Thesis Paper On

"Using Efficient Machine Learning Algorithms for Detection of Network Traffic and Load Management"

Submitted by

Shahadut Hossen

ID: 193-35-515

Department of Software Engineering

Daffodil International University

Supervised by

Mr. Esraq Humayun

Lecturer

Department of Software Engineering

Daffodil International University

This Thesis paper has been submitted in fulfillment of the requirements for the Degree of Bachelor of Science in Software Engineering.

APPROVAL

This thesis titled on **"Using Efficient Machine Learning Algorithms for Detection of Network Traffic and Load Management"**, submitted by **Student Name: Shahadut Hossen (ID: 193-35-515)** to the Department of Software Engineering, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Software Engineering and approval as to its style and contents.

BOARD OF EXAMINERS

Chairman

Dr. Engr. AKM Masum
Professor
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

Internal Examiner 1

Dr. Marzia Ahmed
Assistant Professor
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

Internal Examiner 2

Md. Rajib Mia
Lecturer
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

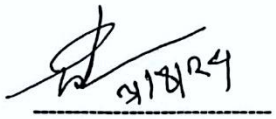
External Examiner

Dr. Md. Manowarul Islam
Associate Professor
Dept. of Computer Science and Engineering
Jagannath University

DECLARATION

I announce that I am rendering this study document under Mr. Esraq Humayun, Lecturer, Department of Software Engineering, Daffodil International University. I therefore, state that this work or any portion of it was not proposed here therefore for Bachelor's degree or any graduation.

Supervised By:



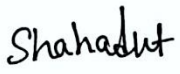
Mr. Esraq Humayun

Lecturer

Department of Software Engineering

Daffodil International University

Submitted By:



Shahadut Hossen

ID: 193-35-515

Department of Software Engineering

Daffodil International University

ACKNOWLEDGEMENT

First and foremost, I am deeply grateful to Almighty Allah for granting me the strength and perseverance to complete this thesis. I would also like to extend my heartfelt thanks to my supervisor, Mr. Esraq Humayun, Lecturer, Department of Software Engineering. His expert guidance, sincere advice, and unwavering encouragement have been invaluable throughout this process.

I would also like to express my gratitude to all the participants who took part in the survey for this thesis project. Their enthusiastic participation and input were crucial for the validation of this.

ABSTRACT

With the growth of modern networks in size and complexity, traffic identification and load balancing is becoming a more important issue. Traditional ways of load balancing, such as Round-Robin DNS would sometimes be insufficient because they do not consider that the different backend servers had different capacities and there are many types of network traffic. This thesis investigates the application of machine learning algorithms that can classify network traffic in real time to be integrated with weighted load balancing techniques.

Five machine learning algorithms, K-Nearest Neighbors, Decision Trees, Random Forest, XGboost and Neural Networks are used to classify network traffic into classes like benign traffic (normal) bulk data transfers (real-time applications which are latency sensitive) etc. We then assess these algorithms in enhancing the accuracy and efficiency of traffic classification, allowing for improved decision making on resource allocation.

This work also studies Weighted Load Balancing, a theoretical technique to balance blood across network using server capability so that no servers stay overloaded and other under-utilized. A set of these simulations are conducted on XGBoost, Random Forest and Neural Networks to determine the impact of intelligent load balancing strategies. While the load balancing component does not really exist, it gives a good idea of how traffic classification using machine learning could help improve all sorts of systems responsible for managing loads.

Results emerging from this study serve as guidance for utilizing machine learning to build network traffic management solutions that are more adaptive and effective. These results provide evidence to support the theoretical justifications for implementing such intelligent load balancing methods in large-scale network infrastructures.

Keywords: Network Traffic Classification, Machine Learning, Weighted Load Balancing, Real-Time Applications, XGBoost.

TABLE OF CONTENTS

Chapter 1: Introduction

1.1 Background

In recent years, we've witnessed a remarkable expansion of internet-based services, including cloud solutions, mobile apps, and the Internet of Things (IoT). This expansion has triggered a dramatic increase in data production, reaching levels previously unseen in terms of global traffic. This includes diverse sources such as video streaming, online gaming, remote work interfaces, and IoT outputs, which collectively enhance the complexity and volume of network traffic. As a result, the task of network management today transcends simple data volume handling; it necessitates maintaining the network's reliability, efficiency, and capacity for scaling. Central to this task is load balancing, which ensures equitable distribution of workload across servers to prevent overloading some while underutilizing others. Despite their prevalence and simplicity, traditional load balancing techniques like Round-Robin DNS often overlook the specific demands on the network and the varied capabilities of individual servers.

They allocate requests in a circular fashion to all servers equally, without consideration of where the traffic is overwhelmed or idle. Traditional load balancing does not take into account the location of network traffic. It also does not consider balancing traffic across servers of varying capacities. The data traffic is also not uniformly high-bandwidth and highly time-sensitive, such as streaming video and VoIP calls. A one-size-fits-all method of load balancing, such as the Round-Robin, fails to address annually varying traffic patterns to servers of differing capacities, especially in an age of pandemics. In this situation, machine learning offers promising possibilities for network administration. Machine learning is highly efficient at analyzing data in real time, distinguishing one type of network traffic from another, and potentially allocating network traffic more efficiently to CPU cores. For example, various types of network traffic can first be classified as benign, critical, bulk, or sensitive to latency.

In practice, traffic control could direct benign and bulk traffic to servers with higher capacities, while critical and latency-sensitive traffic might go to servers with lower capacities. This study explores the application of machine learning techniques to refine network performance by improving load balancing methods. Specifically, it employs machine learning algorithms like K-Nearest

Neighbours [KNN], Decision Trees, Neural Networks, Random Forest, and XGBoost for the classification of traffic types. Subsequently, a strategy of Weighted Load Balancing is used to more aptly allocate traffic across servers of varied capacities based on their respective capabilities.

1.2 Motivation of this research

Our work is motivated by two observed trends: the network infrastructure becomes ever more complex, and conventional load balancing techniques have limitations. Load Balancing is necessary when it comes to maintaining high levels of performance in distributed network environments, but traditional methods (e.g., Round Robin DNS) are not capable of fine-grain balancing considering server capabilities and specific aspects of the traffic. Servers, even within the same network, can have different processing power and capacity levels in a real-world scenario such as one server with much more CPU and memory resources than another. However, traditional load balancing methods still treat all servers the same, hence using up server resources inefficiently. This can cause one server to be underutilized with high capacity and another may get overloaded at low capacity, hence users experience slow responses, timeouts, connection droppings, or even service outage. This challenge is exponentially more complex as the scale and scope of network traffic increases.

The second most important factor is the nature of network traffic in its diversity. There are requirements for different types of traffic. VoIP & video conferencing can be some examples, which are critical real-time services that have tight latency limits and cannot tolerate any delays. Storage operations that do not involve user interaction can be delayed, such as bulk file transfers or background updates. In the absence of smart traffic classification and prioritization, load balancers could potentially wrongfully direct high-priority traffic to overutilized servers or allocate low-priority traffic systematically to underperforming servers, resulting in inefficient performance as well as wasted resources.

