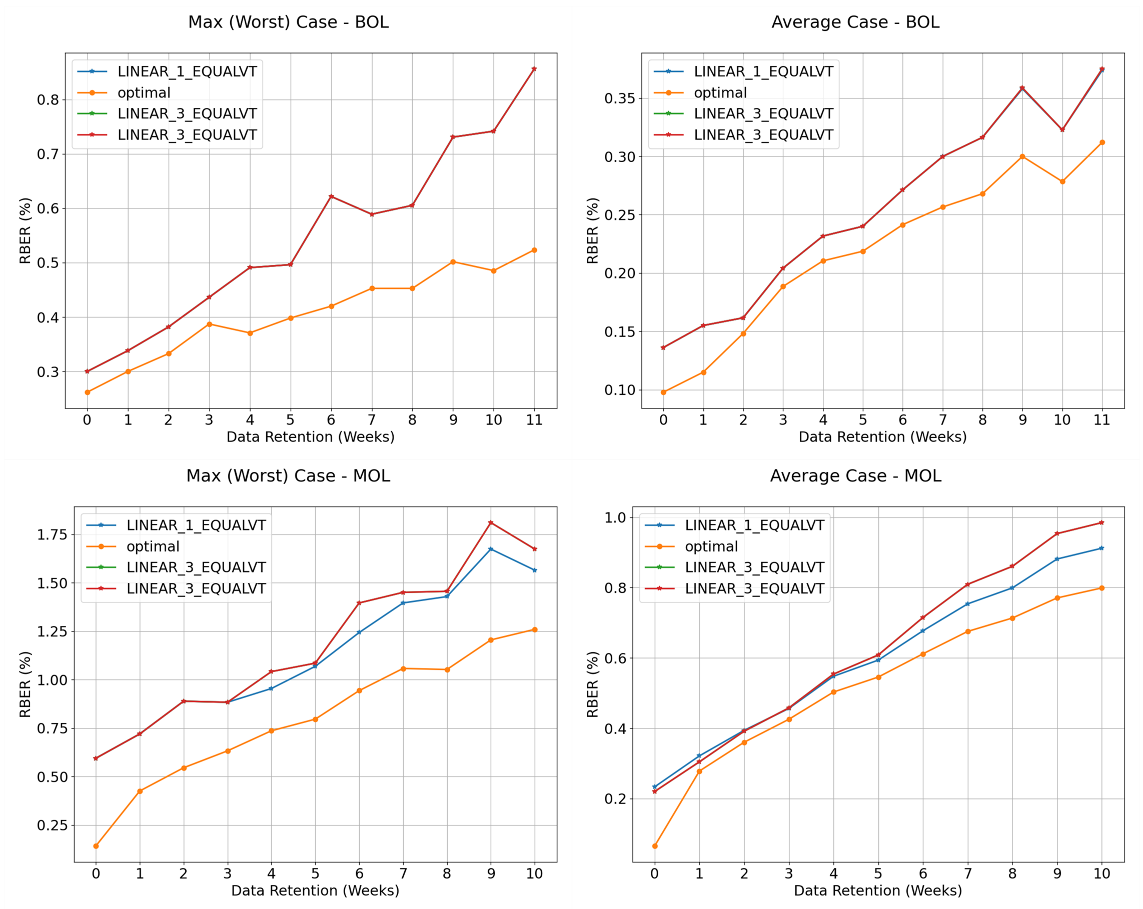


Figure A.2: LINEAR time-series comparisons on RBER % vs DR for both BOL and MOL datasets.

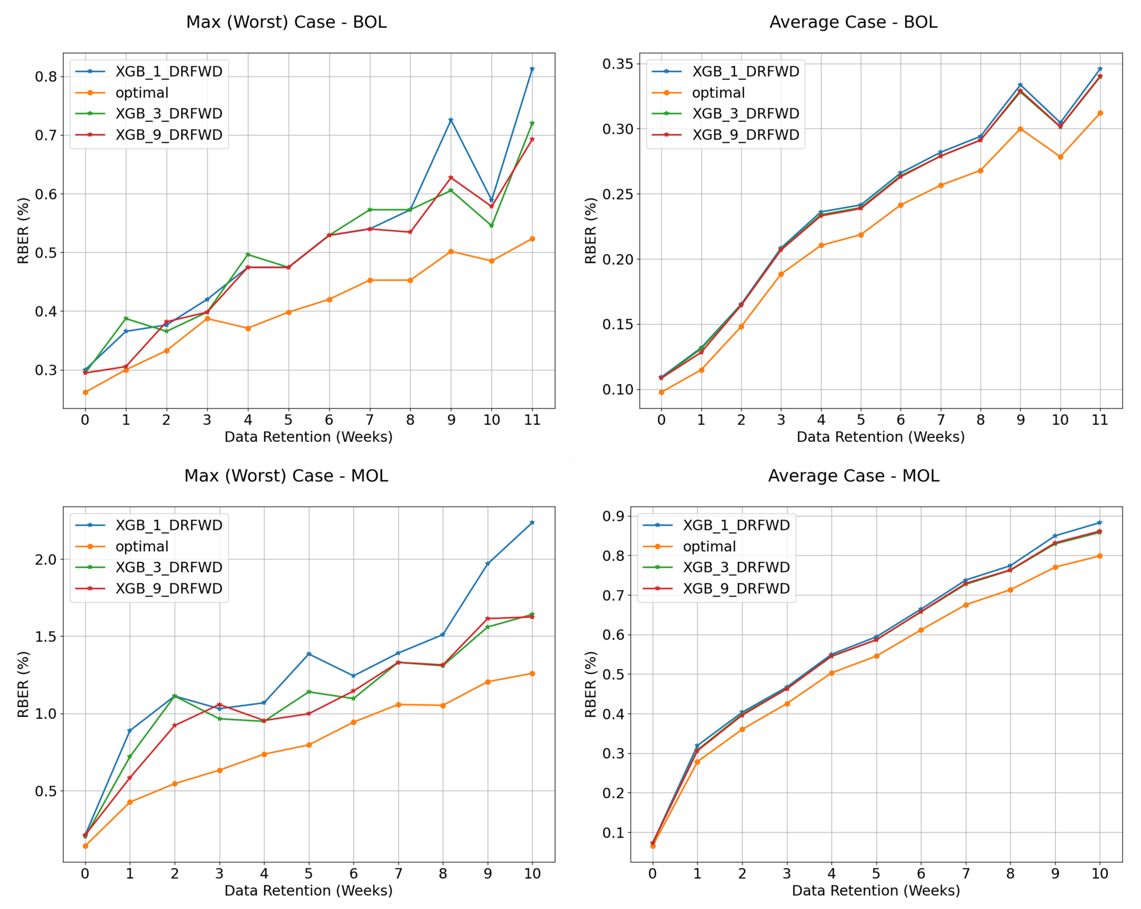


Figure A.3: XGBoost time-series lengths comparisons on RBER % vs DR for both BOL and MOL datasets.

