2022-09

MULTI-DIMENSIONAL PROFILING OF CYBER THREATS FOR LARGE-SCALE NETWORKS

Calnan, Michael C.

Monterey, CA; Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

MULTI-DIMENSIONAL PROFILING OF CYBER THREATS FOR LARGE-SCALE NETWORKS

by

Michael C. Calnan

September 2022

Thesis Advisor:	Armon C. Barton
Co-Advisor:	Gurminder Singh

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2022	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE
MULTI-DIMENSIONAL PROFILING OF CYBER THREATS FOR LARGE-SCALE NETWORKS

5. FUNDING NUMBERS

6. AUTHOR(S) Michael C. Calnan

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release. Distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT (maximum 200 words)

Current multi-domain command and control computer networks require significant oversight to ensure acceptable levels of security. Firewalls are the proactive security management tool at the network's edge to determine malicious and benign traffic classes. This work aims to develop machine learning algorithms through deep learning and semi-supervised clustering, to enable the profiling of potential threats through network traffic analysis within large-scale networks. This research accomplishes these objectives by analyzing enterprise network data at the packet level using deep learning to classify traffic patterns. In addition, this work examines the efficacy of several machine learning model types and multiple imbalanced data handling techniques. This work also incorporates packet streams for identifying and classifying user behaviors. Tests of the packet classification models demonstrated that deep learning is sensitive to malicious traffic but underperforms in identifying allowed traffic compared to traditional algorithms. However, imbalanced data handling techniques provide performance benefits to some deep learning models. Conversely, semi-supervised clustering accurately identified and classified multiple user behaviors. These models provide an automated tool to learn and predict future traffic patterns. Applying these techniques within large-scale networks detect abnormalities faster and gives network operators greater awareness of user traffic.

14. SUBJECT TERMS deep learning, machine learning, classification, clustering	**15. NUMBER OF PAGES** 83
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

MULTI-DIMENSIONAL PROFILING OF CYBER THREATS FOR LARGE-SCALE NETWORKS

Michael C. Calnan
Captain, United States Marine Corps
BS, United States Naval Academy, 2015

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2022

Approved by: Armon C. Barton
Advisor

Gurminder Singh
Co-Advisor

Gurminder Singh
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Current multi-domain command and control computer networks require significant oversight to ensure acceptable levels of security. Firewalls are the proactive security management tool at the network's edge to determine malicious and benign traffic classes. This work aims to develop machine learning algorithms through deep learning and semi-supervised clustering, to enable the profiling of potential threats through network traffic analysis within large-scale networks. This research accomplishes these objectives by analyzing enterprise network data at the packet level using deep learning to classify traffic patterns. In addition, this work examines the efficacy of several machine learning model types and multiple imbalanced data handling techniques. This work also incorporates packet streams for identifying and classifying user behaviors. Tests of the packet classification models demonstrated that deep learning is sensitive to malicious traffic but underperforms in identifying allowed traffic compared to traditional algorithms. However, imbalanced data handling techniques provide performance benefits to some deep learning models. Conversely, semi-supervised clustering accurately identified and classified multiple user behaviors. These models provide an automated tool to learn and predict future traffic patterns. Applying these techniques within large-scale networks detect abnormalities faster and gives network operators greater awareness of user traffic.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

1D-CNN	one-dimensional CNN
AI	artificial intelligence
ANN	artificial neural network
API	application programming interface
ASN	autonomous system number
AWS	Amazon Web Service
CDN	content delivery network
CNN	convolutional neural network
CUDA	Compute Unified Device Architecture
DOD	Department of Defense
DODIN	Department of Defense information network
ERN	Enterprise Research Network
FN	false negative
FP	false positive
FPR	false positive rate
GPU	graphics processing unit
HPC	high performance computing
IoT	internet of things
MAGTF	Marine Air-Ground Task Force

MLP multi-layer perceptron

NIC network interface card

NPS Naval Postgraduate School

PCAP packet capture

ReLU rectified linear unit

RFC random forest classifier

ROS random over-sampling

RUS random under-sampling

SSH secure shell

TCP transmission control protocol

TFLOPS terra-floating point operations per second

TN true negative

TP true positive

TPR true positive rate

URL uniform resource locator

CHAPTER 1:
Introduction

1.1 Motivation

The United States Marine Corps has grappled with cyberspace adversaries for years. While not a kinetic warfighting domain, cyber-warfare creates real threats by exploiting vulnerabilities as surely as maneuver warfare. Moreover, Fleet Marine Forces are cyber-physical systems, with information systems performing critical roles within each Marine Air-Ground Task Force (MAGTF) [1]. As a result, the Marine Corps must leverage rapid decision-making throughout the technological kill chain to gain an advantage. It is therefore imperative that developing technologies mature to this end.

The recent Marine Corps Force Design 2030 Annual Review states that cyber may compose preliminary actions in a peer conflict. The force must "leverag[e] expertise across the total force. . . by accelerating advanced technology development. . . in areas like artificial intelligence, data science,. . . , cyber,. . . and other technology fields" [2]. In particular, harnessing machine learning and artificial intelligence (AI) to harden Marine Corps networks is critical in the force's defensive posture. Through AI, information system networks can actively sense and respond to threats decisively [1]. As adversaries develop more sophisticated and subtle methods of conducting network infiltration and exploitation, the Marine Corps must continue to refine and improve its defensive countermeasures. Moreover, as AI begins to coordinate cyber-attacks, AI should also integrate into the Marine Corps' network security.

Comprehensive security measures are difficult to develop for large-scale networks such as those employed by military forces. Innovations from smaller, experimental environments rarely generalize to enterprise networks as large as the Department of Defense information network (DODIN) [3].

A network's edge is the area that defines the boundary of protected systems and the external worldwide Internet. Security at the network edge requires constant adaptation to changing traffic patterns and threats. The heterogeneous nature of enterprise networks further

exacerbates security considerations. These types of networks service many diverse platforms across a wide network edge. The resulting complexities make traditional firewalls and intrusion detection systems too limited to adequately handle the complex operating environment, as static rule-based systems cannot handle new attacks [4]. The interaction between these intrusion detection systems and human operators requires constant attention, time-consuming efforts, and continual skill upgrade to remain up-to-date. Machine learning is uniquely suited to rapidly gaining situational awareness of network operations and proactively reacting to threats in real-time. This research explores the gap in machine learning in firewall emulation and automation of standard security features.

1.2 Research Objectives

The following research questions address the problem space of automated edge security.

Hypothesis: packet capture (PCAP) records can classify network traffic as benign or malicious and profile abnormal user behaviors through machine learning within large-scale networks.

1. What deep learning techniques can emulate firewalls to classify network traffic? This question frames common problems with intrusion detection systems and seeks to classify packet traffic effectively. By observing abstract patterns from known benign and malicious packets, deep learning models can predict future traffic and demonstrate the validity of machine learning in automating firewall functionality. Additionally, exploring the efficacy of different deep learning model types can determine the optimal machine learning solution for this problem space.
2. What machine learning techniques can profile user behaviors? User behavior is a standard network traffic pattern shared amongst multiple users. These behaviors could include typical applications, temporal patterns, and other distinguishing features. This question involves exploratory learning, where classes of user behaviors are unknown and may change between networks. It may also provide a profiling tool to analyze different behaviors to determine abnormal traffic behaviors.
3. How can these techniques be implemented within large-scale networks? Due to the enterprise nature of the Marine Corps and joint networks, this line of inquiry seeks to determine whether solutions from the two previous problems apply to large-scale

network environments. In addition, the use of large-scale network traffic confirms the feasibility of machine learning solutions.

1.3 Scope

This thesis focuses on understanding network behavior through PCAP analysis using machine learning to reveal observable patterns within large-scale networks. Although other cyber threat analysis vectors exist, such as packet payloads and network log files, PCAP records are the sole emphasis in this research. In addition, Naval Postgraduate School (NPS) Enterprise Research Network (ERN) traffic feeds machine learning model training and evaluation. However, additional validation on real-time, live, tactical networks is outside the scope of this research. Furthermore, transfer learning of models across multiple data sets is limited due to the time constraints in collecting and constructing the data pipeline.

1.4 Contributions

This thesis makes the following contributions.

- We demonstrate the validity of deep learning in real-world network traffic.
- We demonstrate the validity of convolutional neural networks for packet classification.
- We demonstrate a novel method of profiling user behaviors and predicting future traffic patterns.

1.5 Organization

This thesis is organized into the following chapters:

Chapter 2 defines key network security areas and explains the machine learning process. The chapter also provides an overview of deep learning and describes the models used in the research.

Chapter 3 describes the methodology for the experiments. Specifically, this chapter includes the design of model architecture, learning patterns, and other training and evaluation processes—data collection, cleaning, transformation, visualization, and exploration.

Chapter 4 presents the results from experiments and evaluates models within the context of network security.

Chapter 5 summarizes the research and identifies critical limitations and potential for future work.

CHAPTER 2:
Background Research

This chapter provides a literature review of machine learning and related research area topics. First, network traffic analysis techniques are described. Next, firewall automation through several deep learning and clustering techniques are presented. Machine learning considerations are then examined when working with network data. Finally, this chapter seeks to address missing elements in current research and identify the problem area this research addresses.

2.1 Network Traffic Analysis

As information technology continues to mature, network traffic becomes significantly larger and more complex. Large-scale enterprise internet of things (IoT) networks service many diverse applications across heterogeneous platforms. No longer does a single point of entry suffice for more extensive networks. Multiple firewalls, demilitarized zones, tunneling over virtual private networks, and other considerations have increased the domain size that the network administrators must monitor. As such, network traffic analysis has become a key consideration and a continuously growing field. While many different tasks exist under the network traffic analysis hierarchy, the automation of network security and identification of users is significant.

2.1.1 Network Traffic Classification

At its most basic, Network Traffic Classification is the process of identifying classes of network traffic based on patterns in network data. There is no standardized list of classes, as the requirements vary by application. Each problem may require a different number of classes and may account for different types of traffic. For example, Rezaei and Liu describe how there may be as few as two classes, such as benign or malicious, or one for every application running across a network router [5], but there can be a larger number as well. Additionally, the number of classes does not indicate the difficulty of a classification problem, such as identifying the broad fields of normal and malicious network traffic.

Regardless, there are three classification techniques used more extensively than any other: *port-based*, *payload-based*, and *statistics-based*.

Port-based classification is the simplest to implement but performs poorly with modern network traffic. The port-based classification technique was among the first methods for classifying network traffic. The source and destination ports are analyzed, and traffic is classified based on which typical application uses the port. It is easy to create such models, especially when only several applications are running. However, today with the diversity of network traffic, applications may share ports or use multiple. Thus, port-based classification is relatively obsolete in current network traffic analysis [6].

Payload-based classification is another commonly used classification technique. This classification process is the most complex but generally has the best accuracy. By inspecting packet payload information, known as deep packet inspection, key network traffic patterns can be deduced [7]. While this remains the most accurate method, encryption may skew payload information, causing a significant issue for model performance. Additionally, Lopez et al. show that deep packet inspections may violate confidentiality policies and common network security standards [6]. Payload data scales poorly, requiring sizable storage and computational capacity for analysis. Therefore, the lack of scalability makes this method infeasible in large-scale networks.

Statistics-based classification is a more recent method and has found effective use in machine learning by analyzing statistical information to determine network data patterns. Statistical measurements can be created by generating packet stream information from aggregated packet or router collections. These measurements feed into machine learning models, which extract high-level patterns. Moreover, statistics-based classification can operate using packet header information alone, reducing data capacity requirements and removing confidentiality concerns. Chockwanich and Visoottiviseth and Lopez et al. show the validity of machine learning models using this method of classification to achieve high levels of performance in recent research [6], [8]. Because of the compatibility with machine learning, this research will focus on statistics-based classification.

2.1.2 Cyber Profiling

User behavior profiling is the process of identifying and categorizing a person's cyber behaviors. It is a branch of criminal profiling, in which the actions of criminals are grouped to understand motives and tendencies. Moreover, profiling can match a person's actions to their behaviors and personality [9]. Since humans are the primary operators of computer networks, behavioral profiling can be applied to a person's network use. These behaviors include user information such as locations and past actions [10]. As a subset of anomaly detection, cyber profiling involves data exploration and visualization to allow operators to draw conclusions from the data. Therefore, presenting the patterns in an easily understandable format is essential.

Since large-scale networks generate high traffic volumes, machine learning is suitable for handling profiling actions. By examining traffic, algorithms can dynamically define commonalities among different behaviors [10]. The extraction of this information is fed to security professionals to gain a greater situational awareness of the network. In addition, machine learning lends itself to profiling, as Kipane explains that behaviors and patterns constantly change [9]. Cyber profiling with machine learning can be accomplished using unsupervised or semi-supervised algorithms, such as clustering models. The output of these models can be visualized to understand normal and abnormal behaviors. Further analysis and profiling actions after visualization are outside the scope of this research.

2.2 Deep Learning

Deep learning uses artificial neural networks (ANNs) to solve machine learning problems. Based on organic neural connections in brains, neural networks seek to emulate effective pattern recognition behavior. These neural networks have seen extensive use in fields such as image processing, natural language processing, and audio recognition [11]. In addition, deep learning models are used in network traffic analysis to automate network security, often outperforming traditional alternatives [8]. Network traffic classification commonly uses convolutional neural networks and multi-layer perceptrons for deep learning.