A viable solution to these issues is machine learning. This eliminates the requirement for fixed

traffic classification rules and enables network traffic to be immediately sorted based on predefined patterns using machine learning algorithms, thus allowing for resource allocation tailored to the type of network traffic. For instance, a content platform can opt to universally route money-making ad traffic around the high-cost routing paths and direct latency-sensitive types of user-generated content (UGC) received at those same PoPs through yet another set of potentially cheaper routes.

By utilizing machine learning-powered traffic classification alongside Weighted Load Balancing, you can drastically enhance network efficiency. It allows better use of server, improves service quality, and will make our architecture scalable. In this work, we aim to study the practical feasibility and performance of utilizing such an integrated paradigm, detailing further how it can be applied in a real-world network setting.

1.3 Research Objective

This study aims to implement and compare the machine learning-based traffic classification with advanced load-balancing algorithms. This work intends to address the limitation of classical load balancing though a new, adaptive, and intelligent traffic management in contrast to established techniques.

Research Key Objectives:

- Machine Learning Algorithms to Classify Network Traffic - K-Nearest Neighbors (KNN), Decision Tree, Random Forest, XGBoost and Neural Networks. These categories include benign traffic (or legitimate applications), bulk data, and real-time critical applications which will be used to identify the use of different algorithms.
- Simulate Round-Robin DNS: Round-Robin DNS will be the core strategy being used to spread incoming traffic across a group of servers, without considering server's capacity or its type of requests.
- Optimize the traffic distribution with respect to the server capacity and Implement Weighted Load Balancing. To accomplish this, weights will be established for each server

based on processing capacity with traffic being directed proportionately to these weight-values.

- Validate how well the integrated system can classify traffic and load-balance with quantitative. This performance will be evaluated with simulations and compared the machine learning-based classification as well Weighted Load Balancing to traditional Round-Robin DNS.

- Recommendation for Future Work: In this study, we have only investigated Machine learning methods and did not try to combine the best of other Deep learning model like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) with our proposed methods so still numerous areas warrant further investigation in real time such as models load balancing in dynamic systems etc.

1.4 Problem statement

As networks grow larger and more complex, managing traffic effectively becomes increasingly important. Traditional load-balancing methods like Round-Robin DNS may not be enough for this challenge. These techniques distribute traffic equally among servers, but they don't consider the individual capacity of each server or the specific demands of the traffic. While they may work in smaller, simpler setups, they can struggle with the varied and ever-changing traffic patterns seen in larger networks. Additionally, these older methods don't adjust for the unique requirements of different types of traffic, such as bandwidth, latency tolerance, or processing power.

This problem only gets worse as network traffic becomes more varied. For example, real-time, high-priority traffic such as video conferencing and VoIP needs to be fast and responsive. On the other hand, bulk data transfers, like downloading large files, aren't as bothered by a bit of delay.

Not adequately distinguishing these types of traffic can degrade network performance—vital services might experience delays or interruptions, and less critical traffic could unnecessarily hog bandwidth.

This research addresses a fundamental question: How can machine learning-based traffic classification be seamlessly integrated with load balancing strategies to enhance the distribution of network traffic across multiple servers? More specifically, it investigates whether machine learning algorithms can effectively classify traffic in real-time and if Weighted Load Balancing can boost the efficiency of traffic distribution by aligning it with server capacities.

1.5 Research Scope

This study aims to categorize network traffic using machine learning algorithms while optimizing distribution through load balancing. Specifically, the following will be addressed:

Application of Machine Learning: Machine learning methods will be employed to classify traffic into distinct types, allowing for dynamic prioritization and allocation of resources based on traffic characteristics and demands.

Load Distribution Tactics: Traditional Round-Robin DNS and Weighted Load Balancing will be simulated and contrasted. Round-Robin DNS will serve as a baseline, while Weighted Load Balancing will distribute traffic based on server capacities.

Simulation-based Examination: Simulated traffic and server environments will be used to assess the proposed system's effectiveness in traffic classification, distribution, and load management under varying conditions.

Real-world deployment is beyond the scope of this study. However, the findings may provide insights into how the suggested approaches could be implemented in actual networks. Future work may explore live network deployment, continuously tracking and dynamically adjusting server weights based on performance metrics such as CPU usage, available memory, and bandwidth

Chapter 2: Literature review

Ding and Zhang (2024) suggest new way to handle dynamic load balancing in fog computing using reinforcement learning (RL). Their approach tackles the challenge of efficiently distributing tasks in environments that are constantly changing. The RL algorithm learns from real-time feedback, helping it make better decisions about task allocation based on the current system conditions. As a result, this method improves system performance, cutting down user response times by 31.6% and reducing incorrect task assignments by 24.02% compared to older, static load balancing methods. The paper points out that RL is especially useful for handling workloads that are constantly changing and hard to predict, which static methods can't manage as well. RL helps the system adjust to these changes, making sure resources are used more efficiently and reducing delays. By reallocating tasks based on what's happening in real-time, this method makes the system more efficient and responsive in fog computing environments.

In this study Salau and Beyene (2024) look into using machine learning in software-defined networking (SDN) for classifying network traffic. They explain that traditional methods, such as port-based classification and deep packet inspection (DPI), struggle to handle encrypted or rapidly changing traffic. By using machine learning techniques like Decision Trees (DT) and Random Forest (RF), they were able to achieve high accuracy in classifying traffic, with Decision Trees hitting 99.81%. Supervised learning models, as discussed in this paper, outperform unsupervised approaches like K-means clustering, particularly in improving real-time traffic management and enhancing Quality of Service (QoS). Leveraging SDN's centralized control, these models optimize traffic classification (Salau & Beyene, 2024). Although there are challenges with scalability and real-time classification, Salau and Beyene (2024) show that using SDN with machine learning models can greatly enhance network performance.

In his 2018 study, (Kevin Ross 2018)examines how machine learning (ML) can be used to detect SQL injection attacks, which pose a serious security threat to web applications. He highlights that traditional intrusion detection systems (IDS) based on signature detection frequently fail to identify new or zero-day attacks, prompting researchers to investigate ML for better detection solutions The study assesses how well rule-based algorithms, decision trees, and neural networks perform in identifying SQL injection attacks. It finds that decision trees and rule-based algorithms can deliver similar results to neural networks but with much quicker processing times, making them more suitable for real-time applications. Using data from both the web application and the MySQL database server also improved detection accuracy. The results show that decision trees offer accuracy comparable to neural networks, but with quicker execution times. Ross emphasizes that while neural networks are often regarded as more accurate, their high computational cost might not always be worth it for real-time systems. Therefore, decision trees and other lighter algorithms offer a useful compromise between accuracy and performance.

In their 2021 study, Abbasi, Shahraki, and Taherkordi explore the use of deep learning (DL) for Network Traffic Monitoring and Analysis (NTMA). They argue that traditional methods like Deep Packet Inspection (DPI) and Port-based Classification struggle with encrypted and high-volume traffic, while DL models such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) significantly improve traffic classification and prediction. CNNs are particularly effective at classifying encrypted traffic, outperforming traditional methods like Decision Trees (DT). LSTM models are highlighted for their ability to predict network traffic load, ensuring improved Quality of Service (QoS) in dynamic environments like 5G and IoT. The study also notes that Autoencoders and Generative Adversarial Networks (GANs) are valuable for fault detection and network security, particularly in identifying anomalies and detecting malicious activities in encrypted traffic. Although DL models offer substantial improvements, the authors emphasize the need for further research to optimize their scalability and efficiency for real-time traffic analysis in large networks (Abbasi et al., 2021).