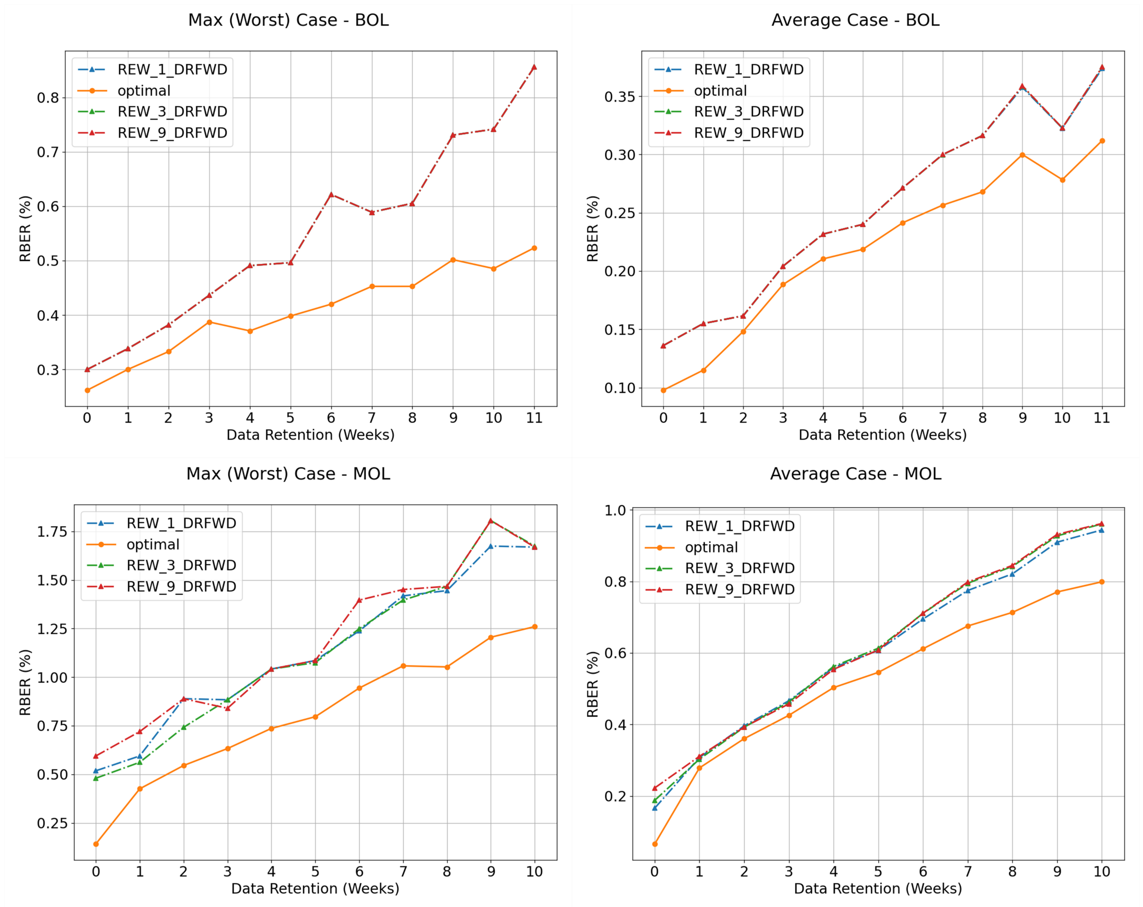


Figure A.4: Different REW time-series variation comparisons on RBER % vs DR for both BOL and MOL datasets.

Appendix B

Object Tracking IoU for individual videos of OTB-100

Video File Name	**MOSSE**	**KCF**	**YOLO-Lite**	**YOLOv3**	**GOTURN**	**WiSARD**
Basketball	0.0000	0.0037	0.2219	0.4074	0.3516	0.1705
Biker	0.0000	0.1080	0.0383	0.0421	0.6772	0.3379
Bird1	0.0000	0.0031	0.0081	0.0517	0.0557	0.0443
Bird2	0.6878	0.0394	0.0031	0.1004	0.4741	0.3373
BlurBody	0.0000	0.0678	0.7434	0.8234	0.3126	0.4842
BlurCar1	0.1078	0.2455	0.5126	0.7249	0.0882	0.1037
BlurCar2	0.6256	0.2405	0.7525	0.8003	0.5117	0.0903
BlurCar3	0.1504	0.1846	0.4774	0.7897	0.0866	0.0713
BlurCar4	0.6652	0.7362	0.4550	0.6735	0.2039	0.5616
BlurFace	0.3126	0.5442	0.0652	0.1275	0.5703	0.1857
BlurOwl	0.0473	0.0887	0.0041	0.0089	0.1181	0.1366
Board	0.6678	0.6161	0.0001	0.0010	0.1060	0.1743
Bolt	0.0000	0.0043	0.0264	0.5655	0.0034	0.0132
Bolt2	0.0000	0.0052	0.1342	0.6108	0.3756	0.0080
Box	0.2520	0.0696	0.0011	0.0019	0.1239	0.6068
Boy	0.0000	0.1345	0.0731	0.0466	0.8045	0.6262
Car1	0.0830	0.0782	0.1333	0.6992	0.1569	0.7257
Car2	0.6588	0.6144	0.6944	0.7559	0.7598	0.8408
Car24	0.1138	0.4091	0.4195	0.8312	0.7831	0.6589
Car4	0.4686	0.2091	0.7240	0.8203	0.7857	0.8296
CarDark	0.6226	0.5400	0.0052	0.0812	0.0331	0.5365

Video File Name	**MOSSE**	**KCF**	**YOLO-Lite**	**YOLOv3**	**GOTURN**	**WiSARD**
CarScale	0.4080	0.3953	0.3696	0.8698	0.7258	0.3587
ClifBar	0.1172	0.0624	0.0260	0.1197	0.1791	0.5874
Coke	0.1550	0.0542	0.0000	0.0384	0.5014	0.1780
Couple	0.0000	0.0179	0.4489	0.6681	0.6013	0.0499
Coupon	0.8841	0.4614	0.0000	0.0758	0.7899	0.8629
Crossing	0.0000	0.0602	0.0253	0.7245	0.4860	0.6424
Crowds	0.0000	0.0076	0.0011	0.5773	0.4981	0.4114
Dancer	0.3982	0.1112	0.3818	0.4848	0.4825	0.7507
Dancer2	0.6988	0.2493	0.4862	0.6312	0.6434	0.7345
David	0.1026	0.0219	0.0533	0.0732	0.2303	0.1744
David2	0.0319	0.1648	0.0427	0.0493	0.4423	0.6589
David3	0.0000	0.0732	0.2168	0.6134	0.1985	0.0844
Deer	0.0776	0.0082	0.0350	0.0582	0.2179	0.0344
Diving	0.2131	0.1098	0.0763	0.4208	0.5057	0.1858
Dog	0.0119	0.0185	0.1963	0.2574	0.2666	0.4779
Dog1	0.4431	0.4235	0.2041	0.4095	0.3957	0.7738
Doll	0.0000	0.3067	0.0111	0.0615	0.1154	0.7439
DragonBaby	0.0400	0.0206	0.0855	0.1084	0.4578	0.1809
Dudek	0.7077	0.7364	0.1460	0.1779	0.8048	0.2661
FaceOcc1	0.0000	0.7586	0.0874	0.4264	0.5528	0.7565
FaceOcc2	0.6263	0.7272	0.0491	0.1696	0.6031	0.5535