2.2.1 Multi-Layer Perceptrons

Multi-layer perceptrons (MLPs) are the foundational form of ANNs. The root component of a MLP is the artificial neuron. Each neuron is self-contained and operates by accepting input from neurons in a previous layer and producing a single output value, as seen in Equation 2.1. Unique values, known as weights, scale all neuron inputs. In addition, a distinct weight, known as the bias, is independent of other input values and unique for each neuron. The summation of these scaled weights is sent as input to the neuron's activation function to produce a single output value. A simple neuron can be representation mathematically as shown in Equation 2.1. The vector x contains all inputs from the previous layer, with w acting as the vector of weight values. These two vectors are transposed, and fed as input into the activation function, shown by φ. Multiple neurons constitute MLP layers, in which each neuron in a single layer feeds into every other neuron in the next, as shown in Figure 2.1.

$$h_w(x) = \varphi(x^T w) \tag{2.1}$$

[12]

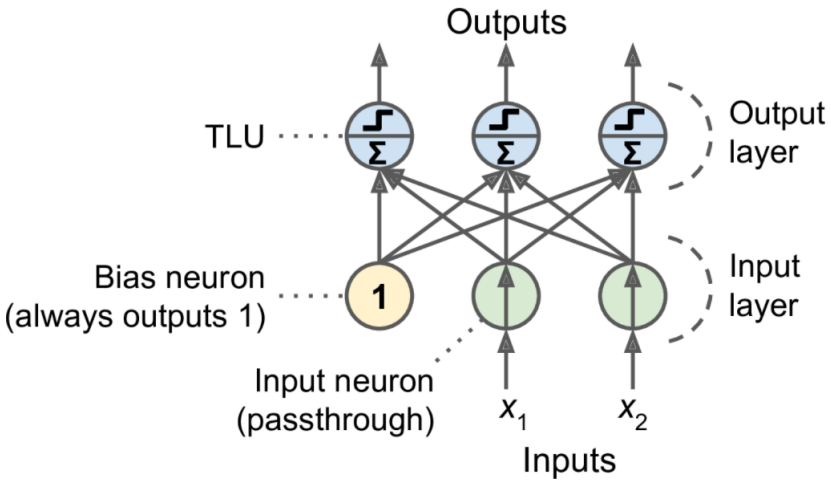

Figure 2.1. A Simple MLP Architecture. Source: [12].

Activation functions perform mathematical operations on the summation of scaled inputs to a neuron. Two functions used in this research are rectified linear unit (ReLU) and softmax. ReLU is simple and effective, returning the input value if greater than zero, else returning zero. This function effectively solves propagation issues such as the vanishing gradient problem and requires little computational power [12]. The other activation function is softmax, which is used in the final layer of multi-class classification networks. The algorithm takes the neurons in the layer and computes the probabilistic distribution. Each output represents a class's likelihood of membership.

Training a multi-class MLP classifier consists of processing batches of data, computing the loss using categorical cross-entropy, and updating neuron weight values and bias through an optimization function, continuing until the epoch is complete. The categorical cross-entropy loss function takes the softmax results and computes a loss metric for each class, which can be fed via backpropagation to update neuron weights. The optimization function seeks to minimize the loss. This process continues through a data set for the designated number of epochs. Upon completion, the model can classify future data.

A common issue with MLP is regularization or preventing overfitting. Overfitting occurs when the model fits the training data too closely, resulting in a loss of performance on new data. This loss of generalization often means new predictions are poor, but the model performs well on training data. Dropout layers are introduced to solve this problem. Dropout layers randomly select a specified percentage of neurons and removes their output to the next layer of the network. Dropout adds randomness to the training process, reducing the possibility of overfitting the training data. For example, Geron shows how a 10-50% dropout can add regularization to the model while adding little computational complexity [12].

2.2.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) use convolutional filters to extract key, high-level features from data. The CNN draws behavior patterns by taking input and analyzing local data. The critical difference separating CNN capabilities from traditional ANN is the use of convolutional and pooling layers while removing fully connected layers in favor of localized shared weights [7].

Convolutional layers are not connected to every neuron of the previous layer but instead use

receptive fields of local neurons as input. These receptive fields are created through filters, which are matrices of weight values. This filter is applied across the entire input, illustrated in Figure 2.2, to determine how far a distance to shift. The combined output of all neurons using the same filter is known as a feature map and feeds into the next layer as input. For additional complexity, multiple filters can be used at each layer resulting in multiple feature maps [12]. This stacking of feature maps can be thought of as adding dimensions to the input data, whereby each additional filter adds another layer of depth.

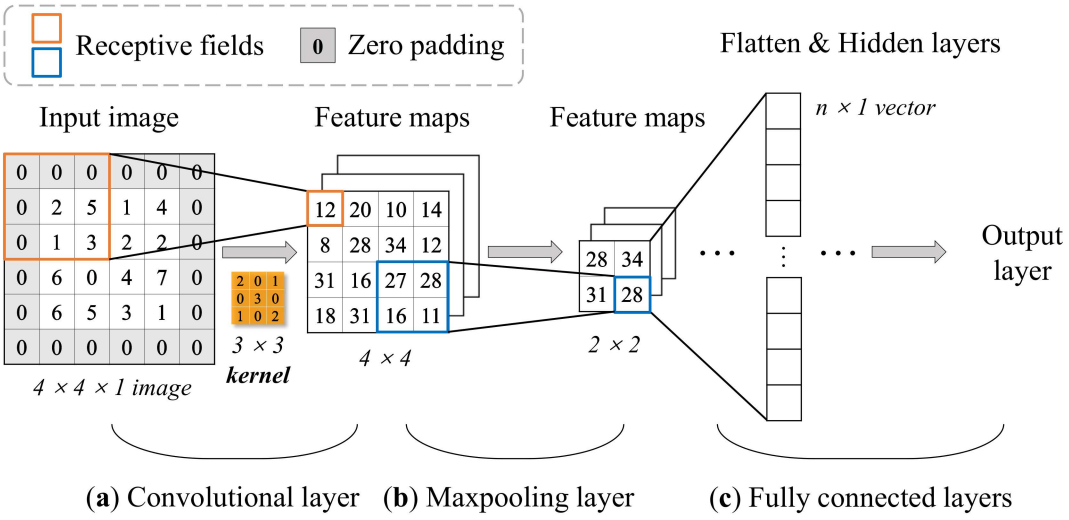

Figure 2.2. An Example of a CNN. Source: [13].

This functionality can be mathematically summarized at the neuron-level in Equation 2.2, where $Z_{i,j,k}$ is the output of the neuron located in row i, column j, of feature map k. s_h and s_w are the vertical and horizontal stride lengths. f_h, f_w, and $f_{n'}$ are feature maps from the previous layer. $x_{i',j',k'}$ is the output of the neuron at the previous layer, and $w_{u,v,k',k}$ is the weight of all neurons in feature map k [12].

$$Z_{i,j,k} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f_{n'}-1} x_{i',j',k'} \times w_{u,vk',k} \; with \begin{cases} i' = i \times s_h + u \\ j' = j \times s_w + v \end{cases} \tag{2.2}$$

[12]

Pooling layers reduce the input vector size by locally sampling to create new feature maps. This sampling occurs using one of several functions. The most common is max pooling, where a subset of input is searched for the maximum value, reducing a larger matrix to that single value, shown in part b of Figure 2.2. This pooling removes many values from the input feature map but adds invariance in the output [5]. While other functions exist, max pooling remains the most used for its simplicity and minimal computational requirements.

The use of CNNs in network traffic analysis is widespread. Vinayakumar, Kwon, and Shahraki et al. demonstrate how one-dimensional CNNs (1D-CNNs) are frequently used to handle the sequential vector format of network traffic [11], [14]–[16]. Network traffic may aggregate to form two-dimensional images to best use the two-dimensional pattern recognition capabilities for which CNNs are known. Each packet composes a row of pixels, and each feature composes a column. Lopez, Hwang, and Wei Wang et al. reshape network data to allow for network traffic to be fed as images into CNN models [6], [15], [17]. Kwon et al. and Haghighat et al. prove that CNNs can quickly extract network traffic behaviors with five or fewer convolutional layers, allowing for shallow models to be effective [4], [14].

2.3 Clustering

Clustering is a powerful exploration tool to identify patterns with potentially unlabeled data. This pattern identification is useful in anomaly detection, as clusters may indicate new classes of traffic or highlight outliers that may warrant additional analysis. In user behavioral profiling, clustering can assist security professionals in identifying normal and abnormal traffic patterns. In essence, clustering identifies regions of data points that are closely grouped, allowing for further extrapolation of anomalies or cluster classification [18]. This research uses clustering to understand common, shared features among large quantities of data, thereby assisting in cyber profiling.

2.3.1 K-Means

K-Means is an effective and direct method of clustering data by examining groups when creating a set number of clusters. The algorithm creates k clusters, a modifiable hyperpa-

rameter. First, the model creates randomly generated centroids and assigns each data point to the closest cluster. Next, all points are analyzed to ensure accurate association with the correct centroid. Suppose there are discrepancies; the model re-assigns centroid locations and tests again. The process then repeats until centroids no longer require updating, at which point the model can classify new data points. K-Means is especially effective because the algorithm is guaranteed to converge. However, it may do so at a sub-optimal answer [12].

Nevertheless, due to the simplicity of operation and the minimal storage requirements, K-Means is beneficial as a simple clustering tool. Combined with in-depth data exploration, K-Means can overcome some deficiencies by tuning hyperparameters. Fan and Liu used K-Means for network traffic analysis when unlabeled network data is handled [19].

Mini-batch K-Means is used to perform clustering for user behavioral profiling on ERN data. Mini-batch K-Means differs from K-Means by analyzing only a small subset of data, instead of the whole dataset, before making cluster center adjustments. This method significantly reduces the training time while performing only marginally worse than standard K-Means, as demonstrated by Scikit-Learn [20]. With larger-sized datasets, like those found in enterprise networks, training time is a crucial consideration that mini-batch K-means solves.

2.4 Machine Learning Considerations for Network Traffic

Specific considerations must be assessed to ensure machine learning model performance. Among these are PCAP, the data pipeline and critical features, imbalanced data handling, and the performance measurements used to compare model effectiveness.

2.4.1 Packet Capture Records

PCAP files are a well-known application programming interface (API) used in network traffic collection and analysis. These files contain all packets routed through a collection point for a specified time. Generally, entire packets are captured and stored. While examining a whole packet is ideal for network security purposes, encryption of packet payloads and network confidentiality policies may prevent complete analysis. Thus, packet header information is considered the most accessible and least intrusive data collected from network traffic. In addition, network traffic classification can easily convert individual packet

fields to machine learning input. Since supervised learning requires a significant amount of data to learn the complex behaviors of the network, the sizeable PCAP files make excellent training sources.

2.4.2 Data Pipeline and Key Features

Processing network traffic into formats suitable for deep learning models involves specific data capture methods and pipelines. Generally, there are two methods in which network data is analyzed: per packet and per stream.

Analyzing each packet is a more simplistic method, but may lack sufficient depth as input for deep learning models. Data are primarily gathered from packet header fields, which are easily accessible and can be fed directly to machine learning models. PCAP collections are configured to capture required features, further simplifying the process. However, pre-processing is still required before effective use in network traffic analysis. First, data cleaning analyzes each field, removing outliers and filling missing values. Packets may be dropped altogether if enough fields are missing. Next, PCAP packets are transformed into numeric values. For categorical data, one-hot encoding or enumeration may be used to denote specific fields [16]. Fields with many possible values, such as IP addresses, require more abstract solutions. Normalization is the last pre-processing step, in which features are scaled with the entire dataset to provide a normal distribution of values [16].

In contrast, analyzing multiple packets through statistical meta-data for each packet stream provides robust network data. A stream is defined as a group of IP packets that share characteristics such as protocols and source and destination IP addresses and ports. Stream information is collected at routers, commonly placed at the network boundary, or processed through the conversion of packet aggregates. However, the additional required processing turns stream analysis into a time-consuming process. Mohammed et al. demonstrates that stream-based information is generally more successful in determining network traffic patterns [7].

2.4.3 Handling Imbalanced Data

Imbalanced data refers to the disparity in the number of samples between different classes. It is a critical problem in supervised machine learning if models are trained with imbalanced

datasets. In these circumstances, the models will often over-classify the majority class, the class with the highest sample count [21]. For example, in network traffic classification, an overwhelming majority of data is benign, while malicious traffic composes only a small percentage. In these cases, a model predicts the majority class every time, resulting in few misclassifications. As expected, the under-represented class often is classified poorly. Thus, techniques for mitigating imbalanced data are critical in classification problems. Johnson and Khoshgoftaar define algorithmic and sampling methods as two effective means for overcoming data imbalance [21]. This research will examine both as well as hybrid combinations.

Sampling seeks to reduce the imbalance in the data before feeding it into the machine learning model. The two most basic types are random under-sampling (RUS) and random over-sampling (ROS) [21]. RUS removes data points from the majority classes, reducing the imbalance among larger classes. However, RUS may cause the data to lose fidelity as potentially significant data points are lost. In contrast, ROS adds synthetic data points to the minority class, reducing the imbalance among the under-represented classes. However, ROS increases training times and develops potential memory challenges as the data set grows. For the scope of this research, large-scale network traffic renders RUS as the only feasible sampling technique.

Algorithmic methods alter the machine learning model instead of the data. Specifically, the learning or prediction functions are changed to increase the significance of the minority class [21]. This research examines the use of class weights as the preferred algorithmic technique. Class weights find the proportion of data points for each class in the training dataset and scale the model output, as shown in Equation 2.3. Where the weight for each class is the proportion of total samples to the samples of that class. Through this scaling, minority class samples are given more importance in the loss function, incentivizing the model to classify them correctly. Class weights are only used during training and add little additional computational requirements. However, training with class weights assumes the data retains the same imbalance characteristics to remain effective. Should the imbalance change, the model may begin to perform poorly.

$$Weight_x = \frac{Total\ Samples}{Samples_x} \tag{2.3}$$

2.4.4 Metrics of Performance

Machine learning model performance requires specific metrics which vary in importance depending on the network traffic analysis problem. Performance indicators are a combination of expected and predicted results. These are further divided into true and false predictions. For classification, there also exists classes of results of positive and negative, indicating a prediction for the class and prediction of any other class.