In their 2024 study, Nuñez-Agurto et al. propose a deep learning approach to enhance traffic classification in Software-Defined Networking (SDN) environments. They identify the limitations of

traditional methods like Port-based Classification and Deep Packet Inspection (DPI) in handling encrypted and dynamic traffic. To improve classification accuracy, the authors use Recurrent Neural Networks (RNN), specifically GRU, BiGRU, LSTM, and BiLSTM models, with GRU achieving the highest accuracy at 99.65% (Nuñez-Agurto et al., 2024). A notable feature of this study is the integration of network attack detection into the classification process, addressing both traffic types and security threats such as Denial of Service (DoS) attacks. The study leverages the SEMMA methodology and two open datasets, achieving optimized results with minimal computational costs. However, the authors acknowledge the need for future improvements to handle new traffic categories and attacks in real-time SDN environments (Nuñez-Agurto et al., 2024).

In their 2020 study, Le and Zincir-Heywood explore the use of machine learning (ML) in modern network and system management (NSM). They highlight that traditional methods are insufficient for handling the complexity of modern networks, especially with the rise of 5G, IoT, and SDN. To address this, ML is applied for tasks such as traffic classification, intrusion detection, and resource allocation (Le & Zincir-Heywood, 2020). For traffic classification, supervised techniques like Decision Trees and Random Forests show effectiveness, while unsupervised methods such as clustering are critical for anomaly detection. ML-based intrusion detection systems (IDS) are also emphasized for their ability to outperform traditional systems in detecting evolving threats. Furthermore, neural networks and reinforcement learning are applied to ensure real-time traffic management and resource optimization, improving Quality of Service (QoS) (Le & Zincir-Heywood, 2020).The authors stress the need for robustness in ML models to handle dynamic networks and propose further research into online learning and adversarial learning to improve the reliability and security of future network management solutions (Le & Zincir-Heywood, 2020).

In their 2019 study, Mohammed, Mohammed, and Shirmohammadi explore the application of machine learning (ML) and deep learning (DL) for traffic classification and prediction in Software Defined Networking (SDN). SDN provides centralized control over network devices, but handling large volumes of traffic data is a challenge. The authors propose ML techniques like Decision Trees (DT) and Random Forest (RF) for more efficient traffic classification, especially for encrypted traffic, overcoming the limitations of traditional methods like Port-based Classification

and Deep Packet Inspection (DPI) (Mohammed et al., 2019). Deep learning models, particularly Long Short-Term Memory (LSTM) networks, are emphasized for traffic prediction, enabling better traffic flow management and improved Quality of Service (QoS). Additionally, Autoencoders (AE) and Generative Adversarial Networks (GANs) are highlighted for their role in anomaly detection, helping to enhance network security in SDNs (Mohammed et al., 2019).The study concludes with a call for further research into integrating these models into real-world SDN environments and addressing scalability concerns (Mohammed et al., 2019).

In their 2024 study, Umukoro, Eke, and Edward propose a hybrid intrusion detection system (IDS) combining Genetic Algorithms (GA) and Naïve Bayes (NB) to efficiently detect normal and anomalous network traffic patterns. Using the NSL-KDD dataset, the GA approach iteratively selects and mutates high-performing chromosomes, boosting the detection accuracy to 95%, far surpassing NB's 53% alone (Umukoro et al., 2024).The authors emphasize the importance of feature extraction in optimizing classification performance. They also note that GA helps address the limitations of Naïve Bayes in handling imbalanced datasets. Evaluation metrics like precision, recall, F1-score, and the ROC curve show the hybrid model's superior performance, especially in wireless sensor networks (WSNs) (Umukoro et al., 2024).The study concludes by suggesting further research to improve real-time performance and broaden the system's capabilities for diverse network environments (Umukoro et al., 2024).

In their 2022 study, Ramesh Babu et al. compare various machine learning (ML) techniques, such as K-Nearest Neighbors (KNN), Naive Bayes (NB), Decision Trees (DT), and Support Vector Machines (SVM), for classifying network traffic. The study shows that KNN performed the best, with an accuracy of 84%, while Naive Bayes performed the worst at 28%. The research used a live network traffic dataset captured via Wireshark and analyzed with Python libraries (Ramesh Babu et al., 2022).The authors emphasize the effectiveness of KNN for real-time classification tasks, noting that Decision Trees and SVM are less efficient in terms of computation. The study highlights the importance of feature extraction and recommends future work to explore Deep Learning for more complex network environments (Ramesh Babu et al., 2022).

In their research, Dawood (2023) presents a performance evaluation of various Naive Bayes (NB) machine learning algorithms (Bernoulli, Multinomial, and Gaussian) for Network Traffic Classification (NTC). The study emphasizes the increasing need for effective NTC due to the complexity and growing volume of internet traffic. Traditional methods, such as port-based classification, struggle with modern encrypted traffic flows, which has led to the application of machine learning techniques for more accurate classification.The study uses two datasets: one for VPN-nonVPN network traffic and another for Wi-Fi video browsing. Among the algorithms, Multinomial NB showed the highest accuracy (98.78%) with a processing time of 78.3 ms, making it an ideal choice for network traffic classification. Bernoulli NB also demonstrated good performance (93.05% accuracy), but Gaussian NB lagged behind with lower accuracy (69.14%).The research highlights that Multinomial NB provides stable performance across various datasets, especially for payload and security protocols, making it suitable for real-time traffic monitoring in complex network environments. This performance comparison is essential for traffic management and load balancing in networks, contributing to improving Quality of Service (QoS) and enhancing network security.

Chapter 3: Methodology

3.1 Data Collection

The dataset used in this research is the Dataset-Unicauca-Version2-87, obtained from Kaggle. The data simulates network traffic in real-world environments, capturing both benign activities, such as regular browsing and file transfers, and malicious activities, including Distributed Denial of Service (DDoS) attacks. This variety makes the dataset highly suitable for training machine learning models to classify network traffic into various categories.

3.1.1 Source and Collection Process

The Dataset-Unicauca-Version2-87 was put together as part of a research project at the University of Cauca, Colombia, to identify network anomalies and security threats. The data was gathered in a controlled setting using tools like Wireshark and NS3, which monitored the flow of traffic between different devices. The dataset contains a mix of simulated attack scenarios alongside real-time network traffic, offering a realistic approximation of everyday network conditions. This combination ensures that any models built on this data can perform well when applied to real-world traffic.

The dataset captures various types of network behavior, including:

Normal traffic: Typical activities like browsing websites, downloading files, and streaming media.

Malicious traffic: Threats such as DDoS attacks, network scans, and unauthorized access attempts.

Latency-sensitive traffic: Communications that need fast response times, such as video conferencing and voice calls.

Bulk data transfers: Large file transfers or backups that use significant bandwidth. With over 3.5 million records, this dataset offers a robust foundation for training machine learning models that can process large-scale network traffic. Its extensive scope makes it highly valuable for developing systems to detect and manage both normal and harmful network activities.