Video File Name	**MOSSE**	**KCF**	**YOLO-Lite**	**YOLOv3**	**GOTURN**	**WiSARD**
Fish	0.7186	0.0544	0.2484	0.1198	0.7362	0.6923
FleetFace	0.4577	0.5760	0.1599	0.1922	0.7127	0.5292
Football	0.4329	0.2707	0.0339	0.0540	0.3587	0.5212
Football1	0.0000	0.0266	0.0106	0.0643	0.4873	0.4614
Freeman1	0.0000	0.0926	0.0717	0.0768	0.6677	0.3107
Freeman3	0.0000	0.0565	0.0744	0.0918	0.7699	0.0009
Freeman4	0.0000	0.0435	0.0308	0.0886	0.2063	0.1366
Girl	0.0212	0.0212	0.0855	0.2390	0.5086	0.2697
Girl2	0.3130	0.6674	0.3131	0.6815	0.1072	0.1178
Gym	0.0000	0.0121	0.2790	0.6519	0.4437	0.2652
Human2	0.0000	0.4411	0.5783	0.6972	0.7062	0.4520
Human3	0.1239	0.0026	0.2111	0.7604	0.0262	0.0058
Human4	0.1114	0.0565	0.0433	0.5848	0.0523	0.0959
Human5	0.0000	0.1058	0.1486	0.7995	0.6212	0.4174
Human6	0.0000	0.0063	0.2959	0.6643	0.6954	0.1883
Human7	0.3103	0.1702	0.3475	0.8425	0.4773	0.2236
Human8	0.4641	0.0215	0.2581	0.8553	0.2002	0.0951
Human9	0.0000	0.0166	0.2558	0.8289	0.5012	0.0738
Ironman	0.0000	0.0155	0.0307	0.0509	0.2039	0.0592
Jogging	0.0000	0.0002	0.0322	0.0367	0.0122	0.1261
Jump	0.0000	0.0747	0.1243	0.3910	0.1254	0.0559

Video File Name	MOSSE	KCF	YOLO-Lite	YOLOv3	GOTURN	WiSARD
Jumping	0.0474	0.0382	0.0609	0.0633	0.5235	0.1427
KiteSurf	0.0370	0.1491	0.0905	0.0790	0.3706	0.4545
Lemming	0.1253	0.0204	0.0063	0.0123	0.3620	0.3131
Liquor	0.3077	0.6085	0.4136	0.7496	0.3139	0.3114
Man	0.1434	0.1781	0.0737	0.0745	0.8030	0.7461
Matrix	0.0000	0.0052	0.0176	0.0374	0.0669	0.1346
Mhyang	0.7498	0.6930	0.0914	0.1333	0.6844	0.8329
MotorRolling	0.0000	0.0764	0.0482	0.3474	0.5296	0.0872
MountainBike	0.7633	0.0639	0.0172	0.5409	0.7076	0.2473
Panda	0.0018	0.0262	0.0033	0.2765	0.4680	0.4260
RedTeam	0.0667	0.4761	0.0010	0.0281	0.3004	0.4942
Rubik	0.2122	0.0966	0.0221	0.3202	0.2372	0.1958
Shaking	0.0000	0.0048	0.0439	0.0983	0.7889	0.1279
Singer1	0.0000	0.3503	0.1800	0.6312	0.5727	0.4683
Singer2	0.6609	0.0422	0.1563	0.3939	0.0418	0.5953
Skater	0.4669	0.0441	0.6152	0.6016	0.6933	0.4064
Skater2	0.0503	0.1782	0.5903	0.6469	0.6694	0.3916
Skating1	0.1608	0.0410	0.1644	0.7110	0.5442	0.2738
Skating2	0.0000	0.0003	0.0705	0.0705	0.0751	0.0644
Skiing	0.0000	0.0085	0.0043	0.1661	0.5413	0.0765
Soccer	0.0918	0.0484	0.0059	0.0278	0.1162	0.1291

Video File Name	**MOSSE**	**KCF**	**YOLO-Lite**	**YOLOv3**	**GOTURN**	**WiSARD**
Subway	0.0000	0.0489	0.0953	0.6163	0.0711	0.1518
Surfer	0.0000	0.0142	0.0823	0.0730	0.5146	0.2772
Suv	0.4820	0.3803	0.4664	0.5836	0.3190	0.3024
Sylvester	0.4055	0.3579	0.0034	0.1148	0.3214	0.4965
Tiger1	0.0716	0.0350	0.0062	0.0053	0.4779	0.5244
Tiger2	0.0880	0.0092	0.0003	0.0047	0.2675	0.2706
Toy	0.0862	0.0379	0.0131	0.1023	0.4740	0.4921
Trans	0.5447	0.5044	0.0337	0.2262	0.4764	0.3980
Trellis	0.5586	0.3120	0.1017	0.1248	0.7051	0.5751
Twinnings	0.0000	0.2101	0.0290	0.5320	0.1962	0.4912
Vase	0.0000	0.0978	0.0342	0.1082	0.5702	0.5653
Walking	0.0017	0.0236	0.1269	0.7262	0.6816	0.6760
Walking2	0.0000	0.2712	0.3695	0.6445	0.2825	0.3758
Woman	0.0000	0.0412	0.2835	0.5382	0.1013	0.1265

Table B.1: IoU per Object Tracker algorithm for every video present in OTB-100.

Appendix C

Fake News Transparency Dataset Relevance Scores

C.1 Politifact Top-100 Tokens

Politifact (SAFE-β Text-only)							
True Positive		True Negative		False Positive		False Negative	
Word	Score	Word	Score	Word	Score	Word	Score
Headline							
Republican	0.8727	forgot...	1.0000	presidential	0.8065	’Regular	0.8816
presidential	0.8369	’just	0.9721	Democrats	0.7853	’Never	0.8062
debate	0.8248	Wiping	0.9612	election	0.5924	Teen-Ager	0.7663
Economic	0.7888	Admits	0.8958	2020.0000	0.5718	Miami	0.7003
Congressional	0.7715	Attacked	0.8878	Senator	0.5391	Found	0.6937
Barack	0.7702	Fired	0.8761	in	0.4898	Shore	0.6645
President	0.7530	Accused	0.8717	Biden	0.4722	!	0.6606
Outlook	0.7430	Arrested	0.8654	government	0.4619	Of	0.6349
Transcript	0.7330	president... They	0.8624	States	0.4494	Jersey	0.6288
Sen.	0.7269	Her	0.8467	of	0.4482	Mercurial	0.6178
transcript	0.7205	Microchipping	0.8217	Statement	0.4384	CITY	0.6155
Appropriations	0.7190	Granddaughter	0.8199	proposed	0.4365	Confident	0.6009
Amendment	0.6746	Falsely	0.8197	General	0.4322	Hedge-Fund	0.5994
Congress	0.6702	Meth	0.8193	Voters	0.4310	In	0.5945
policymakers	0.6551	To	0.7954	Sessions	0.4168	Sign	0.5929
Obama	0.6475	Raped	0.7911	:	0.4096	Journey	0.5924
Office	0.6429	Accuser	0.7893	the	0.4086	FOR	0.5807
’This	0.6138	Being	0.7871	law	0.4078	Guy	0.5800
Democrats	0.6044	Specially-Colored	0.7839	raises	0.4074	HALL	0.5780