Thus, a false positive (FP) renders an incorrect prediction of a class, while a true negative (TN) indicates the prediction correctly deduced that the result was not the indicated class. A true positive (TP) indicates a correct class prediction, while a false negative (FN) denotes an incorrect prediction of another class. Key machine learning metrics are formed by combining these classes of predictions and the ground truth of correctness.

Accuracy is the ratio of correct predictions compared to all predictions.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision is the ratio of correct predictions of one class across all positive predictions. It measures the model's probability of a positive prediction being true, similarly known as a hit rate.

$$Precision = \frac{TP}{TP + FP}$$

Recall measures the ratio of correct predictions of one class across all correct expected predictions, also known as the true positive rate (TPR).

$$Recall = \frac{TP}{TP + FN}$$

false positive rate (FPR) measures the rate of false positive predictions across all expected negative class samples. It is known as the likelihood of a false alarm.

$$FPR = \frac{FP}{FP + TN}$$

F1-Score combines the recall and precision metrics in a harmonic mean, created as a generalized performance metric for machine learning models.

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

While these measurements can be applied to machine learning models, some are less indicative of deep learning models for network traffic analysis. Since the data is imbalanced, accuracy becomes less important as the model can predict the higher percentage class and retain high accuracy. This is especially true for network traffic, where benign traffic makes up most data, and malicious traffic may only be a fraction in comparison. Therefore, metrics such as precision or recall are more practical than accuracy.

2.4.5 Transfer Learning

Transfer learning allows a model to be re-trained on a new dataset while maintaining the learned weights across crucial layers. However, training a model can become consuming in time and effort. Frequently, supervised learning requires substantial labeled data for training, which may not be available. However, a fully trained model can transfer learned patterns to new environments, a process known as transfer learning [5].

Transfer learning begins with a trained model, which has weighted layers pre-established. Then, a new model is created to solve a similar problem within a different data domain. First, essential layers are identified, and their parameters are saved. Usually, these layers are closer to the input since the initial patterns remain the same, but the outcome or predictions may differ. This new model then initializes additional layers and begins the standard training process on the desired dataset. Once training is complete, the new model contains the influence from the previous iteration, thereby improving model performance.

Transfer learning is beneficial when a new dataset is not large enough to train a model to satisfaction. Since the saved layers are weighted to extract useful features from existing

data, there is a carryover to the model's ability to detect patterns. This is especially true where network traffic is handled. Rezaei and Liu show how a smaller dataset can overcome a lack of labeled information using a larger, different dataset to pre-train specific layers of supervised models [5].

2.5 Problem Space

While deep learning has seen extensive use in network traffic analysis, there is little research regarding the emulation of current network edge security, such as firewalls. This is especially true for modeling packet header data, which is the same format in which firewalls receive data to make decisions, rather than packet payload data since payload information is often unavailable and cannot be relied upon for generalized solutions functional across many networks. Additionally, there is little work in classifying traffic in broader classes, such as allowed or blocked. Most related research focuses on classifying a specific list of applications but rarely seeks to discover more extensive patterns of benign and malicious traffic. Firewalls use static rules created or influenced by humans to determine whether traffic should be allowed or blocked. The ability to use machine learning to automate this behavior has yet to be examined thoroughly.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
Methodology

3.1 Equipment and Software Tools

The following software and hardware tools are utilized in this research. The hardware consists of a high performance computing (HPC) environment at NPS, while the software consists of Python programming language libraries.

3.1.1 Hamming High Performance Computer

Hamming is a HPC system designed to provide distributed computing capabilities through clusters of nodes. Created in 2009 and named after a prominent NPS professor, Dr. Richard Hamming, the system grants students and faculty superior research capabilities. The Hamming environment consists of 4,282 computing cores with over 18 Terabytes of memory. In addition, there are 81 graphics processing unit (GPU) computing cores, resulting in 79,744 Compute Unified Device Architecture (CUDA) cores. Hamming can achieve upwards of 36.0753 terra-floating point operations per second (TFLOPS) across the entire cluster [22]. Task management is handled through a Slurm workload manager, allowing Unix access via secure shell (SSH). Compatible with GitLab and GitHub collaborative repositories, the system is well suited for computing-intensive research. These capabilities create a good environment for machine learning, data exploration, and visualization across larger datasets. Submitting batch jobs for simultaneous execution of multiple models reduces time spent training and evaluating model performance.

3.1.2 TensorFlow

TensorFlow is a distributed deep learning framework created in 2015 as a successor to DistBelief. Designed "for implementation and deployment of large-scale machine learning models" [23], TensorFlow can conduct all steps in a machine learning project. It is hardware agnostic, allowing deep learning models to run from mobile devices to HPC. By utilizing a TensorFlow Profiler, optimization of program runtime can be fit over GPU hardware nodes.

It functions through stateful dataflow graphs, which create simulated model computations. When run in parallel, these graphs can scale to larger projects. Each graph is composed of nodes with zero or more inputs and outputs, representing an instantiation of an operation. Values moved between nodes are called tensors, arrays of varying dimensions to support deep learning values. Utilizing Keras, a deep learning API for the Python programming language, complex models are easily tested and implemented on the TensorFlow platform. In addition, it can conduct collections of model performance metrics and statistical analytics, which can be analyzed in depth. This deep learning platform is also open-source, allowing for easy adoption in research environments, and ideal for the parallel computing capabilities of the HPC [23]. TensorFlow will be used in this research to create deep learning models for Network Traffic Classification.

3.1.3 Scikit-Learn Library

Scikit-Learn is a powerful open-source library for the Python programming language for machine learning and data analysis. Scikit-Learn can not only create machine learning models of varying types but has built-in performance estimators to measure the effectiveness of all potential models. Moreover, Internal libraries handle data transformations and pre-processing. The ability to conduct initial data exploration, model selection, model construction, and model training and validation makes it a powerful research tool. In addition, it allows for a wide range of data types, including other commonly used data analytics libraries such as NumPy and SciPy arrays, as well as Pandas DataFrames [20]. Scikit-Learn will be used in this research to create clustering algorithms for User Behavior Profiling.

3.2 Design

3.2.1 Data Collection

The primary dataset for training and validation is labeled network traffic from the NPS ERN. ERN is the primary NPS information system network that handles all campus traffic. The ERN PCAPs were captured over seven days, July 14 through July 20 of 2022. During this time, over 120 billion packets were captured. Collections were conducted through internal and external network interface cards (NICs), positioned on either side of the ERN firewall, as shown in Figure 3.1. These NICs collected all packet header information, along with

a timestamp. All packet payloads were dropped to maintain confidentiality and allow for unobtrusive traffic analysis. Furthermore, all NPS internal IP addresses were obfuscated to protect the privacy of users.

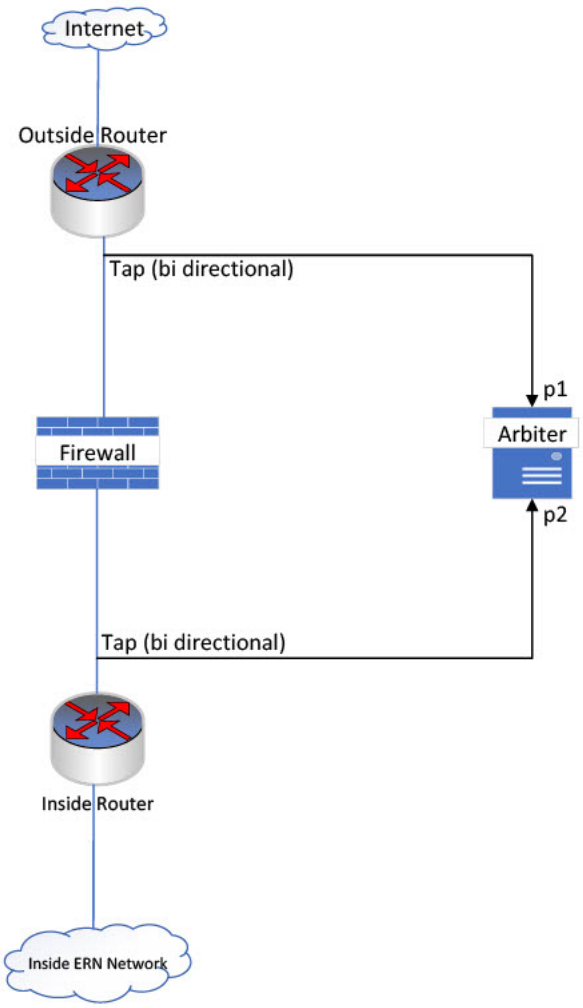

Figure 3.1. Logical Architecture for ERN Data Collections.

Labels were added to the PCAP data by comparing records collected at each NIC. All packets were stored under a subdirectory for each NIC and were run through a preprocessing data

pipeline to extract selected header fields. These included timestamp, source and destination IP addresses and ports, IP protocol, IP packet length, transmission control protocol (TCP) sequence number, and TCP acknowledgment flag. At this point, a separate preprocessing step compared packets captured from the internal NIC to those captured from the external NIC. Packets with the same identifying features (i.e., shared protocols, ports, and IP addresses) with timestamp deviation of 150 microseconds or less are assumed to be allowed through the firewall. Packets, where the source and destination IP addresses are both internal or external to the NPS domain, are considered not to have attempted to traverse the firewall and are labeled interior only or *exterior only*. The remaining packets, only seen by a single NIC, were considered blocked.

Following the initial packet labeling, a second pass through the data reclassifies erroneously blocked traffic as dropped. For example, if burst traffic and TCP reset packets of allowed streams were seen only by one NIC, they would be incorrectly classified as blocked. Therefore, a second pass through the traffic compares the same identifying features of blocked traffic and searches for corresponding allowed traffic. If there is a match, the label changes from *incoming blocked* and *outgoing blocked* to either *incoming dropped inside*, *incoming dropped outside*, *outgoing dropped inside*, or *outgoing dropped outside* depending on the direction of the packet and the associated NIC location.

The output from the data pipeline results in a cleaned dataset with labeled network packets. There are 10 potential labels for a packet, shown in Table 3.1. In addition, each packet has eight features, shown in Table 3.2. The dataset comprises 13,853 parquet files, each containing approximately 8.7 million packets. The breakdown makes access simple and reduces the memory required to read traffic segments. This dataset is used in the following sections and chapters for supervised machine learning classification.

Table 3.1. ERN Packet Classes.

Index	Name	Description
1	Incoming	Allowed packet from outside NPS ERN to inside
2	Outgoing	Allowed packet from inside NPS ERN to outside
3	Outgoing Blocked	Outgoing packet blocked at NPS ERN firewall
4	Interior Only	Packets with both IP addresses internal to the NPS ERN
5	Incoming Dropped Inside	Incoming packet only captured by inside NIC, but shares packet features with other allowed packets
6	Outgoing Dropped Inside	Outgoing packet only captured by inside NIC, but shares packet features with other allowed packets
7	Incoming Blocked	Incoming packet blocked at NPS ERN firewall
8	Exterior Only	Packets with both IP addresses external to the NPS ERN
9	Incoming Dropped Outside	Incoming packet only captured by outside NIC, but shares packet features with other allowed packets
10	Outgoing Dropped Outside	Outgoing packet only seen on outside NIC, but shares packet features with other allowed packets

Table 3.2. ERN Packet Features.

Field	Description
ip_protocol	The IP protocol, such as TCP or UDP
ip_src	The source IPv4 address from the packet
ip_dst	The destination IPv4 address from the packet
port_src	Source port of the packet
port_dst	Destination port of the packet
timestamp	Linux epoch timestamp of packet as float64
ip_len	The total length of the packet, specifically IP header plus payload
packet label	The packet label

Next, the labeled packet information is aggregated into packet streams. A stream is composed of traffic between two nodes, with one node being inside ERN and the other being outside. In a stream connection, packets are sent and received across the firewall. Packet streams can be captured over TCP connections or UDP connections. Packet streams are all unlabeled since all packets cross the firewall in a stream connection. Therefore, semi-supervised learning is applied to learn generalized labels over the stream dataset. This approach is discussed in detail in Section 3.3.2.

Streams are capped at a maximum length of 1,000 packets with overflow packets overwriting older packets circularly. For example, if 1,010 packets are captured, then the first 10 packets are overwritten as new packets are captured. In this sense, the stream always consists of the most recent packets captured within the connection. Each stream comprises shared information such as packet directions, lengths, and time differences. The features for each stream are shown in Table 3.3. The list features are aggregated statistics over each stream such as bytes in, bytes out, packets per second, etc. The *packet directions* field is a list of packet directions for all packets in the stream that, when aggregated, may be used by ML models to learn traffic burst patterns between two nodes. The dataset is split into 4,000

NumPy array files, each with an average of 40,000 streams. The following sections and chapters use the resulting dataset for semi-supervised user behavior profiling.

Table 3.3. ERN PCAP Stream Fields.

Fields	Description
ip_protocol ip_nps, ip_other port_nps, port_other	Connection information
other_country other_ASN other_lat, other_long	Geolocation features of IP address external to NPS ERN
list_in_packets, list_in_bytes list_in_mean_packet_length list_in_packets_per_second list_in_bytes_per_second list_out_packets, list_out_bytes list_out_mean_packet_length list_out_packets_per_second list_out_bytes_per_second list_t_duration	Aggregated totals of the last 1,000 packets captured for the given stream
packet directions	List of individual packet directions in a stream, where *outgoing* = 2, *incoming* = 3

For machine learning, the packet classification dataset is split into training and validation sets. The training set consists of network traffic collected on July 14, 2022. The validation set consists of network traffic collected at 1400 over the next five consecutive days from July 15 to July 20, 2022. By adjusting the validation set time window over several days, the machine learning model's ability to generalize over time may be examined. For the stream

dataset, the training data was captured on July 14, 2022. The validation set was captured on the same day, a few hours after the training data with no overlapping streams.