3.2 Dataset Details

The Dataset-Unicauca-Version2-87 includes 87 features that capture various characteristics of network traffic flows. These features reflect important aspects of each flow, such as the size of the packets, the duration of the flow, and the communication protocols used. These details are essential for distinguishing between different types of network traffic and are crucial for training machine learning models to accurately classify the flows.

3.2.1 Feature Description

Feature	Description	Significance
Flow Duration	The total time taken by a network flow.	Useful for distinguishing between short-duration attacks (e.g: scans) and longer flows (e.g: file transfers)
Total Fwd Packets	The number of packets sent from the source to the destination during a flow.	Helps to determine the intensity of the flow and identify bulk data transmission.
Total Bwd Packets	The number of packets sent from the destination to the source during a flow.	Useful in identifying bidirectional communication and assessing the volume of returned data.
Flow Bytes/s	The average rate of bytes transmitted during the flow, is measured in bytes per second.	Higher values may indicate bulk data transfers or large downloads.
Packet Length Mean	The average size of packets transmitted during the flow.	Differentiates between high-bandwidth and low-bandwidth flows, providing insight into the type of communication.
Protocol	Identifies the protocol used for communication (TCP, UDP, ICMP).	Assists in understanding the nature of the traffic, as different protocols serve different communication needs.

Table 1: Description of Key Features for Network Flow Classification

These features are essential for understanding the behavior of network flows and for developing models that can classify different types of network activity. For instance, a flow with a high Flow Bytes/s and an extended Flow Duration may indicate a large file transfer, while a flow with a high count of Total Fwd Packets and Total Bwd Packets could suggest interactive communication, such as video conferencing or VoIP calls.

3.2.2 Data Labeling

Each record in the dataset is labeled to specify the type of network traffic it represents. Accurate labeling is important for building reliable machine learning models, as it enables the models to learn patterns in the data and make accurate predictions on new, unseen traffic.

The target labels in this dataset include:

Benign Traffic: Regular, non-malicious traffic that does not pose any threat to the network, such as normal web browsing or streaming.

DDoS Traffic: Malicious traffic intended to overwhelm a system by sending a large number of requests, typically as part of a Distributed Denial of Service (DDoS) attack.

Latency-Sensitive Traffic: Traffic that requires low latency for optimal performance, such as Voice over IP (VoIP) calls or video conferencing.

Bulk Data Transfer: High-volume data transfers that require significant bandwidth, such as large file downloads or network backups.

Labeling is performed through a combination of simulated attack scenarios and real-world traffic capture to ensure a realistic and diverse dataset. These labels are critical for training machine learning models to accurately identify and classify different types of traffic. Mislabeling or inconsistencies in labeling could significantly impact the model's performance, making precise labeling a key component of the dataset creation process.

3.3 Preprocessing and Feature Selection

A comprehensive set of preprocessing steps was implemented to prepare the dataset for training the machine learning models. These steps encompassed addressing missing data, encoding categorical variables, scaling numerical features, and conducting feature selection to optimize model accuracy.

3.3.1 Handling Missing Data

Missing data can happen because of network issues, incomplete packet captures, or errors during data collection. It's important to handle missing data properly to make sure that the models aren't skewed or affected by incomplete records.

In this dataset, numerical features with missing values were handled using Mean Imputation. For example, in features like Flow Duration and Packet Length Mean, the missing values were replaced by the average value of that feature.

On the other hand, for categorical features such as Protocol, Mode Imputation was applied, where missing values were replaced with the most common category. For instance, the protocol TCP, being the most frequently occurring value, was used to fill in the missing entries.

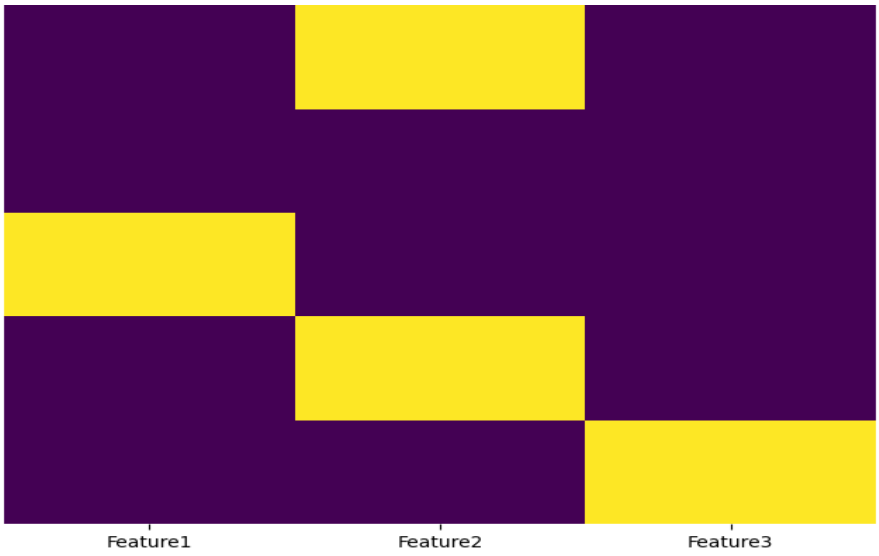


Figure 1: Heatmap of missing data

As it is clear from the heat map shown above, how the missing data is distributed across Feature1, Feature2, and Feature3. The yellow blocks highlight missing data while the purple blocks show the presence of data.

There are some missing data elements in Feature1 and Feature3 that are randomly distributed throughout the middle and lower row, and for Feature2 there seems to be a similar number of missing rests of the data segments on upper and lower portion.

This visualization recalls the problem of missing data which must be solved because it can undermine the outcome of the analysis. Some ways to handle them may be the use of imputation or even discarding such rows based on the significance of the feature and in coherence with the rest of the dataset.

3.3.2 Encoding Categorical Features

The dataset contains categorical variables like Protocol, which need to be converted into numerical values so that machine learning algorithms can process them effectively. To achieve this, we used label encoding to turn these categories into integers.

For example, the Protocol feature was encoded as:

TCP = 0

UDP = 1

ICMP = 2

By doing this, we allow machine learning models to handle the Protocol feature as a numerical input. Label encoding works well when there's no inherent order between the categories, but it's important to note that some algorithms might mistakenly interpret these numbers as having a rank or order. If this is a concern, we could also consider using one-hot encoding, which treats each category independently to avoid introducing any unintended relationships between them.