Energy	0.6006	Wedding	0.7825	Attorney	0.3994	Said	0.5769
Fiscal	0.6000	Of	0.7712	United	0.3987	A	0.5754
amending	0.5946	Lying	0.7650	on	0.3924	His	0.5710
political	0.5885	Stranded	0.7647	numbers	0.3893	Took	0.5692
Policy	0.5881	RECOUNT	0.7637	possibility	0.3891	Dinosaur	0.5643
Mitt	0.5854	Crashed	0.7587	</pad>	0.3882	With	0.5368
Act	0.5819	Gifts	0.7583	Information	0.3882	Bad	0.5318
McCain	0.5815	anti-far-right	0.7523	The	0.3871	Nexis®	0.5305
,	0.5794		0.7468	Occupy	0.3837	Mayor	0.5296
:	0.5683	NFL	0.7453	for	0.3829	Bay	0.5279
Romney	0.5664	Destroyed	0.7417	URGENT	0.3813	Recall	0.5260
Statistics	0.5614	Clothes	0.7371	do	0.3807	Hunter	0.5191
constitution	0.5611	Suspended	0.7363	-	0.3797	Missing	0.5087
Santorum	0.5581	Was	0.7362	would	0.3758	Magazine	0.5084
Skype	0.5545	Permanently	0.7359	bid	0.3754	mtvU	0.5060
voters	0.5519	MS-13	0.7344	shutdown	0.3747	Match	0.4994
GOP	0.5514	SLAMS	0.7324	n't	0.3729	He	0.4993
'	0.5428	Sale	0.7319	...	0.3721	Does	0.4980
U.S.	0.5417	Seized	0.7302	North	0.3701	n't	0.4882
McSame	0.5361	Penalty	0.7290	lie	0.3686	detainees	0.4760
Rep.	0.5281	SHOCKING	0.7286	with	0.3545	RACE	0.4737
Budget	0.5252	Sheriff	0.7257	by	0.3544	Encounters	0.4733

Kaine	0.5191	Expensive	0.7228	key	0.3541	The	0.4683
and	0.5139	Killing	0.7225	terrorism	0.3476	vaccinations	0.4635
Speech	0.5069	Gang	0.7218	Facebook	0.3455	’Editorial	0.4618
Security	0.4917	Thugs	0.7217	affected	0.3411	At	0.4558
Politico	0.4890	Were	0.7210	is	0.3387	Product	0.4555
(	0.4862	MURDERED	0.7197	goal	0.3376	AUTOS	0.4531
ballots	0.4844	Got	0.7167	Carolina	0.3358	’s	0.4526
Clinton	0.4828	A	0.7134	?	0.3306	Company	0.4523
says	0.4816	Handed	0.7125	Who	0.3304	’	0.4511
Medicaid	0.4810	Be	0.7120	Nation	0.3303	Says	0.4509
Geithner	0.4759	TERRORISTS	0.7110	charge	0.3288	in	0.4493
Transcripts	0.4732	For	0.7082	DEA	0.3227	Peanut	0.4441
as	0.4719	LIFE-SHATTERING	0.7049	protesters	0.3216	Guantanamo	0.4400
Secretary	0.4717	Booting	0.7025	Just	0.3117	New	0.4361
Week	0.4716	His	0.7020	ICE	0.3036	H1N1	0.4353
News	0.4715	Guns	0.6999	removal	0.2942	tonight	0.4342
HHS	0.4694	Invented	0.6996	112-Year-Old	0.2899	Trump	0.4322
109th	0.4640	Discontinue	0.6982	A	0.2684	doing	0.4304
1999-2000	0.4614	..	0.6912	dailynative.us	0.2327	Word	0.4301
2007.0000	0.4613	Never	0.6902	Recusal	0.2016	Plan	0.4287
International	0.4583	COMEY	0.6899	Raided	0.1883	be	0.4278
Bush	0.4554	Robocall	0.6859			Tough	0.4243

Boehner	0.4541	’Trump	0.6842			York	0.4205
TV	0.4526	Victims	0.6841			to	0.4182
;	0.4520	Murder	0.6782			Taxes	0.4157
Bills	0.4509	murder	0.6777			Medium	0.4145
John	0.4482	Ingestion	0.6729			About	0.4140
Obamacare	0.4481	Toys	0.6715			Carly	0.4139
news	0.4481	Burn	0.6678			is	0.4121
Education	0.4476	Abortions	0.6663			–	0.4121
2006.0000	0.4466	Irate	0.6655			:	0.4094
Priebus	0.4447	IMMEDIATELY	0.6650			Fred	0.4030
Political	0.4439	Illegal	0.6638			First	0.4026
Action	0.4437	Knee	0.6636			What	0.4005
Rand	0.4372	Warn	0.6631			offered	0.4004
Rubio	0.4365	Tampered	0.6623			VIDEO	0.3908
oil	0.4346	. . . Using	0.6621			Preview	0.3902
Department	0.4345	CONTROL	0.6602			</pad>	0.3884
Challenges	0.4342	Car	0.6580			Bill	0.3810
Care	0.4341	Streep	0.6573			Talk	0.3808
second	0.4340	SHOOT	0.6559			Fiorina	0.3762
daily	0.4336	Saw	0.6542			Clinton	0.3759
universal	0.4332	Death	0.6515			Board	0.3647
budget	0.4321	Mommy	0.6486			on	0.3568

to	0.4307	Doors	0.6474			Thompson	0.3520
voter	0.4304	Is	0.6470			Approval	0.3440
Ad	0.4298	Admission	0.6430			Special	0.3402
Hillary	0.4273	Royal	0.6421			;	0.3083
tax	0.4271	Get	0.6416			–	0.2952
Organizing	0.4265	Wear	0.6413			,	0.2819
Op-Ed	0.4238	Gay	0.6401			Candidate	0.2326
Repeal	0.4238	POWERFUL	0.6397			Senator	0.1755
Bing	0.4221	After	0.6387			Obama	0.1642
Bob	0.4219	Meant	0.6385			Republican	0.0695
Statement	0.4208	DiCaprio	0.6377				
Browse	0.4201	MILLION	0.6375				
Dianne	0.4195	Absorbing	0.6360				
Rick	0.4192	Activist	0.6333				
by	0.4192	Secretly	0.6329				

Table C.1: Politifact Headline top-100 token scores for different predictions.