3.2.2 Data Exploration and Visualization

Data exploration and visualization examines the data for trends, distributions, and correlating factors that would impact its use for machine learning. The output of these processes are visual representations such as tables and graphs that depict the relationships between features in an easily interpretable manner. Specifically, in supervised learning, identifying the correlations of features to the sample's label is critical to understanding which features are most important for accurate model predictions, and which are least important. Likewise, exploring distributions over target features may reveal potential bias in the dataset, and exploring class distributions may identify penitential imbalances in the data.

Packet-Level Exploration

Feature correlations, as shown in Figure 3.2, were calculated by sampling 16 million packets across the entire dataset. Of note, autonomous system number (ASN), country name, and latitudes and longitudes are engineered features created from resolving the raw IP addresses and extracting geolocation information. Geolocation features are further discussed in Subsection 3.2.3.

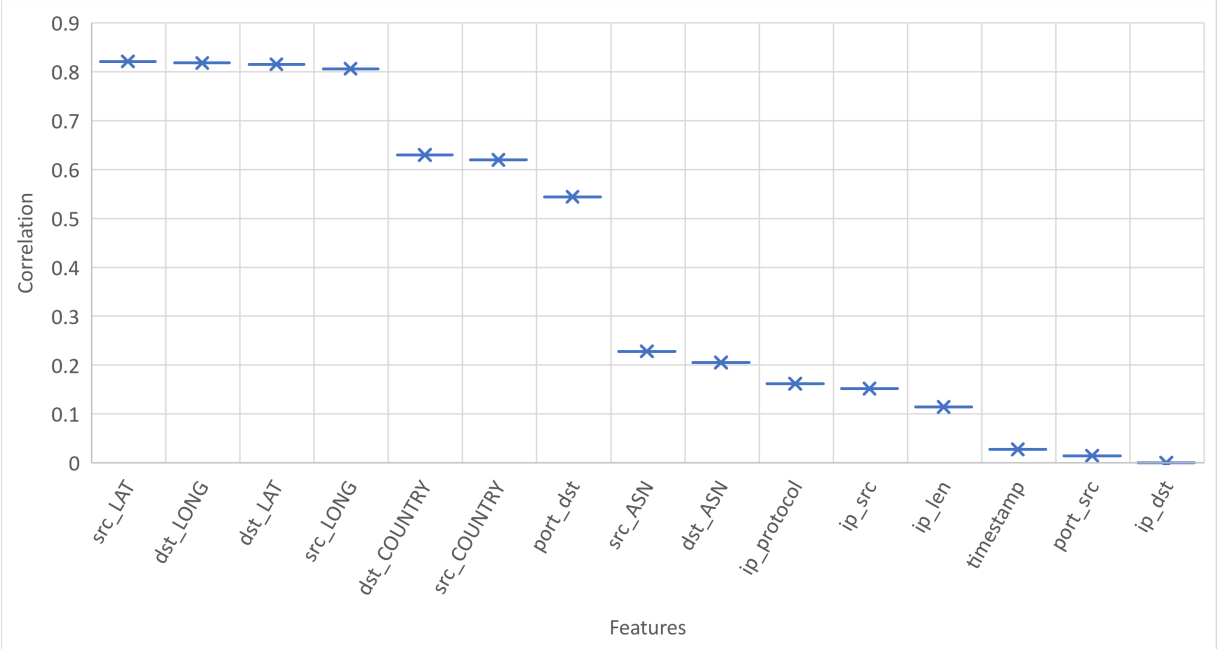

Figure 3.2. Feature correlation in relation to the target label.

It is clear from Figure 3.2 that timestamp, source port, and destination IP address have minimal correlation with the target label. IP protocol, source IP address, and IP packet length have somewhat more correlation, though still low with values under 0.2. However, engineered geolocation features have the most significant correlation with the target label; correlation values are greater than 0.6 for the country names, latitudes, and longitudes. Destination port and ASN numbers also retain moderate correlation values between 0.2 and 0.6.

Next, class distribution over the dataset is visualized using a histogram plot to understand class composition and membership, as shown in Figure 3.3. It is clear from the Figure that *interior only* traffic is the majority and makes up approximately 40% of all traffic. The next largest class is *incoming blocked* which contains 33% of total traffic volume. Allowed traffic classes total around 20% of traffic, with slightly more *incoming* than *outgoing*. The remaining classes compose 1.8% of total traffic. Since re-classified packets share packet features with allowed traffic, these packets merged into their respective allowed classes.

Interior only and *exterior only* are removed from the dataset, as they do not attempt to cross the firewall. These changes produce four classes of traffic, shown in Table 3.4. Of note, there is a clear imbalance between the amount of allowed and blocked packets.

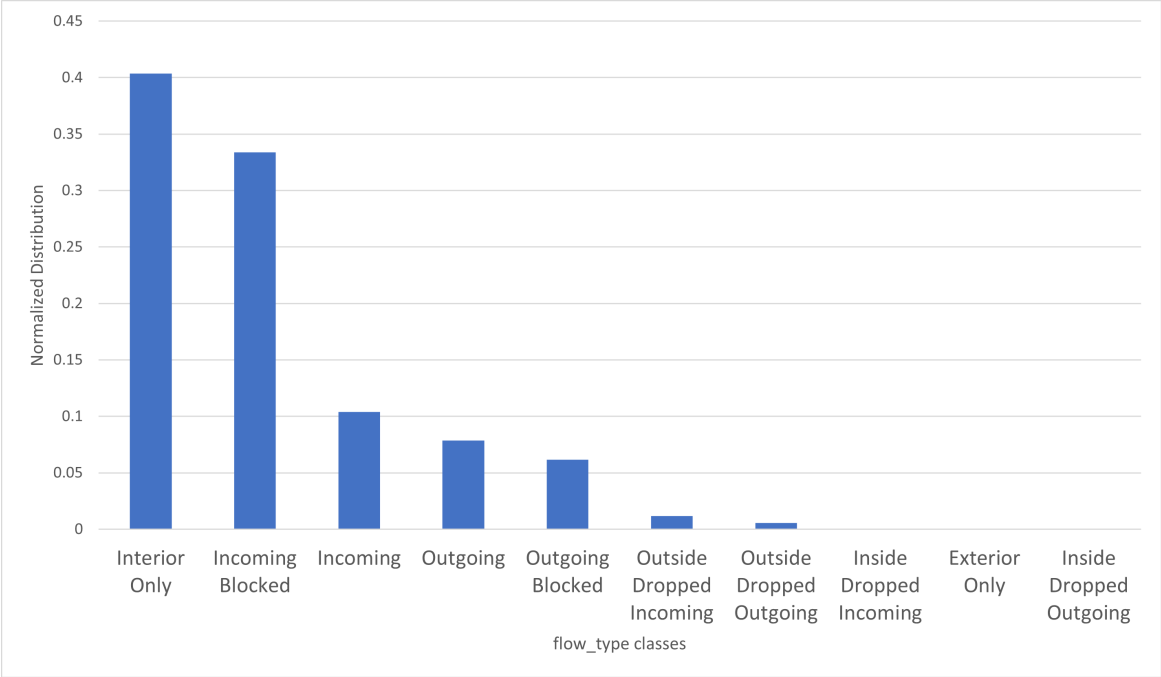

Figure 3.3. Target Feature Class Distribution.

Table 3.4. Simplified Class Distributions.

	Allowed	Blocked
Incoming	0.206	0.590
Outgoing	0.157	0.047
Total	0.363	0.637

3.2.3 Data Preparation

Data preparation involves the cleaning, feature selection, feature transformation, and feature scaling of input before feeding it into machine learning models. Data cleaning is conducted during the pre-processing pipeline, which involves removing any packets with missing data and dropping unidentifiable packets. Furthermore, no outliers were removed in order to more closely resemble real-world enterprise networks.

The feature transformation process was applied in the same way for both packet and stream datasets. An IP address gives little tangible information, especially since NPS obfuscates internal IP addresses. Therefore, IP addresses are transformed into helpful geolocation features, utilizing MaxMind's GeoIP2 API and MaxMind's open-source ASN, City, and Country geolocation databases [24]. The APIs resolve each IP address to the IP's ASN, country name, latitude, and longitude. ASNs indicate the IP address's routing network in reference to the world-wide-web. Country names are the country that the IP address resides in. Placeholder values of zero are added for any IP addresses not found within the databases. After this process, the source and destination IP addresses are dropped from the feature set, and the engineered features are added.

Feature scaling involves scaling or encoding data to give the machine learning model consumable input. Most data is not on the same numeric scale, and categorical features are incompatible without encoding. All numerical features, like IP packet length, are scaled between 0 and 1 by finding the mean and applying the deviation from that mean. Categorical features require a different approach since similar values, like ASN numbers, may indicate significantly different categories. For this problem, a binary encoder is used. Binary encoders assign a number to each category, convert that number to its binary form, and create a feature for each present digit. For example, a feature with 10 categories will be encoded to four binary features. After scaling and encoding, the data is ready to be fed into machine learning models.

3.3 Implementation

This section will explain the design and implementation of machine learning models. Packet classification models will be discussed first, where several deep learning models are designed. Next, user behavior analysis through clustering is examined, and semi-supervised

learning is applied.

3.3.1 Deep Learning Network Traffic Classification

Several deep learning models will be evaluated on their ability to conduct packet classification. A CNN and a MLP will be tested against a random forest classifier to determine the validity of deep learning for network traffic classification in this domain. Specific model hyperparameters will remain consistent across all deep learning models, reducing variability between experiments. Table 3.5 shows the hyperparameter values used for each model.

Table 3.5. Shared Deep Learning Hyperparameters.

Hyperparameter	Value
Maximum Epochs	100
Activation Function	ReLU (Softmax on final layer)
Loss Function	Categorical Cross-Entropy
Training Optimizer	Adam
Learning Rate	0.01, 0.001, 0.0001
Batch Size	512

The maximum number of epochs is set high to ensure convergence [8], [11], with early stopping if 10 epochs pass without improvement in loss [6]. The ReLU activation function is used on all layers excluding the final output layer, due to its low computational cost and high performance [5], [6], [11], [14]–[16]. As this is a multi-class classification problem, the final layer will use softmax as an activation function with categorical cross-entropy as the loss function [8], [14], [17]. Adam is the training optimizer and is generally regarded as a standard in deep learning research, as it provides numerous benefits when working on large datasets with complex models [5], [6], [8], [14]. A multi-step piece-wise learning rate scheduler with an initial learning rate of 0.01 is applied to the training procedure. After 10 epochs, the learning rate is reduced by a magnitude of 10 to 0.001. After another 10

epochs, it is further reduced to 0.0001. This piece-wise learning rate allows the model to quickly reach the global minimum value. Finally, a batch size of 512 ensures a balanced trade-off between stochasticity and model convergence as used by the Adam optimizer during training.

Multi-Layer Perceptron

A shallow MLP is used as the traditional ANN for classification. The architecture is described in Table 3.6. There will be three fully connected layers of 512 neurons, followed by an output layer of four, one for each major class. A low dropout is added between each fully connected layer to prevent overfitting the model. The input vector array is intentionally left blank to indicate that the size is variable depending on the training data. The pre-processing encoders may produce different sized input depending on the diversity of the data. This allows the model to better conform to the dataset. On ERN traffic, an average input array size of 64 is generated during pre-processing and fed to the model.

Table 3.6. MLP Model Architecture.

Layer	Info
Input	[x, 1] Vector
Fully Connected 1	512 Neurons
Dropout 1	0.2 dropout rate
Fully Connected 2	512 Neurons
Dropout 2	0.2 dropout rate
Fully Connected 3	512 Neurons
Fully Connected 4	4 Neurons (Softmax Activation)

Convolutional Neural Network

The CNN will be one-dimensional and relatively shallow. Only several convolutional layers and a single pooling layer are required for the model to extract useful data features [5],

[16]. Several fully connected layers follow this. Additionally, high dropout is added after each convolutional layer to add regularization to the model [12], [16], [17]. Initial model parameters are indicated in Table 3.7. For the same reasons as the MLP, the input vector array size is intentionally blank.

Table 3.7. 1D-CNN Model Architecture.

Layer	Info
Input	[x, 1] Vector
Conv1D 1	64 Filters, 3 Filter Size
Dropout 1	0.3 Dropout Rate
Conv1D 2	128 Filters, 3 Filter Size
Dropout 2	0.3 Dropout Rate
MaxPooling1D 1	2 Filter Size, 2 Stride Length
Flattening	
Fully Connected 1	512 Neurons
Dropout 3	0.2 Dropout Rate
Fully Connected 2	512 Neurons
Fully connected 3	4 Neurons (Softmax Activation)

Random Forest Classifier

A random forest classifier (RFC) will be used as a baseline model, against which all deep learning models will be compared. This ensemble will contain 100 trees in the forest. Gini will be the evaluation criteria at each split, with a required minimum of two samples to split and a minimum of a single sample per leaf. As random forests generalize well across complex datasets, they will serve as a comparative baseline. Additionally, extracting feature importance from the ensemble allows for further feature examination.

3.3.2 Clustering for User Behavior Profiling

Clustering

The clustering model fits the list features of each stream. First, input is transformed within the same scale in Euclidean distance, which is necessary to accurately compute clusters in feature space. Next, the mini-batch K-means algorithm creates 25 clusters. These clusters indicate up to 25 unique user behaviors for classification and were chosen as an arbitrarily large value that could be trained in a reasonable time. After fitting the model to the data, each cluster's five closest points in Euclidean space are taken and assigned as representative points. These points describe the behavior of the cluster. There is a possibility that points may be representative points for multiple clusters, indicating a more prominent shared behavior. These five points for each of the 25 clusters are placed within a table for further data analysis.