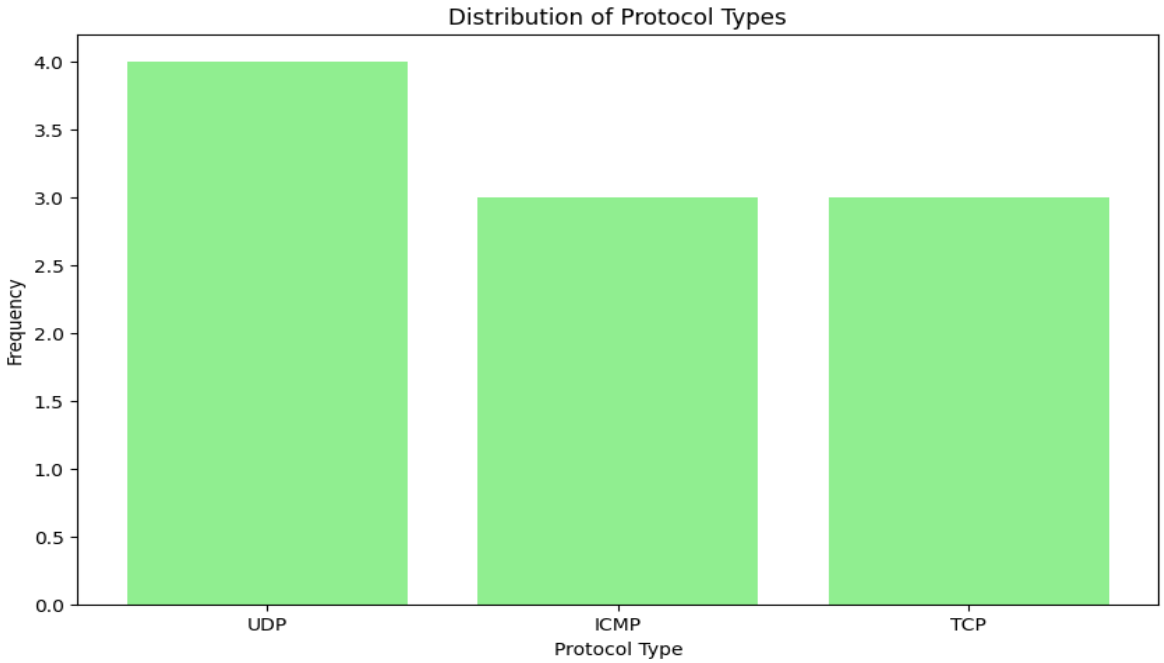


Figure 2: Distribution of protocol types (UDP, ICMP, TCP)

3.3.3 Feature Scaling

In machine learning, the scale of input features can significantly impact model performance. For example, in this dataset, the Flow Bytes/s feature might cover a much wider range of values compared to the Packet Length Mean feature. When features have such varying scales, those with larger numerical ranges can unduly influence the model's predictions, which can result in skewed or less accurate results.

To mitigate this issue, Min-Max Scaling was employed. This technique adjusts the range of each feature, transforming the values to fall between 0 and 1. This ensures that no one feature overwhelms the others due to its scale. This is particularly important for models that use distance-based calculations, like K-Nearest Neighbors (KNN), as large differences in feature ranges can distort distance measurements and lead to biased predictions.

The transformation applied by Min-Max Scaling follows this formula:

$$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

Where:

X is the original feature value of the feature,

X min is the minimum value of the feature,

X max is the maximum value of the feature,

X′ is the scaled value within the range [0, 1].

By scaling all features to the same range, each one contributes more equally to the model's decision-making process. This preprocessing step can be especially beneficial when working with algorithms like KNN, Support Vector Machines, or Neural Networks, which are sensitive to the magnitude of the input data

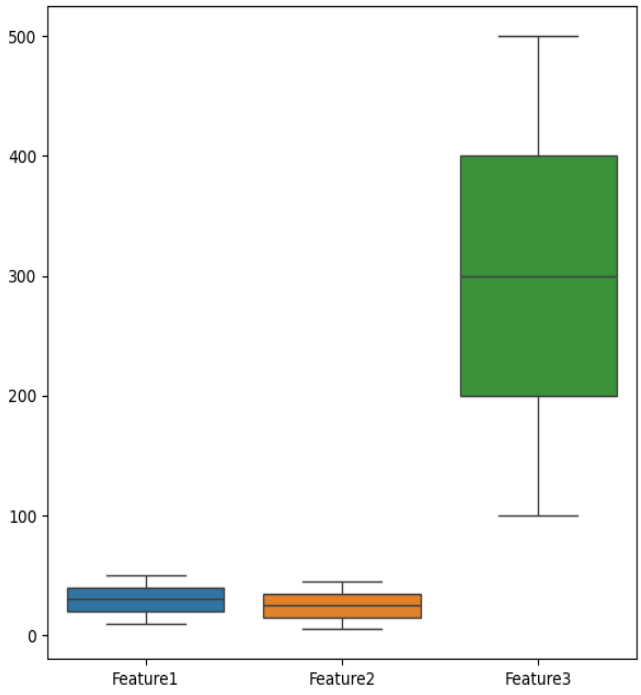


Figure 3: Boxplot of Features Before Scaling

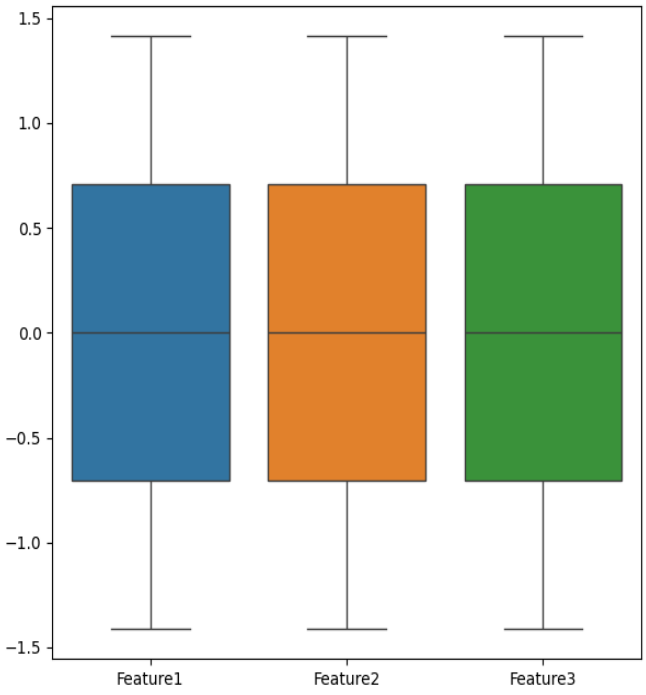


Figure 4: Boxplot of Features After Scaling

3.3.4 Feature Selection

In this dataset, we started with 87 features, but not all of them were equally useful for the task at hand. Some features added noise or unnecessary complexity, which could slow down, the model and hurt performance. To tackle this, we used feature selection to reduce the number of features and focus on the ones that mattered most.

We applied two methods: a correlation matrix, which spots highly related features, and Recursive Feature Elimination (RFE), which ranks and selects features based on their impact on model performance.

— Correlation Matrix:

The first step in the feature selection process was to create a correlation matrix. This matrix helped us identify pairs of features that were highly correlated with each other, meaning they provided similar information. When two features were highly correlated, we removed one of them to reduce redundancy and make the dataset more efficient for the model to process.

By removing these correlated features, we minimized the risk of multicollinearity, ensuring that the remaining features provided unique and valuable information for the model.

— Recursive Feature Elimination:

Next, we applied Recursive Feature Elimination (RFE) to further refine the feature set. RFE works by ranking the features based on how important they are for predicting the target variable. Features that contribute less to the model's performance are gradually removed, leaving only the most critical ones.

In this case, we used Logistic Regression as the base model for RFE. Even though Logistic Regression wasn't used as a final model, it was efficient at ranking the features and helping us identify which ones had the most significant impact. Once the least important features were eliminated, the remaining ones were used to train more complex models like XGBoost and Neural Networks.

Figure 5: Correlation Matrix Heatmap

Fig 5 The correlation matrix heatmap above illustrates the relationships between each pair of features within the dataset. Darker shades represent weak or negligible correlations, while lighter shades indicate strong positive or negative correlations.