Politifact (SAFE-β Text-only)

True Positive		True Negative		False Positive		False Negative	
Word	Score	Word	Score	Word	Score	Word	Score
Body-Text							
678-8511	0.6596	Trendolizer™	0.4193	Senate	0.5218	tire-melting	0.4010
Rating	0.6276	al-Islam	0.4074	Act	0.4985	RadarOnline.com	0.4000
Outlook	0.6116	implanted	0.4046	//actionnetwork.org/	0.4965	suspected	0.3966
Remove	0.6062	Accuser	0.4040	Judiciary	0.4910	southern-fried	0.3962
cqrollcall.com	0.5950	Trump	0.4027	Committee	0.4906	murder	0.3960
Fiscal	0.5865	Hayhoe	0.4024	intervening.	0.4813	Trump	0.3956
Economic	0.5762	diagnosed	0.4010	NVRA	0.4773	teen-ager	0.3950
Watch	0.5744	surgically	0.4006	source=	0.4697	Maj.	0.3945
browser	0.5633	blood	0.4006	nowrapper=true	0.4671	flock	0.3934
transcript	0.5343	Etemad	0.4003	Registration	0.4658	’	0.3927
version	0.5341	fake	0.4003	referrer=	0.4593	Haynie	0.3927
RADDATZ	0.5330	Mickens	0.3999	(	0.4592	police	0.3927
:	0.5288	Allred	0.3995	:	0.4542	strange	0.3924
1987-	0.5236	flew	0.3992	https	0.4538	treated	0.3923
Please	0.5222	sorties	0.3992	Biden	0.4508	”	0.3918
@	0.5189	murdered	0.3988	Voter	0.4475	his	0.3916
Parnass/The	0.5170	Stoneman	0.3986	Democratic	0.4443	who	0.3915
Sen.	0.5137	49-year-old	0.3985	javascript	0.4416	Diana	0.3915
Queue	0.5132	evacuated	0.3984	presidential	0.4379	Sasha	0.3914

MSN	0.5060	corgis	0.3984	Immigration	0.4366	eats	0.3913
AMANPOUR	0.5053	brutally	0.3982	2020.0000	0.4339	terrorists	0.3911
Disconnect	0.5034	32-year-old	0.3981	voters	0.4316	with	0.3910
Senate	0.5027	royal	0.3974	President	0.4304	her	0.3907
)	0.5014	reportedly	0.3974	log	0.4298	found	0.3906
Act	0.5011	actor	0.3971	Sessions	0.4280	disdainful	0.3906
hotline	0.5007	accuser	0.3970	)	0.4270	s	0.3904
interactive	0.4962	woman	0.3968	U.S.	0.4256	Carly	0.3903
Financial	0.4941	teens	0.3968	?	0.4250	family	0.3902
congressional	0.4925	14-year-old	0.3968	agencies	0.4248	laying	0.3902
ballot—so	0.4901	FBI	0.3965	government	0.4244	whom	0.3901
Action	0.4840	grabbed	0.3964	&	0.4235	Gibbs	0.3901
Committee	0.4839	Beverly	0.3961	federal	0.4221	deserved	0.3901
.	0.4818	Keanu	0.3960	immigration	0.4218	Ridgefield	0.3900
Viewer	0.4816	’	0.3959	Elections	0.4211	man	0.3900
Javascript	0.4806	nabbed	0.3959	Enforcement	0.4206	is	0.3900
STEPHANOPOULOS	0.4804	informant	0.3959	National	0.4203	charm	0.3900
Geithner	0.4792	González	0.3957	Board	0.4197	a	0.3899
SCHIEFFER	0.4791	U-boat	0.3957	must	0.4193	reunited	0.3898
Budget	0.4791	wrath	0.3956	]	0.4188	swarmed	0.3898
R-Tenn.	0.4776	allegedly	0.3956	employees	0.4184	it	0.3898
all	0.4775	O	0.3954	nomination	0.4180	PHOTOS	0.3897

McCain	0.4762	relentless	0.3954	bid	0.4177	holding	0.3897
Republican	0.4746	semiautomatic	0.3953	liberal	0.4177	the	0.3897
800.0000	0.4745	s	0.3952	.	0.4176	“	0.3897
externalActionCode	0.4719	owned	0.3952	continue	0.4175	according	0.3897
OBAMA	0.4718	Wilkinson	0.3951	at	0.4175	she	0.3897
Barack	0.4715	her	0.3951	Customs	0.4175	he	0.3896
older	0.4713	covering-up	0.3950	political	0.4170	has	0.3896
Obama	0.4674	Antifa	0.3949	[	0.4169	having	0.3895
PRESIDENT	0.4661	Katharine	0.3947	House	0.4169	hugs	0.3895
Contact	0.4658	Gloria	0.3946	available	0.4164	lunch.	0.3895
Document	0.4657	assassinations	0.3946	2013.0000	0.4162	Connecticut	0.3894
Center	0.4656	implants	0.3945	General	0.4157	away	0.3893
Secretary	0.4652	minding	0.3944	2010.0000	0.4157	after	0.3893
Congress	0.4647	wheeled	0.3944	Attorney	0.4156	carried	0.3892
VIDEO	0.4643	DiCaprio	0.3944	Monday	0.4153	food-safety	0.3892
trillion	0.4639	actress	0.3944	ballots	0.4151	that	0.3891
Years	0.4635	entrepreneur	0.3944	You	0.4145	in	0.3891
Memos	0.4633	pose	0.3943	,	0.4142	answers	0.3891
billion	0.4630	Parkland	0.3943	Democrats	0.4141	afternoon	0.3890
R-Fla.	0.4626	fire	0.3942	on	0.4140	of	0.3890
202-225-3001	0.4618	spotted	0.3940	Key	0.4137	both	0.3889
try	0.4615	guard	0.3939	funds	0.4136	Fred	0.3889

President	0.4615	officer	0.3939	carryover	0.4136	clear	0.3889
BARACK	0.4611	founded	0.3939	i	0.4134	A	0.3889
or	0.4603	Wiping	0.3939	commentators	0.4131	and	0.3889
amending	0.4599	Irma	0.3939	services	0.4127	people	0.3888
2009.0000	0.4589	his	0.3939	policies	0.4126	as	0.3888
Taxation	0.4584	falsely	0.3938	shutdown	0.4124	She	0.3888
loaded	0.4576	battled	0.3937	Vice	0.4124	whether	0.3888
(	0.4564	investigated	0.3937	f	0.4114	took	0.3887
supported	0.4562	girls	0.3937	in	0.4111	Sadler	0.3887
Democrat	0.4549	t	0.3937	Jeff	0.4111	verge	0.3887
KEETER	0.4547	was	0.3936	enforces	0.4111	Thompson	0.3887
use	0.4547	Mojtaba	0.3936	the	0.4108	Remember	0.3887
FL-21	0.4543	injury	0.3936	campaign	0.4107	to	0.3887
before—blanket	0.4543	Winslow	0.3935	issues	0.4106	when	0.3887
Department	0.4540	had	0.3935	workers	0.4103	battling	0.3887
government	0.4539	sprees	0.3934	State	0.4101	just	0.3887
actionDate	0.4530	hasn	0.3932	use	0.4100	interest	0.3886
Pay	0.4522	billionaire	0.3932	Department	0.4096	visit	0.3886
Senator	0.4517	arson	0.3931	laws	0.4095	That	0.3885
BEGIN	0.4512	discredited	0.3931	illegal	0.4095	actual	0.3885
experience	0.4512	brother	0.3931	version	0.4094	said	0.3884
displayText	0.4509	younger	0.3931	Capitol	0.4094	whose	0.3883

TV	0.4509	Geraldo	0.3930	D-NJ	0.4091	was	0.3883
and	0.4503	Reeves	0.3930	White	0.4090	23.8000	0.3883
income	0.4503	eyes	0.3930	uncertainty	0.4088	one	0.3883
Reuters	0.4496	41-year-old	0.3929	Uncertainty	0.4086	their	0.3882
Our	0.4496	Douglas	0.3929	to	0.4076	opponent	0.3882
tax	0.4496	isn	0.3929	agency	0.4075	Bay	0.3882
Republicans	0.4495	died	0.3929	issued	0.4074	headed	0.3882
U.S.	0.4494	chased	0.3928	key	0.4070	ones	0.3881
CQ	0.4492	became	0.3928	downsize	0.4067	</pad>	0.3881
legislation	0.4491	a	0.3928	contractors	0.4062	they	0.3881
Tax	0.4487	meth	0.3928	is	0.4058	antics	0.3881
Care	0.4485	arrived	0.3927	abolishing	0.4056	.	0.3881
Election	0.4478	little	0.3927	”	0.4055	But	0.3881
percent	0.4475	deer	0.3927	ethics	0.4055	Donald	0.3881
CLIP	0.4474	rammed	0.3927	liberals	0.4054	gave	0.3881

Table C.2: Politifact Body-text top-100 token scores for different predictions.