Analyzing Centroids and Profiling Behaviors

The table of representative data points allows for profiling of behaviors and assignment to clusters. First, each data point's IP address, external to the NPS domain, is taken and resolved into a URL. The stream information, geolocation features, and URL combines to create a table of user behaviors. Each cluster's five representative points are present in this table, resulting in 125 total rows. At this point, manual exploration is required. Genres of websites can be assigned to each IP address using the URLs. The cluster will be assigned a singular genre if most representative IP addresses exist in the same genre. Geolocation information will be used instead if the IP addresses cannot be resolved. Centroids sharing similar genres can be grouped into a singular behavior. Conversely, centroids without common genres or geolocations indicate a lack of behavior and are discarded. These behaviors are finally applied to each cluster, allowing for future data points to be assigned a behavior.

3.4 Chapter Summary

This chapter covered the design of the machine learning process and the implementation of machine learning models for automating network edge security. First, data collection and pre-processing was used to create datasets compatible with machine learning models. The features and labels were also examined, and data imbalance was discovered within the

network traffic. Next, random forest classifier and deep learning models were defined for experimentation on packet-level classification. Finally, user behavior identification through semi-supervised clustering methods was identified regarding stream-level traffic. These processes and models will be compared against one another using meaningful metrics in the following chapter.

CHAPTER 4:
Testing and Analysis

This chapter will cover the testing and results analysis of models outlined in the previous chapters. Beginning with packet classification, each model will undergo initial tuning using a validation dataset. Then, imbalanced data handling techniques will be applied to deep learning models to determine their efficacy for network traffic analysis. Next, each model will be compared against one another using meaningful metrics discussed in the preceding chapters. The final packet classification test will measure machine learning models' performance over time.

The chapter will then transition to semi-supervised clustering and classification analysis. First, clusters will be formed from the unlabeled stream dataset, then labels in the form of user behaviors will be assigned. Next, cluster centers will be examined for accuracy when propagating behaviors across streams in the dataset. Finally, the semi-supervised algorithm will classify future streams.

4.1 Packet Classification Testing

Testing models on packet traffic involved training on the July 14 network traffic, then measuring performance from future time windows to evaluate the fit. For the remainder of the section, validation data refers specifically to packets sampled from July 15 unless otherwise specified. This validation dataset contains five million sequential packets starting at 2:00 P.M. on July 15; testing on packets from July 16 through July 20 involved sampling a similar number of packets at the same time each day to reduce the potential variability in the evaluation.

The RFC was initially trained and evaluated using 260,269 network traffic packets from July 14. The Hamming system's memory limits constrained the training dataset size. During the training process, the RFC automatically generates a list of features and their importance for classification. This list of feature importance will enhance our understanding of feature significance for classification.

The rest of the packet classification testing involves the two deep learning model types, the MLP and CNN. These models will train on a dataset containing 31,152,220 sampled packets from 10 A.M. to 5 P.M. on July 14. The validation dataset from July 15 is the same as the RFC. Testing will begin by examining the insignificant features and removing the least important ones until achieving the best validation metrics. This feature removal will be conducted separately for both the MLP and the CNN. Once these features are determined, the follow-on tests will continue with those features removed.

Next, for each deep learning model, multiple data imbalance handling techniques will be tested. Using the same training and validation datasets, the MLP and CNN will integrate one of the three imbalanced handling techniques; RUS, class weights, and transfer learning. As described in Subsection 2.4.3, RUS is the process by which the majority class samples are removed, reducing the importance of the majority class. For testing, the dataset removed packets from the majority classes until all classes had the same membership numbers. Class weights change the weight of output neurons when computing the loss during training. This change allows different classes to hold different importance during the training evolution. For imbalanced traffic, these weights allow minority classes to have greater significance to the algorithm and do not change the dataset. By only affecting the loss function, there is minimal increase in computational load or pre-processing actions. This research assigns class weights through the previously mentioned Equation 2.3. Transfer learning involves pre-training the model using one dataset, then re-training with another. For this research, iterative training was used, with an initial training using the RUS dataset and re-training with the entire dataset with class weights applied.

All models will then measure their performance against each other to determine which model performed the best. The most critical responsibility for network traffic analysis is accurately identifying blocked traffic since it is the only traffic class threatening network security. Therefore, the recall score of blocked traffic classes is the most critical metric for comparing different models. While correctly classifying allowed traffic is crucial, misclassifying allowed traffic as blocked causes fewer issues from a security viewpoint. Thus, F1-score for allowed classes is also essential for evaluating the general fit and correctness of the model.

The final tests involve a comparison of the model performance over time. Since packet

classification models are not continuously learning, performance degradation as time progresses after training exists. This research will evaluate each day with five million packet datasets from the training day on July 14 through July 20. This evaluation will generate the performance of the RFC, MLP, and CNN models over a week and understand how each model performs after training. For this research, the F1-scores will be analyzed for each day since training occurred.

4.2 Packet Classification Results Analysis

4.2.1 Random Forest Results

After fitting the RFC to the abridged training dataset, Table 4.1 displays the feature importance values. These values are scaled differently than correlation values. However, the order of features is comparable to the previous correlation data found in Figure 3.2. Upon examination, there are key differences between the two lists. The *timestamp* feature ranked low for both lists. Additionally, the *ip_protocol* feature was ranked the second least important feature between the two lists. Generally, the geolocation features maintained their importance. Country names were far less critical to the RFC when classifying packet traffic.

Table 4.1. RFC Feature Importance.

Feature	Gini Value
ip_len	0.30
dst_ASN	0.11
port_dst	0.11
src_ASN	0.10
dst_LONG	0.07
src_LONG	0.06
timestamp	0.03
dst_COUNTRY	0.03
src_COUNTRY	0.02
ip_protocol	0.01

Next, the RFC performance was tested on the validation dataset. As shown in Table 4.2, the evaluation describes the key performance metrics for each class. Allowed traffic classes had worse performance across all metrics when compared with blocked classes. While the *incoming blocked* class had the largest membership, as shown by Table3.4, it did not attain the greatest performance. The *outgoing blocked* class, the smallest of the four, had the highest performance measurements in all metrics. The results suggest that this may be caused by the presence of features in the data that strongly indicate traffic flow; experiments show that models performed better on *outgoing* traffic classes compared to *incoming* classes with respect to their *allowed* and *blocked* counterparts.

Table 4.2. RFC Validation Metrics.

Class	Precision	Recall	F1-Score
Outgoing Blocked	0.991	0.984	0.988
Incoming Blocked	0.877	0.908	0.893
Outgoing Allowed	0.766	0.744	0.755
Incoming Allowed	0.654	0.547	0.596

4.2.2 MLP Results

Initial results for the MLP were attained with no adjustments to feature selection and utilized the hyperparameters described in Chapter 3. Figure 4.2 shows the precision-recall curve for each class of the evaluated validation dataset. This figure illustrates that the model with all features included only produces coherent predictions for the *incoming blocked* class while failing to produce predictions for all other classes. The *incoming blocked* class happens to be the class with the highest membership in this case.

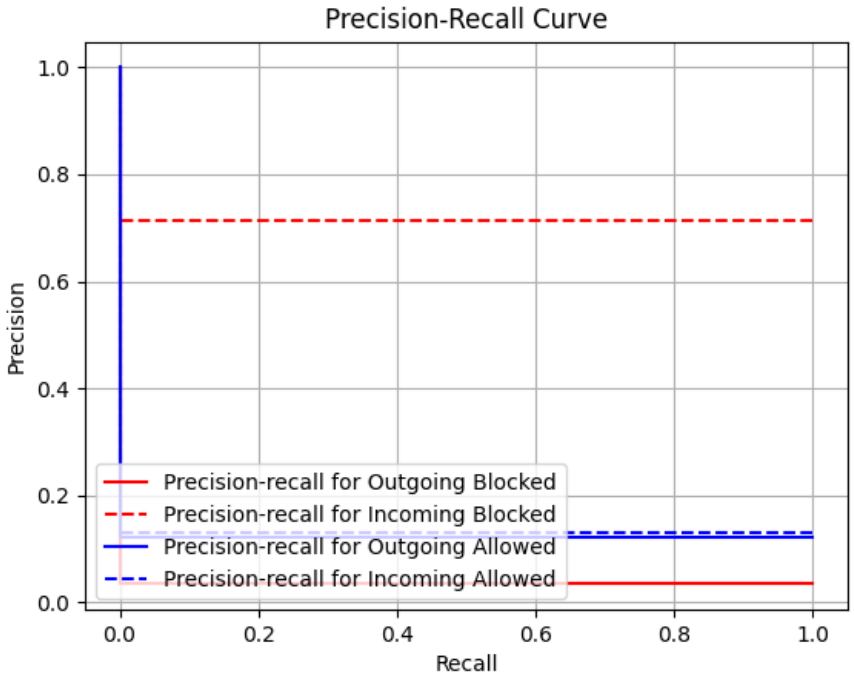

Figure 4.1. MLP Precision-Recall Curve With All Features Included

As mentioned in Subsection 4.2.1, some features are insignificant for classification. Therefore, least important features such as *timestamp* and *ip_protocol* were removed in an attempt to reduce the noise in the data. Figure 4.2 shows the resulting precision-recall curve for the MLP model with *timestamp* features removed. The results indicate that the MLP performs significantly worse on classifying allowed traffic compared to blocked traffic with at least a 0.2 difference in precision, for allowed, while recall values were greater than 0.5. The blocked traffic classes had a recall score greater than 0.93 with a precision greater than or equal to 0.8, indicating high sensitivity to blocked traffic.

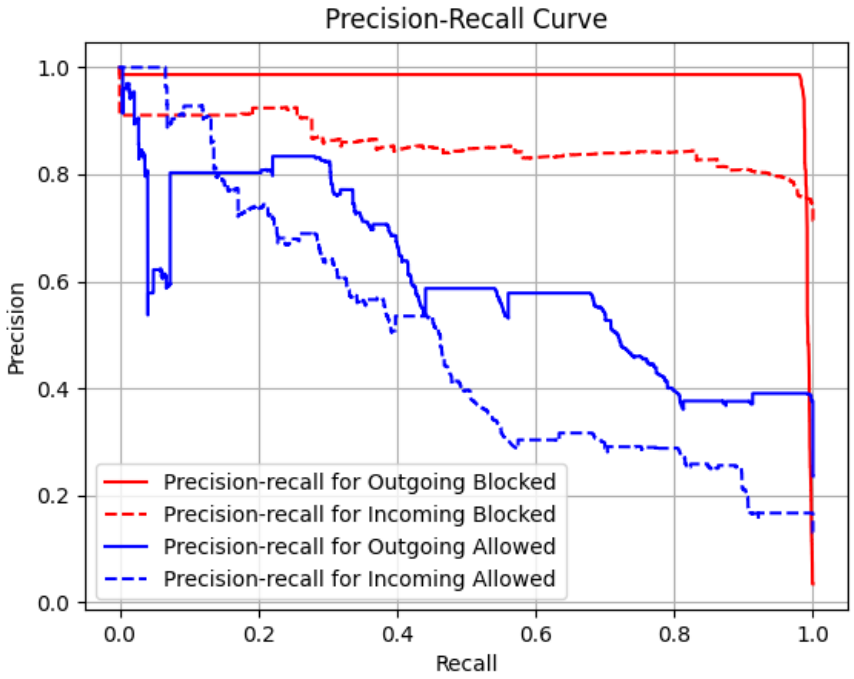

Figure 4.2. MLP Precision-Recall Curve Without Timestamp Feature.

The next feature with the least significance, *ip_protocol*, is removed for reevaluation of the MLP model. The precision-recall curve for the new MLP model without the *timestamp* and *ip_protocol* features is shown in Figure 4.3. Compared to the previous iteration, this feature selection results in lower performance. the recall of the *incoming blocked* class decreased by 0.1 from 0.931 to 0.835, as did the *incoming allowed* class. The only increase in performance is in the *outgoing blocked* class, which had an increase in precision to 0.999 from a previous 0.983. From these results, it is clear that removing one or both features increases model performance. However, *timestamp* feature alone produced the greatest meaningful results in reducing the noise of the dataset. Future tests with the MLP will keep the *timestamp* feature removed during evaluation.

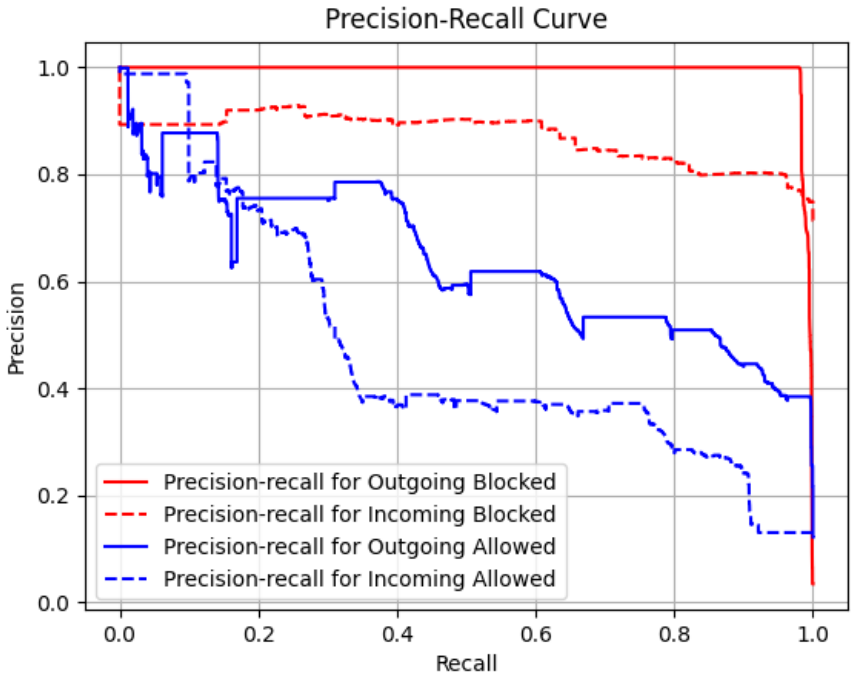

Figure 4.3. MLP Precision-Recall Without Timestamp or IP Protocol Features.