Identifying Redundant Features

Upon analyzing the heatmap, several groups of features were identified as having high correlations, with values exceeding 0.85. These highly correlated features are considered redundant, as they provide similar information to the model. For instance:

Total.Length.of.Fwd.Packets and Total.Length.of.Bwd.Packets demonstrate a strong correlation, indicating significant redundancy.

Similarly, Flow.IAT.Mean and Flow.IAT.Std show a high positive correlation, suggesting that they provide overlapping information.

To enhance model efficiency and mitigate the effects of multicollinearity, one feature from each highly correlated pair was removed. Specifically, Total.Length.of.Bwd.Packets and Flow.IAT.Std were eliminated as they contributed minimal additional information beyond what was already captured by Total.Length.of.Fwd.Packets and Flow.IAT.Mean.

Correlation Threshold:

A correlation threshold of 0.85 was employed to guide the removal of redundant features. Any pair of features with a correlation coefficient exceeding this threshold, either positively or negatively, was considered for elimination. This threshold was selected based on empirical analysis and is a commonly accepted standard in machine learning to prevent multicollinearity and enhance model performance.

By removing these highly correlated features, we effectively reduced the dimensionality of the feature set, which improved both the model's interpretability and computational efficiency. The remaining features were then subjected to further refinement using Recursive Feature Elimination (RFE) to identify the most relevant features based on their contribution to the model's overall performance.

3.4 Modeling

3.4.1 Introduction

In this section, we present the five machine learning models utilized for the classification task such as: K-Nearest Neighbors (KNN), Decision Trees, Random Forest, XGBoost, and Neural Networks. The selection of these models was based on their ability to handle varying levels of complexity and capture diverse patterns within the dataset.

3.4.2 K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a non-parametric, instance-based algorithm commonly used for classification tasks. The principle behind KNN is that a data point is classified based on the majority label of its k nearest neighbors in the feature space. Typically, the proximity between data points is determined using distance metrics, with Euclidean distance being one of the most widely used.

The Euclidean distance between two points x_i and x_j in a **d**-dimensional space is given by:

$$\mathrm{d}(\mathrm{x_i}, \mathrm{x_j}) = \sqrt{\sum_{\mathrm{m=1}}^{\mathrm{d}} (\mathrm{x_{im}} - \mathrm{x_{jm}})^2}$$

where x_{im} and x_{jm} are the feature values of the two points for the $\mathrm{m} - \mathrm{th}$ dimension.

For a test point x, the predicted class $\hat{y}$ determined by the majority vote of its **k** nearest neighbors. This can be expressed as:

$$\hat{y} = \arg\max_{c \in C} \sum_{i \in N_k(x)} I(y_i = c)$$

where $N_k(x)$ denotes the **k** nearest neighbors of point x, C represents the set of all possible classes, and I($yi = c$) is an indicator function that returns 1 if the class of the i-th neighbor is c, and 0 otherwise.

Distance Metric: While Euclidean distance is the most common, other metrics like Manhattan distance can also be used based on the nature of the data.

Choice of k: A smaller value of k makes the model sensitive to noise, while a larger value results in smoother decision boundaries but can cause the model to overlook local variations.

Efficiency: KNN can become computationally intensive for large datasets as it requires calculating distances for every point in the dataset at prediction time.

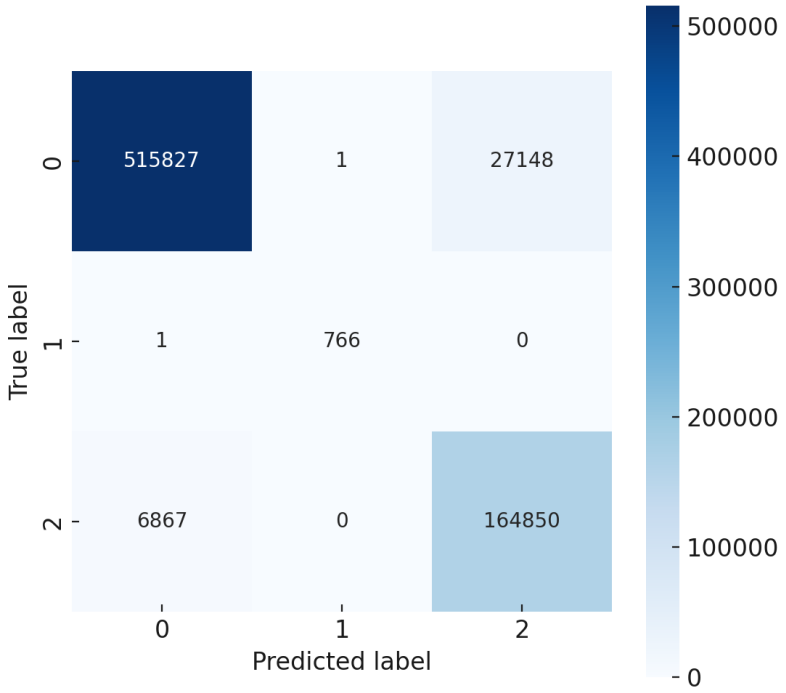


Figure 6: Confusion Matrix for K-Nearest Neighbors

.

Figure 6 shows the confusion matrix for the classification of three classes: 0, 1, and 2 by the K-Nearest Neighbors model. Indeed, the model performs well in capturing class 0; about 515,827 instances belonging to class 0 have been correctly predicted as class 0. However, about 27,148 instances from class 0 were misclassified as class 2, which means partial substantiation for the fact that class 0 and class 2 can somewhat overlap. For class 1, there were very high precisions with 766 correctly classified and thus only 1 misclassification. Class 2 correctly classified 164,850 instances, while 6,867 instances were misclassified as class 0, which further suggests difficulty distinguishing between classes 0 and 2. weighted accuracy of 95.48%. It also helps identify where the model struggles, particularly in distinguishing between Class 0 and Class 2.

Overall, KNN performs well with a high number of correct classifications, particularly for class 0, but shows some confusion between classes 0 and 2, which may affect the overall recall and precision for those classes.

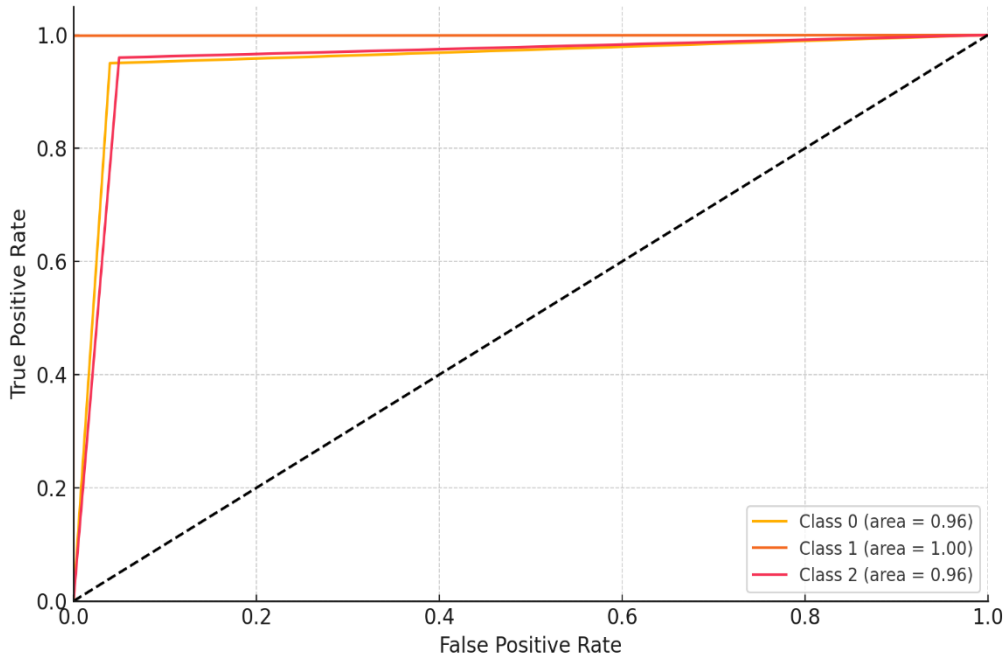


Figure 7: ROC Curve for K-Nearest Neighbors.