C.2 Gossipcop Top-100 Tokens

Gossipcop (SAFE-β Text-only)							
True Positive		True Negative		False Positive		False Negative	
Word	Score	Word	Score	Word	Score	Word	Score
Headline							
18-49	0.9414	Celebrities	0.9799	chalamet	1.0000	Pattinson	0.7435
’guilty	0.8613	Pattinson	0.8727	timothee	0.8508	Aniston	0.6283
Soundsational	0.8580	react	0.8283	’One	0.8191	’She	0.5277
Kufrin	0.8557	Aniston	0.7892	Tweeden	0.6970	Jennifer	0.5267
’Hollywood	0.8434	Kidman	0.7221	’callous	0.6868	Stefani	0.5172
’Reputation	0.8195	Stefani	0.7210	Jam-Packed	0.6711	Pitt	0.5165
Ankle-Wrap	0.8194	’World	0.6939	Makeup	0.6658	’So	0.5073
Boob-Baring	0.8130	’has	0.6894	Season	0.6473	Lopez	0.5064
Zendaya	0.8094	de	0.6892		0.5980	Divorce	0.5039
USweekly	0.7887	&	0.6288	Post-Pregnancy	0.5902	Beckham	0.5011
’One	0.7829	Pitt–Angelina	0.6141	Garbine	0.5818	Bieber	0.4976
Alone-Inspired	0.7723	Kanye	0.6018	Snapchat	0.5805	Selena	0.4972
Sanderson-Approved	0.7668	React	0.6000	Collection	0.5746	Rottman	0.4931
Abloh	0.7663	Jolie	0.5943	Hook-Ups	0.5744	Zellweger	0.4843
High-Waisted	0.7657	torment	0.5790	Momoa	0.5724	Kardashian	0.4760
Kellywise	0.7623	’ordered	0.5758	ICEL	0.5648	Beyonce	0.4758
Migos	0.7594	EXCLUSIVE	0.5680	’Star	0.5587	DiCaprio	0.4708
Wake-And-Baker	0.7580	Report	0.5671	’One-Up	0.5551	Trump	0.4689
Dietland	0.7552	Beckham	0.5632	Over-Sized	0.5547	Talks	0.4655

Camera-Ready	0.7547	?	0.5632	Adorns	0.5475	Jolie	0.4651
Nagasu	0.7536	JENNER	0.5598	’Hey	0.5474	Reportedly	0.4639
Off-the-Shoulder	0.7515	Clooney	0.5591	’Do	0.5472	rumors	0.4631
chalamet	0.7451	’secret	0.5540	’wasting	0.5468	Report	0.4616
’Soul	0.7450	Pitt	0.5510	Pre-Camera	0.5433	vote	0.4462
OUT100	0.7390	Bieber	0.5473	Universidad	0.5378	presidential	0.4437
Hovino	0.7312	celebs	0.5449	Awards	0.5366	Kidman	0.4436
Losique	0.7303	Jennifer	0.5414		0.5359	Cruise	0.4403
M-Inspired	0.7248	DiCaprio	0.5400	TV	0.5319	Falchuk	0.4363
’Drop	0.7217	Theroux	0.5397	Furniture	0.5300	kiss	0.4344
DuShon	0.7215	Weekly	0.5369	1998–2006	0.5295	:	0.4287
∼Cool∼	0.7182	ultimatum	0.5357	Shower	0.5250	?	0.4260
Outlives	0.7181	Affleck	0.5318	Negga	0.5198	Aronofsky	0.4254
Celebritymaximum.com	0.7178	Bendjima	0.5305	’used	0.5133	Devastated	0.4247
’Ultra-Romantic	0.7176	Us	0.5284	IGotBusted	0.5118	reform	0.4242
’Property	0.7152	Marriage	0.5282	14-Year-Old	0.5112	Swift	0.4236
’Cherry	0.7130	Pregnant	0.5269	’leaning	0.5075	tease	0.4226
Lesseps	0.7101	Blake	0.5212	Bastón	0.5070	Clooney	0.4184
BB-8	0.7093	Jenner	0.5186	London	0.5049	Brad	0.4173
Trip’—Here	0.7091	Divorce	0.5170	tickets	0.5041	Rumors	0.4172
Haddish	0.7058	Dating	0.5145	Muguruza	0.5021	&	0.4165
Pre-Grammy	0.7037	’Messin	0.5137	Jacket	0.5012	Biel	0.4127

Chalamet	0.6953	Kristen	0.5136	Premiere	0.5007	Kanye	0.4106
Floribama	0.6952	Gaga	0.5121	'No	0.5005	Shelton	0.4094
'Raven	0.6947	Shelton	0.5092	clinical	0.4980	Gomez	0.4084
Shower	0.6916	To	0.5082	Dion-na	0.4973	Claims	0.4074
Vanderpump	0.6911	Claims	0.5073	Picture-winning	0.4958	Paltrow	0.4074
Costumes	0.6843	Brad	0.5072	'altered	0.4958	split	0.4055
Shoes	0.6841	Selena	0.5022	Caribbean	0.4950	Robert	0.4047
'The	0.6832	'missed	0.5012	Tour	0.4935	Victoria	0.4041
'Easy	0.6832	(	0.4992	Dresses	0.4929	Split	0.4022
'Ellen	0.6800	:	0.4936	Atlanta	0.4912	Himself	0.4010
'not	0.6778	Breakups	0.4911	14.0000	0.4902	Leaning	0.4002
£335	0.6777	Rossi	0.4893	Cannes	0.4900	Pregnancy	0.4000
Snapchat	0.6743	Rumors	0.4873	International	0.4888	Blake	0.4000
Bachelorette	0.6739	Kardashian	0.4860	Bags	0.4884	Theroux	0.3984
Monday—the	0.6698	Hemsworth	0.4837	Series	0.4838	Gwyneth	0.3979
'better	0.6694	Justin	0.4825	Thrones	0.4838	Dating	0.3970
Overalls	0.6685	Robert	0.4823	Selfie	0.4835	A	0.3967
Makeup	0.6673	warns	0.4814	ldol	0.4806	Kristmas	0.3956
TV	0.6667	rumors	0.4782	'	0.4796	Timberlake	0.3939
Richards-Ross	0.6649	Nicole	0.4776	Birthday	0.4764	Ryan	0.3926
'No	0.6636	Relationship	0.4751	Leakes	0.4763	Markle	0.3920
Benchetrit	0.6557	cuddling	0.4721	CEO	0.4759	Alex	0.3910