4.2.3 CNN Results

Initial validation of the CNN occurred similarly to the MLP. The initial CNN model used all input features and the same hyperparameters to evaluate the validation dataset. Figure 4.4 shows the class metrics. As with the MLP, the CNN only predicts the *incoming blocked* class of traffic, the largest class. Therefore, the feature space requires reduction.

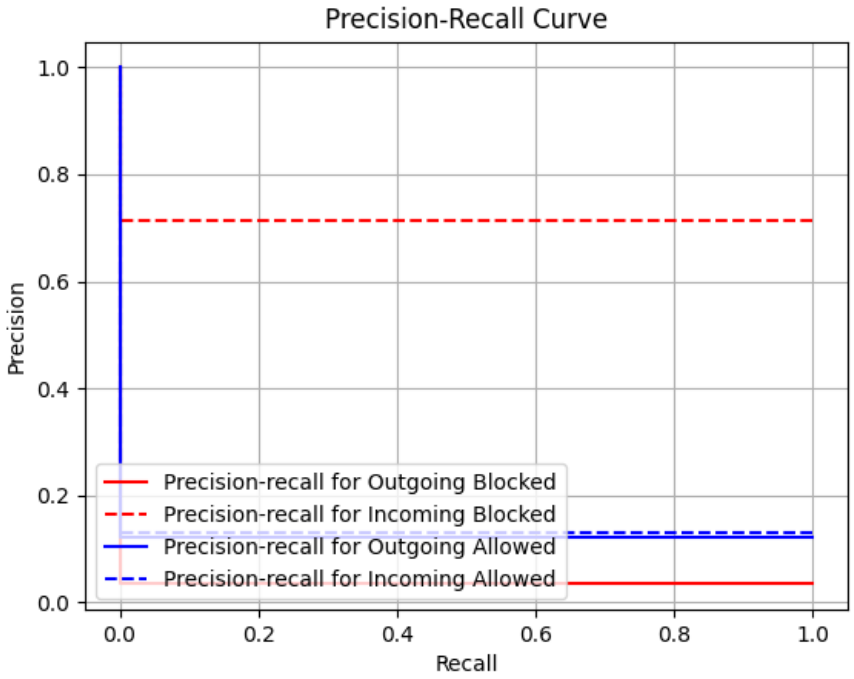

Figure 4.4. CNN Precision-Recall Curve.

Since the feature importance lists are the same for all classification models, the *timestamp* and *ip_protocol* features require noise analysis. The validation precision-recall curves can be found in Figure 4.5 for the CNN without the *timestamp*, and Figure 4.3 for the CNN without the *timestamp* and *ip_protocol* features. Removing one or both features allowed the CNN to make predictions across all traffic classes. While removing the *ip_protocol* in addition to the *timestamp* increases the recall of blocked classes by 0.9 for *outgoing blockedlocked* and 0.2 for *incoming blocked*, it comes at the cost of poor performance in allowed classes. Without the *ip_protocol* feature, the allowed traffic classes have less than 0.004 F1-scores. With only the *timestamp* removed, allowed traffic has increased performance to 0.270 F1-score for *outgoing allowed* and .391 for *incoming allowed*. It is clear that simply removing features from the CNN input does not increase the performance across all classes as it did with the MLP.

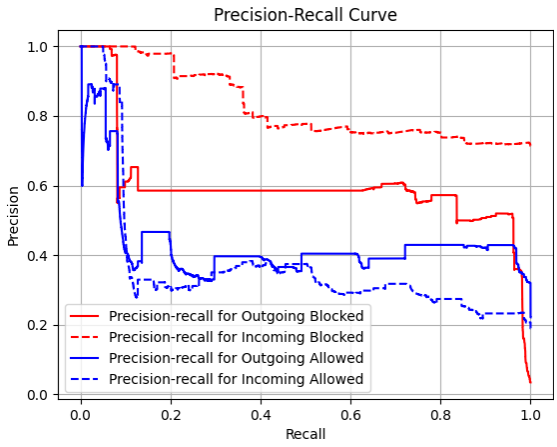

Figure 4.5. Timestamp Removed.

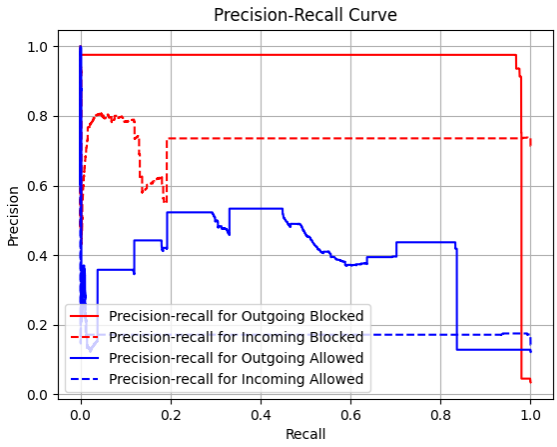

Figure 4.6. Timestamp and IP Protocol Removed.

4.2.4 Imbalanced Data Handling Technique Analysis

While all deep learning models can classify traffic across all classes, there is still a disparity between the performance of blocked classes versus that of the allowed ones. As the *incoming blocked* class makes up the majority of the data, there is a need to either increase the importance of minority classes or reduce that of the majority classes. Therefore, the MLP and CNN models are tested across several common data handling techniques to determine the potential for performance improvement. The testing will begin with RUS, followed by class weights, and transfer learning to incorporate both into a final technique.

RUS Results

RUS is the first imbalance handling technique that the validation dataset evaluates. Figure 4.7 for the MLP and Figure 4.8 for the CNN display their respective model class metrics. In the MLP, the recall scores of the *outgoing blocked*, *incoming allowed*, and *outgoing allowed* all increased, with allowed classes of traffic improving by at least 0.4. However, the performance of the *incoming blocked* class drops significantly, with a reduction in the recall by 0.6. The performance loss indicates that while the smaller classes improved, the largest class did not. This trend is in line with expectations, as RUS reduces the population

of the majority class samples to increase the importance of the minority classes.

The CNN had drastically different shifts in performance. The poor recall score of 0.065 for the *outgoing blocked* class increased drastically to 0.980. The *outgoing allowed* and *incoming allowed* classes also improved in their F1-scores, which increased by 0.06 and 0.05, respectively. However, the *incoming blocked* class suffered a 0.57 drop in recall score using RUS. Compared to the initial CNN metrics, RUS increases the performance of all minority classes. However, the *incoming blocked* class performs significantly worse.

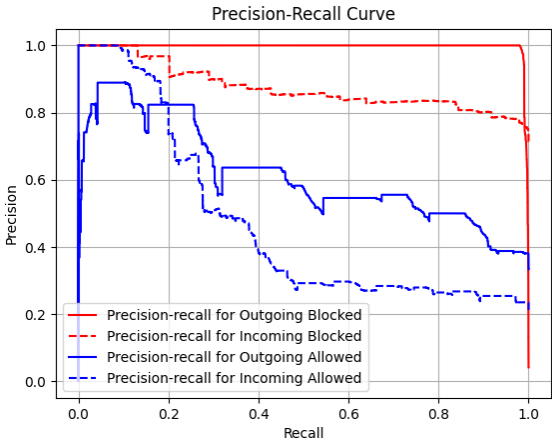

Figure 4.7. MLP with RUS.

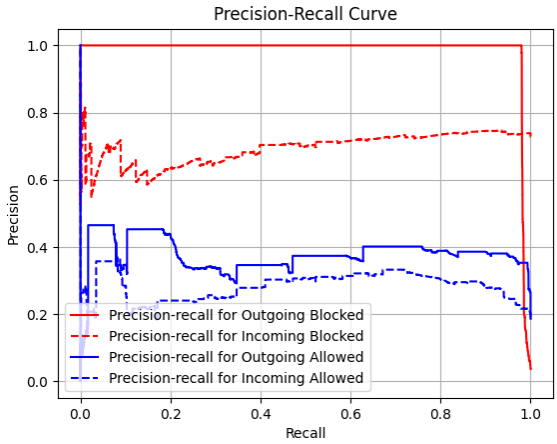

Figure 4.8. CNN with RUS.

Class Weight Results

The implementation of class weights for the MLP and CNN can be visualized in Figure 4.9 and Figure 4.10 respectively. Beginning with the MLP, the *outgoing blocked* class performs nearly as well with class weights, with a drop in the F1-score of 0.003. The *incoming blocked* class dropped noticeably, with the recall score dropping 0.39. The *outgoing allowed* class experienced a significant increase in recall, from 0.389 to 0.857, while the *incoming allowed* class also increased recall from 0.327 to 0.655. From these results, it is clear that class weights increase the performance for allowed classes but experience a drop in performance for blocked classes.

The CNN displayed the opposite results, with an increase in most classes but a decrease in *incoming allowed* class performance. The *outgoing blocked* and *incoming blocked* classes both had large increases in recall scores of 0.55 and 0.12, respectively. *Outgoing_allowed* increased in overall F1-score by 0.23 as well. However, the *incoming allowed* class became significantly worse, with a recall score of nearly zero. This performance shows that while the CNN has increased recall scores across most classes, it no longer predicts *incoming allowed* traffic.

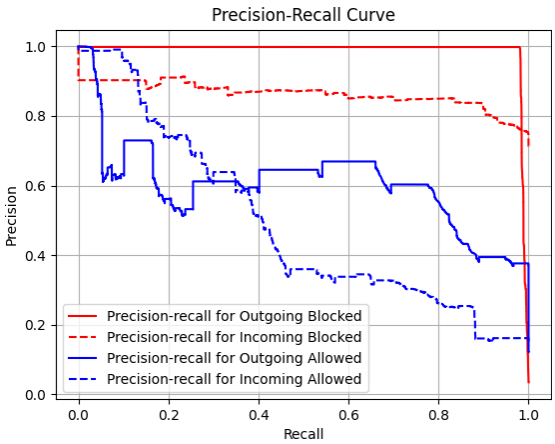

Figure 4.9. MLP With Class Weights.

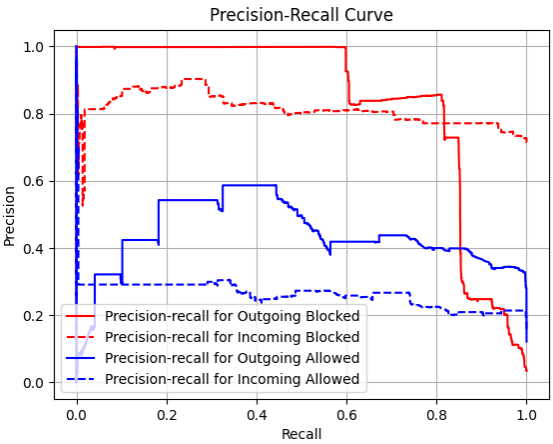

Figure 4.10. CNN With Class Weights.

Transfer Learning Results

Transfer learning combines pre-training using RUS with re-training using class weights. Figure 4.11 shows the precision-recall curve for the MLP performance using transfer learning. Comparing the transfer learning MLP to the initial MLP, there is improvement across the allowed traffic classes, but the *incoming blocked* class suffered a decreased recall score. The *outgoing blocked* class had little change, with a slight increase of 0.002 to the recall score. The *outgoing allowed* and *incoming allowed* both had increased recall scores but decreased precision scores. However, with recall scores of 0.77 for both allowed classes, the overall performance is higher with transfer learning. The *incoming blocked* class suffered a drop of 0.45 in the recall, with a new score of 0.484. While improving other classes in the

MLP, transfer learning still negatively impacts the majority class of traffic.

Transfer learning with the CNN generally improves performance across three of the four classes, with *incoming allowed* decreasing in all metrics as shown by Figure 4.12. The *outgoing blocked*, *incoming blocked*, and *outgoing allowed* classes all improved in their F1-scores, to 0.909, 0.812, and 0.486 respectively. The largest increase was in the *outgoing blocked* class, which increased from 0.06 recall to 0.98. However, the *incoming allowed* decreased in recall score from 0.391 to 0.134. For the CNN, transfer learning seems to have recall and F1-score metrics between RUS and class weight techniques.

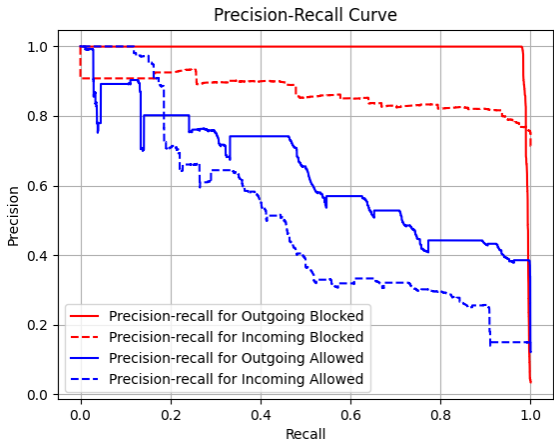

Figure 4.11. MLP With Transfer Learning.

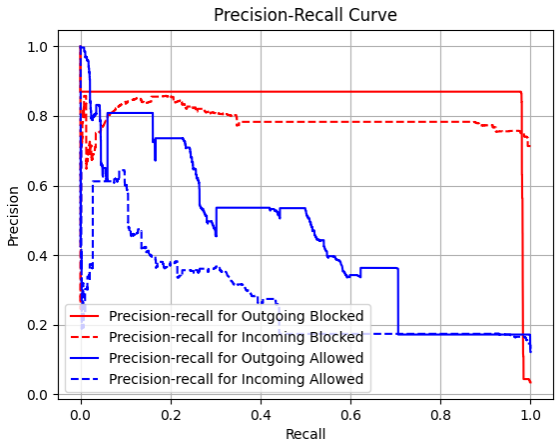

Figure 4.12. CNN With Transfer Learning.