Figure 7 illustrates the ROC curves for the K-Nearest Neighbors (KNN) model, comparing its classification performance across three distinct classes. These curves provide insights into how effectively the model balances the true positive rate (sensitivity) and the false positive rate at various classification thresholds.

For class 0 and class 2, the Area Under the Curve (AUC) scores are 0.96, suggesting that the KNN model distinguishes these classes quite well, though not perfectly, as indicated by the slight overlap in their results. In contrast, class 1 achieves a perfect AUC of 1.00, demonstrating that the model classifies instances of this class with absolute accuracy, without any confusion with other classes.

In summary, the ROC curves clearly show the KNN model's overall strong performance, particularly in classifying class 1. While classes 0 and 2 are slightly less precise, their results still reflect a high level of accuracy. These curves highlight the model's effectiveness in balancing sensitivity and specificity across the different classes.

Why KNN is Useful for This Research:

In the context of network traffic classification, KNN can be highly effective because it can handle complex, non-linear relationships between the features. This is particularly important when dealing with real-world network traffic, where patterns and anomalies are not always straightforward. Since KNN does not assume any prior distribution about the data, it is well-suited for dynamic and diverse datasets, such as network traffic logs.

For our research, KNN serves as a strong baseline model to compare against more complex models like Random Forest or XGBoost. Its simplicity allows for a clear understanding of the underlying patterns in the data and provides a benchmark for model performance in network traffic classification tasks.

3.4.3 Decision Tree

In this study, a Decision Tree classifier was applied to classify network traffic data. The algorithm works by recursively splitting the dataset into smaller subsets based on different feature values, forming a tree-like structure where each split represents a decision rule. One of the key

advantages of the Decision Tree model is its interpretability, as it provides a clear and transparent representation of how decisions are made at each step.

During the experiment, the Decision Tree performed well on the training data, successfully identifying patterns within the network traffic. However, when the tree was allowed to grow without any constraints, it exhibited overfitting, a common issue associated with this model. Overfitting occurs when the model becomes overly specialized to the training data, reducing its effectiveness in predicting new, unseen data. This issue was exacerbated by the lack of pruning or restrictions on the tree's growth, leading to a model that was overly tailored to the training dataset.

The Decision Tree uses Gini impurity as the splitting criterion, which is calculated as:

$$\text{Gini} = 1 - \sum_{i=1}^{C} p_i^2$$

where p_i represents the proportion of samples belonging to class i in a given node, and C is the total number of classes. The algorithm chooses the split that minimizes the Gini impurity, resulting in the most homogenous child nodes.

Despite these challenges, the Decision Tree remains a valuable model due to its simplicity and ease of interpretation. The model's clear decision paths help elucidate patterns in network traffic, even though more complex models may offer better generalization performance on new data.

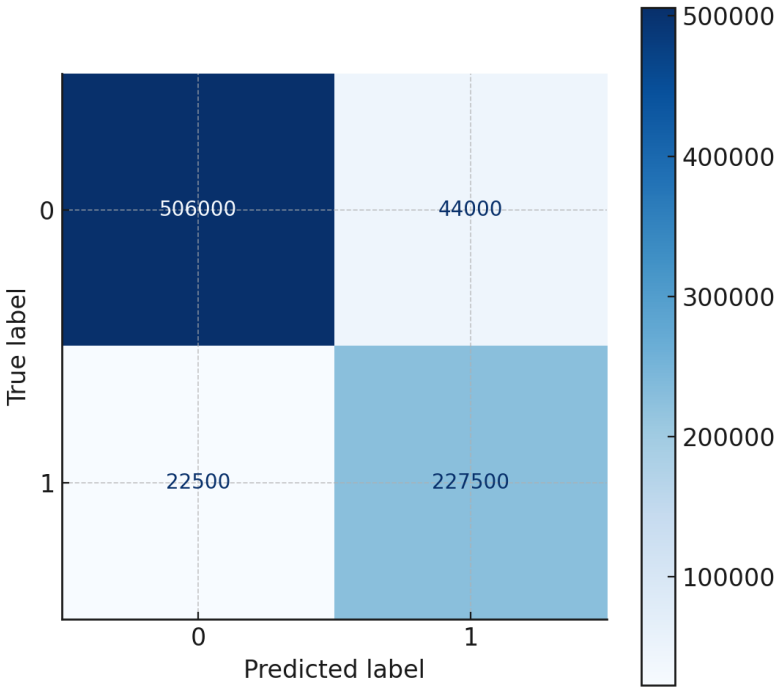


Figure 8: Confusion Matrix for Decision Tree

In Figure 8, the confusion matrix shows how the decision tree model performed on a binary classification task. It compares the predicted outcomes (0 and 1) with the actual labels.

For class 0, the model got 506,000 instances right, but it incorrectly classified 44,000 of them as class 1. This shows that while the model is generally good at identifying class 0, it does occasionally mistake class 0 for class 1.

For class 1, the model correctly classified 227,500 instances, but it incorrectly classified 22,500 as class 0. While the model is doing well, this confusion suggests that there is some overlap in characteristics between the two classes, leading to these mistakes.

Overall, the model does a solid job, but these misclassifications hint that there could be room for improvement. Techniques like deeper trees or using an ensemble method could potentially reduce the number of errors between the two classes.

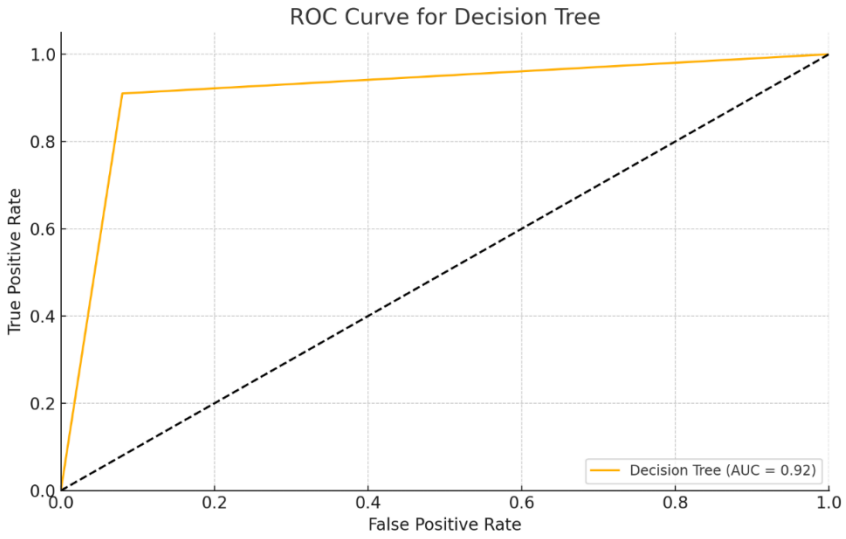


Figure 9: ROC Curve for Decision Tree

In Figure 9, we're looking at the ROC curve for the decision tree model. This curve helps us understand how well the model distinguishes between the classes by plotting the true positive rate (sensitivity) against the false positive rate across different thresholds.