’Episode	0.6549	Debunked	0.4712	world-class	0.4757	parliament	0.3907
’Girls	0.6549	Cruise	0.4698	prima	0.4744	Hillary	0.3853
’Sign	0.6545	Madonna	0.4684	robbery	0.4731	Calls	0.3847
’Guardians	0.6506	Foxx	0.4666	Star	0.4720	Jessica	0.3844
Jacket	0.6503	Jay-Z	0.4659	carpet	0.4717	Feels	0.3837
’Total	0.6479	’The	0.4659	headlinebooth.com	0.4684	Celebrities	0.3837
Awards	0.6463	[	0.4650		0.4682	Gwen	0.3824
’Jersey	0.6446	Gwen	0.4648	Netanyahu-	0.4675	Leonardo	0.3824
Raísa	0.6445	FKA	0.4647	’disgusting	0.4656	Is	0.3817
Demi-Leigh	0.6445	Lopez	0.4634	’cosy	0.4654	[	0.3814
Budget-Friendly	0.6443	Reportedly	0.4626	Medium	0.4651	Donald	0.3813
Swimwear	0.6421	Islam	0.4622	’distancing	0.4645	West	0.3795
Sweater	0.6389	Begging	0.4622	Kits	0.4633	romance	0.3783
’Bachelor	0.6388	REPORT	0.4617	J.Lo-Affleck	0.4631	anymore	0.3769
’Houston	0.6387	Gossip	0.4578	Bestiality	0.4631	supporting	0.3768
BROWN. . . CHRIS	0.6385	Planning	0.4565	residency	0.4630	Jenner	0.3763
Quavo	0.6381	Forced	0.4564	’DWTS	0.4627	bullied	0.3738
Wig-Free	0.6376	Seacrest	0.4562	On-Set	0.4609	Meghan	0.3734
’Gorgeous	0.6373	Plans	0.4537	DACA	0.4589	Hanks	0.3724
Tomeeka	0.6370	Feuding	0.4512	Scarf	0.4583	Joins	0.3713
’Aquaman	0.6359	West	0.4506	Miami	0.4581	Angelina	0.3713
Vinessa	0.6332	’Has	0.4504	Mentalist	0.4560	Garner	0.3711

Skirt	0.6319	’is	0.4503	Halloween	0.4530	Justin	0.3698
Baby2Baby	0.6318	Dump	0.4469	embroidered	0.4523	kissing	0.3686
Nel-Peters	0.6291	Liam	0.4462	Maker	0.4510	Gosling	0.3685
With…	0.6290	Reports	0.4448	Figure	0.4499	2020.0000	0.3684
’Carpool	0.6259	Claim	0.4446	HIV-positive	0.4489	Married	0.3681
dog-smuggling	0.6246	Ex-BF	0.4439	Teefey	0.4486	Hadid	0.3679
’Roseanne	0.6245	’Silence	0.4436	Rearranges	0.4481	Jay-Z	0.3671
’Their	0.6210	tabloid	0.4432	Kim-Kanye	0.4477	Lambert	0.3668
13-episode	0.6210	Angelina	0.4397	MVP	0.4474	Kristen	0.3665
’Wrecking	0.6209	Katie	0.4388	Wars	0.4463	Winslet	0.3664
Ratajkowski	0.6203	Agree	0.4380	Hawaiian	0.4446	Him	0.3659
’50s-Themed	0.6183	shock	0.4361	Sleepovers	0.4444	Listening	0.3648
’There	0.6181	Romance	0.4361	Plus-One	0.4430	Clinton	0.3647
Fitz/Olivia	0.6169	]	0.4357	truck	0.4429	and	0.3641
uniforms	0.6166	J.Lo	0.4347	Reconnected	0.4414	Toll	0.3617

Table C.3: Gossipcop Headline top-100 token scores for different predictions.

Gossipcop (SAFE-β Text-only)							
True Positive		True Negative		False Positive		False Negative	
Word	Score	Word	Score	Word	Score	Word	Score
Body-Text							
notifications	0.9289	HollywoodLife.com	0.6963	low-vamp	0.5418	RadarOnline.com	0.4421
White/The	0.6092	RadarOnline.com	0.5772	Angelillo/UPI	0.5236	well—Us	0.4251
push	0.5654	HollywoodLife	0.4731	kimkardashian	0.5125	insider	0.3734
Aquazzura	0.5623	RadarOnline	0.4590	Vernoff	0.5120	source	0.3713
track-inspired	0.5616	Gossip	0.4583	Bendjima	0.5102	Rumors	0.3658
E	0.5493	Inc.	0.4560	_Andrea_Ant	0.4913	Aniston	0.3654
10-months-of	0.5452	resolved.	0.4505	him—at	0.4754	rumors	0.3635
Rodriguez/Getty	0.5439	insider	0.4268	Mazur/Getty	0.4745	Broadimage/	0.3582
Nadinne	0.5419	EXCLUSIVELY	0.4157	Kardashian–Jenner	0.4589	Theyre	0.3554
Scoopnest	0.5392	EXCLUSIVE	0.4155	background.2nd	0.4577	Radar	0.3542
once-golden	0.5366	webloid	0.4122	FilmMagic/Getty	0.4572	TMZ	0.3514
news	0.5341	DE	0.4095	though—the	0.4565	health—and	0.3510
emrata	0.5325	Radar	0.4084	jdelreal	0.4556	allegations	0.3500
Segretain/Getty	0.5323	Cop	0.4062	Teefey	0.4536	Weekly	0.3470
pic.twitter.com/higzC5aFZM	0.5234	tabloid	0.4031	susana_vp	0.4401	celebrity	0.3443
Foreign/Drama	0.5231	Enquirer	0.3985	'AGT'	0.4379	'care	0.3440
theindustryonblast	0.5211	Shookus	0.3978	E	0.4378	Disick	0.3431
Beador	0.5153	baby—even	0.3956	geofilter	0.4355	dinnerthey	0.3403
the. . .	0.5132	Kardashian	0.3951	justintimberlake	0.4297	gossip	0.3401
Shirtdresses	0.5127	AU	0.3947	Koshkina	0.4283	PHOTOS	0.3386