4.2.5 Model Comparison Analysis

Measuring MLP effectiveness in network traffic classification requires comparing all models against relevant metrics. These model performance evaluations are pictured in Table 4.3. The bold number in the table indicates the highest value for each traffic class across all tested models. There appears to be a mutual exclusion between models with high recall scores for blocked traffic and those with high F1-scores for allowed classes. Specifically, without imbalanced techniques, the initial model had the best recall scores, over 0.9, for both blocked classes. While the class weights and transfer learning models had better F1-scores

for allowed classes, these same models had recall scores below 0.6 for the *incoming blocked* class. The highest performing class, therefore, is the initial MLP model without imbalanced data techniques, with the highest recall for blocked classes and comparable F1-scores for allowed traffic classes.

Table 4.3. MLP Model Metric Comparisons.

	Recall Scores		F1-Scores	
Imbalance Technique	Outgoing Blocked	Incoming Blocked	Outgoing Allowed	Incoming Allowed
None	0.980	**0.931**	0.498	0.424
RUS	**0.985**	0.324	0.560	0.390
Class Weights	0.980	0.549	**0.575**	0.426
Transfer Learning	0.982	0.484	0.535	**0.434**

The comparisons between different CNN models with and without imbalanced handling techniques are next. The relevant metrics are shown in Table 4.4. Unlike the MLP model type, the initial CNN model did not perform nearly as well without imbalanced dataset handling techniques, with less than 0.1 recall score for the *outgoing blocked* class. Only the RUS and transfer learning techniques had recall and F1-scores above 0.2 for all classes. However, transfer learning on the CNN provided both recall scores for blocked traffic above 0.85. Moreover, the *outgoing allowed* F1-score for transfer learning was only 0.004 less than the highest model, with an average F1-score for *incoming allowed*. Therefore, the CNN model with the transfer learning imbalanced handling technique was the best-performing CNN model.

Table 4.4. CNN Model Metric Comparisons.

	Recall Scores		F1-Scores	
Imbalance Technique	Outgoing Blocked	Incoming Blocked	Outgoing Allowed	Incoming Allowed
None	0.065	0.795	0.270	**0.391**
RUS	0.980	0.226	0.277	0.342
Class Weights	0.560	**0.913**	**0.490**	0.001
Transfer Learning	**0.981**	0.857	0.486	0.208

With best performing models identified for each model type, the final metrics can draw final comparisons. Table 4.5 shows the final metric comparisons between the RFC, MLP, and CNN models. The MLP has no imbalance techniques, while the CNN has transfer learning. The CNN overall had the worst metrics compared to the other two model types. However, it was only slightly less so in certain classes. For example, the CNN *outgoing blocked* recall score was only 0.004 less than the maximum value. Additionally, it was 0.074 off from the maximum *incoming blocked* recall value. The MLP, however, had the maximum recall score for *incoming blocked* class by 0.23. Furthermore, *incoming blocked* had the most significant class membership, an important consideration. However, the MLP was over 0.15 below the F1-scores for allowed traffic classes compared to the RFC. The RFC performed better in most classes of traffic. Overall, deep learning models performed similarly to the RFC in identifying blocked, malicious traffic but lacked performance in misidentifying allowed traffic. Thus, we conclude that the best performing model for packet classification was the Random Forests model.

Table 4.5. Overall Model Comparisons.

	Recall Scores		F1-Scores	
Model Type	Outgoing Blocked	Incoming Blocked	Outgoing Allowed	Incoming Allowed
RFC	**0.984**	0.908	**0.755**	**0.596**
MLP	0.981	**0.931**	0.498	0.424
CNN	0.981	0.857	0.486	0.208

4.2.6 Performance Degradation Analysis

When analyzing model performance, a critical consideration is performance over time. Real-world enterprise networks are constantly operating with little downtime. This continuous operation requires network edge security to maintain performance over time since training new models requires significant amounts of data. The last test to be analyzed is the examination of the models in comparison with number of days after initial training time. While accuracy, as mentioned previously, is not the optimal metric to measure network traffic classification, it is helpful for understanding model performance comparisons. As shown in Figure 4.13, the accuracy was measured each day since the training on July 14. At the end of the week, on July 20, the models are all less accurate due to dynamic network conditions. The only exception is a significant fluctuation in accuracy on July 18, where the MLP and CNN dip below 0.55 accuracy. The RFC retains the best performance over time, while the MLP is, on average, 7% less accurate. The CNN has the smallest deviation of all models, but performance is generally worse each day.

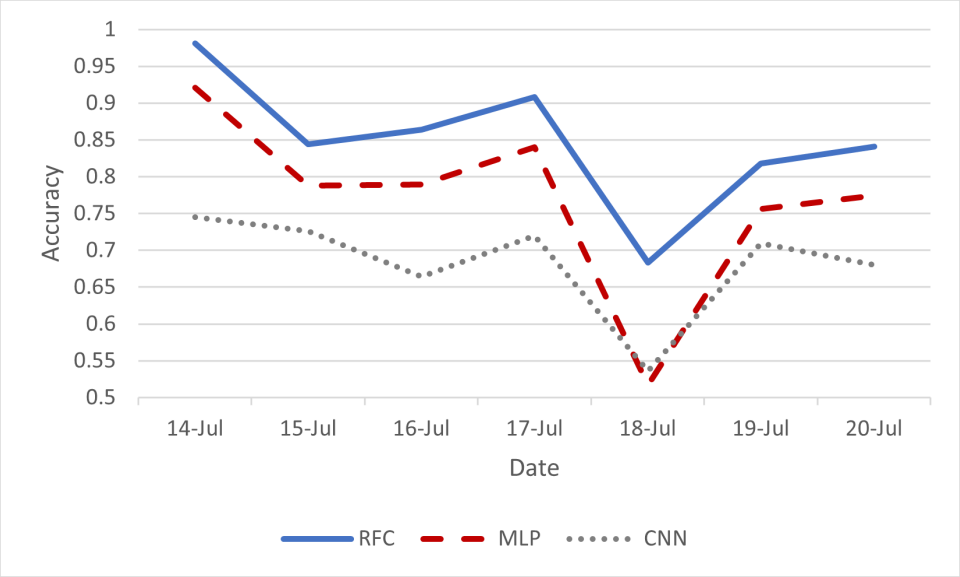

Figure 4.13. Model Accuracy Over Time.

The performance for each class over time can be visualized for the RFC in Figure 4.14, the MLP in Figure 4.15, and the CNN in 4.16. Across all models allowed, classes see significant degradation over time. The *outgoing blocked* class also sees some degradation, but less so than the allowed classes. The only class that rarely drops is the *outgoing blocked* class, which is the smallest class of traffic. Of note, by the end of the week, the RFC blocked classes of traffic both had less F1-score than the MLP and the CNN, indicating the deep learning models began outperforming the RFC six days after training.

Figure 4.14. RFC F1-Scores Over Time.

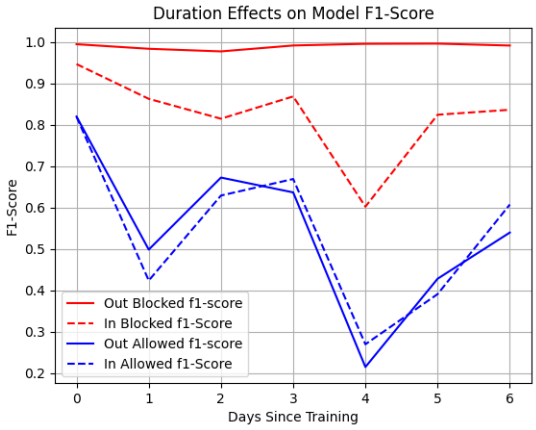

Figure 4.15. MLP F1-Scores Over Time.

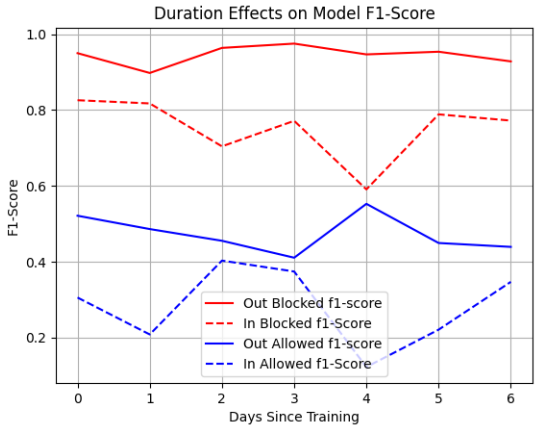

Figure 4.16. CNN F1-Scores Over Time.

4.3 User Profiling Analysis

We now discuss user behavior profiling through clustering and semi-supervised learning. As the stream dataset is unlabeled, the first task is to apply semi-supervised learning to learn the labels over the dataset. First, the clustering algorithm will be applied on packet streams. Then, the mini-batch K-means clustering algorithm will be used to fit 420,000 streams from July 14. Unfortunately, the Hamming system constrained the memory limits of the datasets

and could not handle anything larger due to each stream containing over 1,000 features. Each stream was examined through geolocation information, aggregated statistical features of the stream, and an array of each packet's direction within the stream. The clustering algorithm was used to generate 25 clusters over the dataset.

Next, cluster centers assist in the identification of stream behaviors. Ideally, the cluster center exhibits the representative stream for all other streams in that cluster. Therefore, we may manually label the cluster centers, then extend those labels to all other streams according to their respective cluster. To do this, we generate URLs from the cluster center IP address. We also generate URLs for the five streams that are nearest to the cluster center in Euclidean distance. Overall, the algorithm creates 250 respective URLs over the dataset. Each cluster will be analyzed to determine the majority behavior for which labels will be assigned respectively. Any clusters without common behaviors will be labeled as *None* and dropped from future predictions, as they are considered non-behavioral forming.

The next step is to examine the accuracy of the cluster labels. Initial qualitative analysis will ensure the accuracy of these labels, with any poorly performing clusters being reclassified as *None* if the representative streams do not match the majority of the traffic.

The final testing of user behavior profiling is to determine whether future streams can be classified using existing labeled traffic. This semi-supervised classification approach uses the labels produced by the clustering algorithm to train a classifier to predict future behaviors. The validation dataset comprises 33,000 streams, compiled from July 14, with no repeated streams from the initial clustering dataset. Semi-supervised learning applicability to stream classification will be evaluated and validated by the performance of both the K-means algorithm and the classification model.

4.3.1 Clustering Model Analysis

Figure 4.17 shows a visual representation of the clusters, which maps the samples per cluster in a histogram form. For clarification, while the largest three clusters contain 60.5% of the total streams, the mean samples per cluster are 7,297. This population distribution indicates that while the 22 clusters each contain less than 10% of total streams, there are no outliers with less than 1,000 streams. This information shows us that each cluster has enough samples to remain functional.

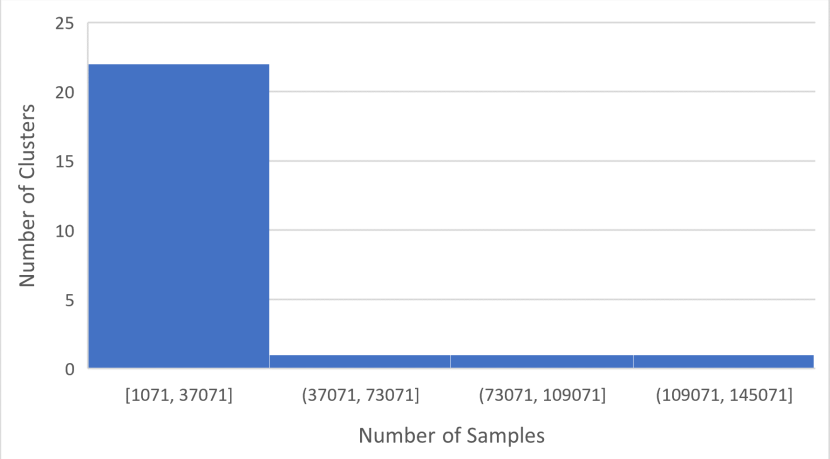

Figure 4.17. K-Means Cluster Sample Histogram.

4.3.2 Cluster Profiling Analysis

After the data is used to create and train the clusters, the clusters can then be assigned user behaviors based on their representative cluster centers and nearby samples relative to those cluster centers. A cluster holds a behavior if a simple majority for a single behavior exists across the cluster's representative samples. For our analysis, if three or more streams for a cluster are from a specific website genre, then we label all samples in that cluster according to the website genre. We say that these streams represent a specific user behavior. If no common behaviors exist for a cluster, then the cluster is labeled as *None*. These clusters imply no specific behaviors since there is a mix of behaviors within. Table 4.3.2 displays the identified behaviors from clusters. The most common URLs included Microsoft online products such as Outlook and Azure cloud computing resources. Thus, *Microsoft Products* encompasses all Microsoft applications into a single behavior. Next, content delivery networks (CDNs) were also present, such as Akamai, Facebook, and Comcast, which service multimedia and social media applications. These CDNs streams were grouped under the *Media* behavior. The *Cloud Computing* behavior designates Amazon Web Service (AWS), Fastly, and other cloud computing streams. Lastly, Google URLs are composed of two clusters, thus creating a *Google Search* behavior. While Google provides some media content, it composes all the

representative streams for clusters 1 and 20 and requires a unique behavior.

Table 4.6. Behavior Profiling of Clusters.

Behavior	Associated Clusters	Number of Clusters
Microsoft Products	0, 2, 7, 8, 9, 10, 11, 13, 15, 16, 18, 19, 21, 22, 23	15
Media	4, 5, 12, 24	4
Cloud Computing	3, 14	2
Google Search	1, 20	2
None	6, 17	2

4.3.3 Label Propagation Analysis

With clusters assigned to behaviors, the clusters can propagate their behaviors to all associated samples. Figure 4.18 depicts the accuracy of each cluster. An initial qualitative inspection was conducted on the propagation of behavior labels across the samples used for clustering to determine initial behavior accuracy. Each sample had its actual behaviors, resolved from URLs, compared against their cluster behavior. This inspection revealed a 50% accuracy for non-representative streams. Clusters other than *Microsoft Products* had 100% accuracy on initial inspection. However, many *Microsoft Products* clusters had very few correct labels. This poor accuracy indicates that while the representative streams were closest to the cluster center in feature space, they were of minority behaviors in the cluster. When examining the accuracy of each cluster, the largest six clusters, the leftmost in Figure 4.18, had accuracy at or over 0.9. These six clusters also make up 78% of the total number of streams, indicating that the overall accuracy for the entire dataset is higher than an equal weighting for each cluster. Therefore, clusters with performance less than 90% accuracy were re-classified to the *None* behavior. This issue of misrepresentation is a caveat of semi-supervised learning, where imperfect labels are assigned.

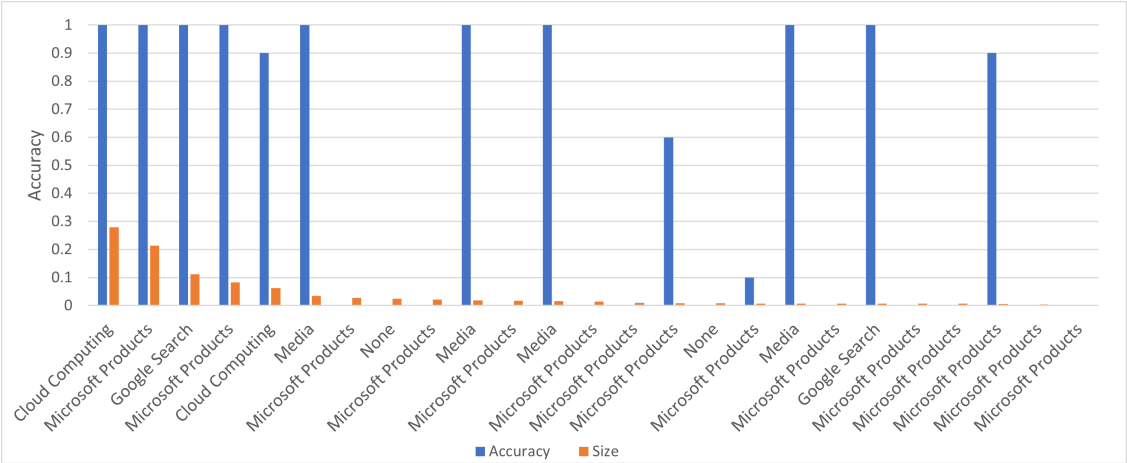

Figure 4.18. Cluster Label Accuracy By Decreasing Cluster Size.

4.3.4 Semi-Supervised Classification Analysis

This semi-supervised learning utilizes the K-means algorithm to predict the user behavior classes based on cluster labels. All labels with the *None* behavior are dropped prior to training, as this research focuses on classifying the identified, known behaviors previously deduced in Subsection 4.3.2. The evaluation metrics are displayed in Table 4.7. The K-means model had an overall classification accuracy of 0.97. The largest two classes, *Cloud Computing* and *Microsoft Products* had 0.97 and above F1-score, indicating high-performance levels. The smallest class, *Media*, had the worst performance, with a lower recall score of 0.72. This disparity in recall scores indicates a potential imbalance issue within the data. However, the K-means clustering model can accurately predict future traffic classes.

Table 4.7. K-Means Semi-Supervised Classification Metrics.

User Behavior	Precision	Recall	F1-Score	Samples
Cloud Computing	1.00	1.00	1.00	13,536
Google Search	0.93	1.00	0.96	4,485
Media	1.00	0.72	0.83	3,914
Microsoft Products	0.94	1.00	0.97	11,380

4.4 Chapter Summary

This chapter analyzed the test results for packet classification and user behavior identification. Both packet classification and packet stream analysis emulated edge security systems and better understood network traffic patterns. For packet classification, different model types were evaluated for use in network traffic analysis and different techniques for handling imbalanced traffic classes commonly seen in real-world networks. After training, these models were examined over time to understand model performance degradation better. Next, the clustering algorithm performed user behavior classification, and clusters were examined and assigned user behaviors. These behaviors propagated labels through the data, with new traffic drawing user behaviors from associated clusters. In the following chapter, the results are discussed, limitations are examined, and future work is posed.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Conclusion and Future Work

In this chapter, the research hypothesis will be examined for confirmation. Then, each research question will be revisited and answered. Next, limitations and encountered constraints during the research process are addressed, and the impact those limitations had on the results. Lastly, potential advances in future work from this research are addressed.

5.1 Discussion

From the results in Chapter 4, this research demonstrates the ability of deep learning techniques to emulate firewalls and classify packet traffic. The deep learning models and RFC achieved over 0.85 recall scores when identifying malicious network traffic packets. While the allowed traffic classes were misclassified more often, the RFC and MLP both had over 0.4 F1-scores. Imbalanced data handling techniques are also demonstrated to mirror the natural imbalance of traffic classes in real-world networks. While RUS and class weights alone did not increase the overall performance of models, transfer learning did with the CNN. The RFC and MLP had the best performance when comparing the different models. The RFC had the highest average metrics across all classes, but the MLP had the highest recall score for the largest blocked traffic class. Moreover, we have demonstrated this prediction ability over seven days, from July 14 through July 20, 2022.

This research demonstrated that machine learning techniques could successfully profile user behaviors. Common user behaviors were defined and associated with clusters using clustering algorithms and representative samples. These clusters were then adjusted from an initial qualitative inspection for accuracy by removing those clusters whose representative samples did not indicate a majority behavior. Moreover, the clustering model successfully predicted future packet stream behaviors through this semi-supervised classification. With a minimum class F1-score of 0.83, the model achieved a high level of performance, with better results for larger classes. Likely, the lower scores indicate data imbalance issues, which further research could explore.

The models and techniques in this research were implemented on enterprise network traffic

and are generalizable to similar problems. All traffic was taken directly from the NPS ERN without dropping any significant percentage of traffic. While likely adding additional noise to the datasets, this real-world traffic provided a proof of concept for deep learning and machine learning solutions in classifying packets and profiling streams. Additionally, the data pre-processing is generalizable across different enterprise networks, as the model input varies according to the network traffic diversity.

This research confirms that PCAP records can classify network traffic and profile user behaviors through machine learning within large-scale networks. Moreover, the previous paragraphs have answered each research question. Furthermore, packet classification is feasible through machine learning and deep neural networks, and semi-supervised classification can accurately predict stream behaviors. While there is certainly room for improvement in model performance, this research serves as proof of concept for the feasibility of such implementation. In this regard, the research succeeds in its proposed goals.

5.2 Limitations

A significant constraint to the research was the question of packet label accuracy. The dataset collections were refined throughout the research to produce the best-guessed packet labels. However, the two NICs encountered various problems that may reduce the accuracy of the dataset. For example, some outgoing packets were captured on the outside NIC, which meant that the interior NIC did not capture the packet. The solution was to add additional truth labels such as *incoming dropped outside* or *exterior only*. Due to the sizeable dataset, there was no feasible method of conducting deep packet inspection for highly accurate truth values. The use of NICs remained the only viable method of identifying packets. As the data is further refined and packet labels reclassified, the performance of machine learning models could increase. The potential for noise and misclassified packets to exist within the dataset is likely and will continue to be iteratively improved over time.

Another constraint to the research was the dataset size regarding training and validation time and memory constraints. While performing at a significantly higher level than most consumer deep learning hardware, the Hamming system still hindered the ability to conduct ideal training on enterprise data. For packet classification, the ability to train a model across every packet on July 14th could produce better results than training on a 31 million packet

cross-section. Additionally, larger validation datasets may decrease sharp dips and spikes in performance when evaluating models. For specific models, such as random forests and K-means, there are no built-in methods to accept chunks of files in sequence, such as a Python generator. This memory requirement limits the training dataset to these models by the hardware memory limits for the process. A reduction in training time for packet classification and an increase in the size of the K-means and random forest training data would likely increase performance in this research.

5.3 Future Work

There is clear potential for future work in improving machine learning model performance in packet classification. All machine learning models had a worse performance for allowed traffic classes over blocked, which could improve through several methods. Testing other machine learning and deep learning models may illuminate a better model fit for network traffic classification. Using Bayesian learning to adjust hyperparameters could also improve deep learning model performance. For example, the piece-wise learning rate decay may have been too aggressive in reducing the learning rate near the optimal loss value. Testing a variety of learning rates or only reducing the learning rates after several epochs of no improvement may increase deep learning model performance.

User behavioral profiling has several different potential approaches for continued research. Utilizing majority voting of stream behaviors to classify a cluster is a novel approach that may improve clustering performance. Further sub-classification of clusters is outside the scope of this research but may allow for the classification of mixed behavior clusters. In addition, the accuracy of clusters would likely improve future stream predictions. Using deep learning to conduct semi-supervised classification is another approach to predicting future traffic behaviors. Finally, if future datasets include stream information for blocked content, the definition and identification of blocked content could extend this research further.

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] L. T. C. Linn, "Marines in the Age of Artificial Intelligence," *Marine Corps Gazette*, no. February 2020, Feb. 2020.

[2] D. H. Berger, "Force Design 2030: Annual Update," May 2022.

[3] R. Sommer and V. Paxson. (2010). Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. Oakland, CA, USA. [Online]. pp. 305–316. Available: http://ieeexplore.ieee.org/document/5504793/

[4] M. H. Haghighat, Z. A. Foroushani, and J. Li. (2019, Oct.). SAWANT: Smart Window Based Anomaly Detection Using Netflow Traffic. Xi'an, China. [Online]. Available: https://ieeexplore.ieee.org/document/8947103/

[5] S. Rezaei and X. Liu. (2020, May). How to Achieve High Classification Accuracy with Just a Few Labels: A Semi-supervised Approach Using Sampled Packets. *arXiv:1812.09761 [cs]*. [Online]. Available: http://arxiv.org/abs/1812.09761. ArXiv: 1812.09761.

[6] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret. (2017). Network Traffic Classifier With Convolutional and Recurrent Neural Networks for Internet of Things. *IEEE Access*. [Online]. Available: https://ieeexplore.ieee.org/document/8026581/

[7] A. R. Mohammed, S. A. Mohammed, and S. Shirmohammadi. (2019, July). Machine Learning and Deep Learning Based Traffic Classification and Prediction in Software Defined Networking. Catania, Italy. [Online]. Available: https://ieeexplore.ieee.org/document/8805044/

[8] N. Chockwanich and V. Visoottiviseth. (2019, Feb.). Intrusion Detection by Deep Learning with TensorFlow. PyeongChang Kwangwoon_Do, Korea (South). [Online]. Available: https://ieeexplore.ieee.org/document/8701969/

[9] A. Kipane. (2019). Meaning of profiling of cybercriminals in the security context. *SHS Web of Conferences*. [Online]. Available: https://www.shs-conferences.org/10.1051/shsconf/20196801009

[10] C. I. Eke, A. A. Norman, L. Shuib, and H. F. Nweke. (2019). A Survey of User Profiling: State-of-the-Art, Challenges, and Solutions. *IEEE Access*. [Online]. Available: https://ieeexplore.ieee.org/document/8851141/

[11] R. Vinayakumar, K. P. Soman, and P. Poornachandran. (2017, Sep.). Applying convolutional neural network for network intrusion detection. Udupi. [Online]. Available: http://ieeexplore.ieee.org/document/8126009/

[12] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. Sebastopol, CA: O'Reilly Media Inc., 2017.

[13] J.-A. Zhu, Y. Jia, J. Lei, and Z. Liu. (2021, Nov.). Deep Learning Approach to Mechanical Property Prediction of Single-Network Hydrogel. *Mathematics*. [Online]. *9*(21). Available: https://www.mdpi.com/2227-7390/9/21/2804

[14] D. Kwon, K. Natarajan, S. C. Suh, H. Kim, and J. Kim. (2018, July). An Empirical Study on Network Anomaly Detection Using Convolutional Neural Networks. Vienna. [Online]. Available: https://ieeexplore.ieee.org/document/8416441/

[15] R.-H. Hwang, M.-C. Peng, C.-W. Huang, P.-C. Lin, and V.-L. Nguyen. (2020). An Unsupervised Deep Learning Model for Early Network Traffic Anomaly Detection. *IEEE Access*. [Online]. Available: https://ieeexplore.ieee.org/document/8990084/

[16] A. Shahraki, M. Abbasi, A. Taherkordi, and M. Kaosar. (2021, Aug.). Internet Traffic Classification Using an Ensemble of Deep Convolutional Neural Networks. Virtual Event USA. [Online]. Available: https://dl.acm.org/doi/10.1145/3472735.3473386

[17] Wei Wang, Ming Zhu, Xuewen Zeng, Xiaozhou Ye, and Yiqiang Sheng. (2017). Malware traffic classification using convolutional neural network for representation learning. Da Nang, Vietnam. [Online]. Available: http://ieeexplore.ieee.org/document/7899588/

[18] J. An and S. Cho, "Variational Autoencoder based Anomaly Detection using Reconstruction Probability," SNU Data Mining Center, Dec. 2015.

[19] Z. Fan and R. Liu. (2017, Aug.). Investigation of machine learning based network traffic classification. Bologna. [Online]. Available: http://ieeexplore.ieee.org/document/8108090/

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[21] J. M. Johnson and T. M. Khoshgoftaar. (2019, Dec.). Survey on deep learning with class imbalance. *Journal of Big Data*. [Online]. *6*(1). Available: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0192-5

[22] High Performance Computing. [Online]. Available: https://wiki.nps.edu/display/HPC/A+Gentle+Introduction+to+Hamming

[23] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. [Online]. Available: https://www.tensorflow.org/

[24] GeoIP2 and GeoLite2 Databases. [Online]. Available: https://dev.maxmind.com/geoip?lang=en

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California