Broadway	0.5120	source	0.3847	$9\frac{1}{2}$	0.4249	tells	0.3384
it–Jora	0.5117	IMDb.com	0.3815	Bitmoji	0.4245	wrong/But	0.3372
glitter-coated	0.5113	//t.co/T6UYZhYU7q	0.3805	StillTheKing	0.4208	baggage—carried	0.3325
//t.co/oB5IEZmOf3	0.5105	told	0.3793	Live	0.4183	dating	0.3271
Ruymen/UPI	0.5094	off-again	0.3781	joh-LEE	0.4175	Patrick/	0.3270
Un-Researched	0.5061	Kardashian-Jenner	0.3766	anti-pimple	0.4160	PEOPLE	0.3238
Angelillo/UPI	0.5048	Rumors	0.3759	OKMagazine	0.4147	'The	0.3232
Jersey-style	0.5027	Aniston	0.3757	live-audience	0.4144	Celebrity	0.3231
Chernuchin	0.5012	PHOTOS	0.3739	'Brad	0.4140	Kardashian	0.3229
Get	0.4980	National	0.3729	@	0.4127	Gossip	0.3225
'Dance	0.4964	Hemsworth	0.3719	KUWTK	0.4124	C-plot	0.3222
Bendjima	0.4964	Nashawaty	0.3698	Básquet	0.4108	column	0.3216
him—at	0.4959	Disick	0.3679	star-turn	0.4078	Brad	0.3214
Harigtay	0.4957	Jolie	0.3679	Hall-Of-Famer	0.4069	Jennifer	0.3202
chrissyteigen	0.4937	Khloe	0.3675	8:45PM	0.4064	Pattinson	0.3188
	0.4929	pic.twitter.com/0PkvL8OmCM	0.3658	KateBennett_DC	0.4050	!	0.3187
O'Brientakes	0.4902	'Gossip	0.3642	fascinaton	0.4050	Touch	0.3181
Szagola	0.4892	wedding	0.3641	Turner/Jonas	0.4048	53-show	0.3174
that…	0.4872	Brad	0.3635		0.4027	fame—and	0.3171
Instagram.Their	0.4869	magazine	0.3628	Elsa/UPI	0.4022	Cheban	0.3170
Prisoner-Chic	0.4860	'Jennifer	0.3625	twerked	0.4021	an	0.3168
2018-19	0.4856	Rossdale	0.3621	fiberbit.net	0.4018	readers	0.3168
for…	0.4823	tells	0.3611	sarahaines	0.4016	Jolie	0.3164
pic.twitter.com/lhGRsJGBW5	0.4820	rumor	0.3610	a-swirling	0.4013	Kardashian-Jenner	0.3156

Maluma	0.4802	Jennifer	0.3608	tennis-themed	0.3999	Daily	0.3154
JerseyShore	0.4801	de	0.3608	Skrein	0.3997	cheating	0.3153
theWrap	0.4788	rumors	0.3601	2018.0000	0.3986	weekly	0.3146
Gillert	0.4779	debunking	0.3599	1.Throw	0.3969	1Oak	0.3138
Bieber-inspired	0.4774	Angelina	0.3578	Night	0.3961	according	0.3138
51-date	0.4769	judgeThe	0.3574	Trump-based	0.3927	Justin	0.3127
	0.4763	'missed	0.3574	Batfleck	0.3924	Blake	0.3126
Stawiarz/Getty	0.4751	learned	0.3574	48-has	0.3901	divorcing	0.3125
McCarthy/Getty	0.4743	were—or	0.3562	Festival	0.3901	Us	0.3122
take...	0.4730	exclusively	0.3555	'smitten	0.3898	father-of-two	0.3116
CHUHSchools	0.4715	Tweets	0.3554	yoyotrav	0.3886	sighting	0.3109
//www.eonline.com/news/891815	0.4714	!	0.3548		0.3872	White	0.3104
Tine/UPI	0.4706	internet-breaking	0.3545	actor/writer/director/producer	0.3862	Mexico	0.3104
'So	0.4703	dating	0.3538	book.Wait	0.3861	VF.com	0.3099
'Twilight'	0.4695	Star	0.3535	Order.	0.3860	Sources	0.3094
with	0.4690	Britney	0.3524	video_id=	0.3848	Mail	0.3091
bethennyfrankel	0.4684	actress	0.3524	'Dressing	0.3847	relationship	0.3088
Comic-Con-themed	0.4680	years... that	0.3522	blogs/paps	0.3841	Cibrian	0.3082
Register/POOL	0.4673	Nierob	0.3521	Kardashian/Jenner	0.3838	top-secret	0.3082
PDA-tastic	0.4668	Online	0.3514	!	0.3832	Bieber	0.3082
cinematic-like	0.4657	'God	0.3514		0.3828	reporting	0.3079
wagsandwalks	0.4655	Magazine	0.3509	-Year	0.3824	so...	0.3075
jimmykimmel	0.4643	tabloids	0.3509	Photo	0.3821	told	0.3071
Lemon/Grimes	0.4640	debunked	0.3500	2017.0000	0.3814	?	0.3070

Wochit-All	0.4631	claims	0.3500	videowaywire	0.3810	’	0.3066
CrazySexyCool	0.4630	can	0.3498	11:38pm	0.3807	earthquake	0.3056
IGGYAZALEA	0.4628	accuracy	0.3494	’the	0.3806	Privacy	0.3054
USweekly	0.4625	site	0.3488	Teen	0.3802	wife	0.3050
royalty-themed	0.4621	false	0.3484	khloekardashian	0.3801	Cop	0.3048
Vogue-approved	0.4617	claim	0.3483	Owyoung	0.3791	Awards	0.3044
84-years-old	0.4611	’The	0.3480	Proavtiv	0.3787	Baldwin—and	0.3043
Muschietti	0.4611	34.0000	0.3468	Younes	0.3783	Kourtney	0.3037
Prancercize	0.4602	priority-wise	0.3465	2005–07	0.3781	cheated	0.3035
PyeongChang	0.4601	subscribing	0.3462	then-couple	0.3760	a	0.3035
naakaakka	0.4598	US	0.3461	once-svelte	0.3758	Grammy	0.3032
twerked	0.4591	true.	0.3460	Harvard-alumni	0.3741	Kanye	0.3030
TheTonyAwards	0.4591	©	0.3459	Nicholaw	0.3741	39.0000	0.3027
	0.4591	Longoria	0.3457	License	0.3741	France	0.3027
Strauss/Invision/AP	0.4590	sources	0.3455	Tweeden	0.3730	baggy-pants	0.3026
//t.co/U5S24qzchL	0.4584	33.0000	0.3453	Markle—specifically	0.3721	49.0000	0.3025
Rentz/Getty	0.4580	Style	0.3452	pic.twitter.com/bGbipwMmxO	0.3714	by	0.3018
Day…	0.4579	has	0.3450	>	0.3700	claims	0.3010
kimksnapchats	0.4578	TMZ	0.3449	|	0.3695	Clooney	0.3010
LAPDHQ	0.4573	50-years-old	0.3447	Alamuddin	0.3686	that	0.3009
royal-inspired	0.4570	Pattinson	0.3442	WWE	0.3667	36.0000	0.3007
chrissyteigen/Instagram	0.4569	ve	0.3437	//bit.ly/2hv8QF8	0.3665	Trump	0.3006
kourtneykardash	0.4565	Email	0.3431	’Betty	0.3654	Rock	0.3003
G-Eazy	0.4557	NW	0.3430	Film	0.3647	Rihanna	0.3002

night. . .	0.4549	bogus	0.3428	//clipreview.net/yt/copyright	0.3645	Shelton	0.3002
9423695.0000	0.4546	1994–2004	0.3427	de	0.3635	ex	0.3001
Pool/Getty	0.4540	rumours	0.3426	Jelena	0.3632	film	0.2999
Photo/Chris	0.4536	ex-spouses	0.3426	namedAw	0.3631	LA	0.2998
Vennaro	0.4526	FR	0.3424	Chavez/Getty	0.3628	exclusively	0.2998
StevieRyan	0.4507	Invalid	0.3408	Thrones	0.3626	tabloids	0.2997
Un-researched	0.4501	'It	0.3396	rocher	0.3618	sources	0.2997
!	0.4494	unfounded	0.3394	acne-fighting	0.3618	spokesman	0.2995

Table C.4: Gossipcop Body-text top-100 token scores for different predictions.