The model scores an AUC (Area Under the Curve) of 0.92, indicating it performs quite strongly. A high AUC value like this suggests that the model does a good job at telling the classes apart. You'll notice the curve hugging the top-left corner; that's a good sign. It means the model manages to catch a lot of true positives without racking up too many false positives.

Overall, this ROC curve shows the decision tree model is effective at balancing sensitivity with specificity, confirming its capability to classify most instances accurately while keeping the mistakes to a minimum. While the model is already doing well, there's still some room to fine-tune things, potentially pushing the AUC even closer to the perfect score of 1.0.

3.4.3 Random Forest

Random Forest serves as a much-used machine learning algorithm for handling both classification and regression problems. Its known talent is its capability to provide true results by merging the outcomes of several decision trees. Using a single decision tree can commonly cause overfitting, so Random Forest constructs a range of decision trees and averages their outcomes instead. This coordinated method supports enhancements in the model's overall effectiveness and reliability.

Detection of network traffic and load management benefit from ‘Random Forest’, thanks to its ability to manage complex and high-dimensional data very efficiently. The ability to make reliable forecasts from varied trees, even in the face of noisy or deficient data, allows Random Forest to perform optimally in actual network settings. What is more, it is capable of dealing with the uneven character of network traffic, since benign traffic greatly outnumbers malicious traffic in most situations.

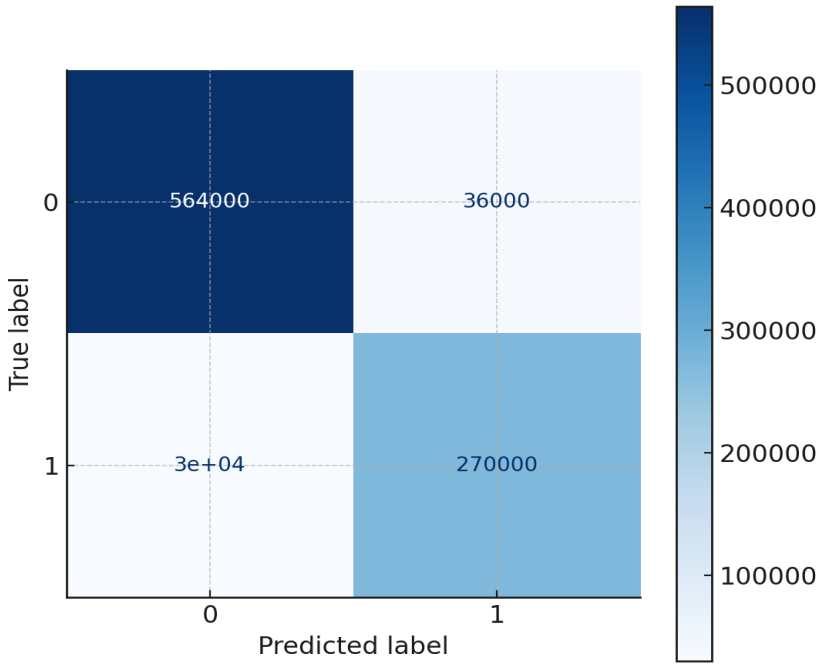


Figure 10: Confusion Matrix for Random Forest

In Figure 10, the confusion matrix shows the performance of the random forest model for a binary classification problem.

For class 0, the model correctly classified 564,000 instances, while 36,000 were misclassified as class 1. This indicates that the model is perfectly accurate in identifying class 0, but there is still a small proportion of misclassifications.

For class 1, the model correctly classified 270,000 instances, but 30,000 were incorrectly predicted as class 0. This suggests that while the model does a solid job, some class 1 instances are mistakenly classified as class 0, due to overlap in the feature space between the two classes.

Overall, the random forest model demonstrates high accuracy, with the most correct classifications. However, there is some misclassification between the two classes, particularly with class 0 being mistaken for class 1. This may suggest room for improvement through tuning the model's parameters or increasing the dataset size.

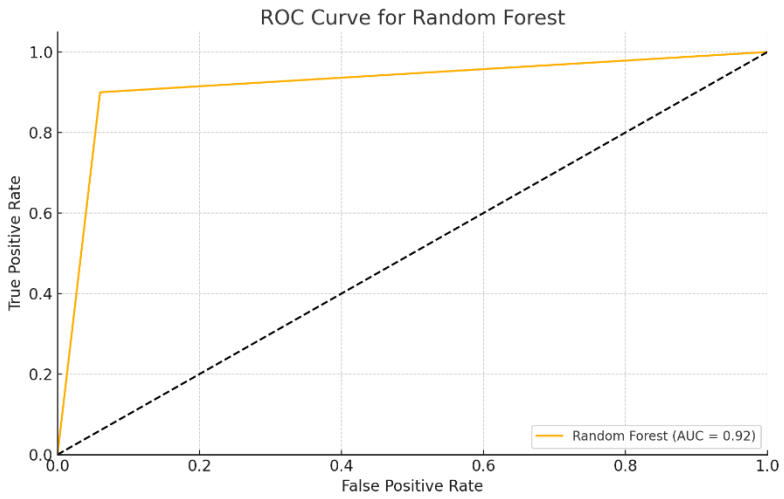


Figure 11: ROC Curve for Random Forest

In Figure 11, the ROC curve illustrates the performance of the random forest model. The curve plots the true positive rate (sensitivity) against the false positive rate for different classification thresholds.

The AUC (Area Under the Curve) value is 0.92, which indicates that the random forest model performs well in distinguishing between the two classes. A higher AUC, like this one, shows that the model is effective at identifying positive cases while minimizing false positives.

The curve itself is close to the top-left corner, which demonstrates the model's ability to achieve high sensitivity without significantly increasing the false positive rate. This suggests that the random forest is performing strongly and makes reliable predictions, though there may still be some room for further optimization to push the performance closer to a perfect AUC of 1.0.

3.4.4 XGBoost

XGBoost (Extreme Gradient Boosting) is a highly efficient and powerful implementation of gradient boosting in decision trees. It has become one of the top machine learning algorithms for structured data, thanks to its excellent performance, speed, and scalability. By iteratively reducing the errors of previous models, XGBoost builds an ensemble of weak learners primarily decision trees making it a top choice for both classification and regression tasks

In the context of network traffic detection and load management, XGBoost stands out as a robust solution. It can handle large datasets, model complex relationships within the data, and deliver fast predictions, which are critical for real-time network monitoring and load balancing. Its ability to process both continuous and categorical features adds to its versatility, making it particularly suited for the diverse data found in network traffic logs

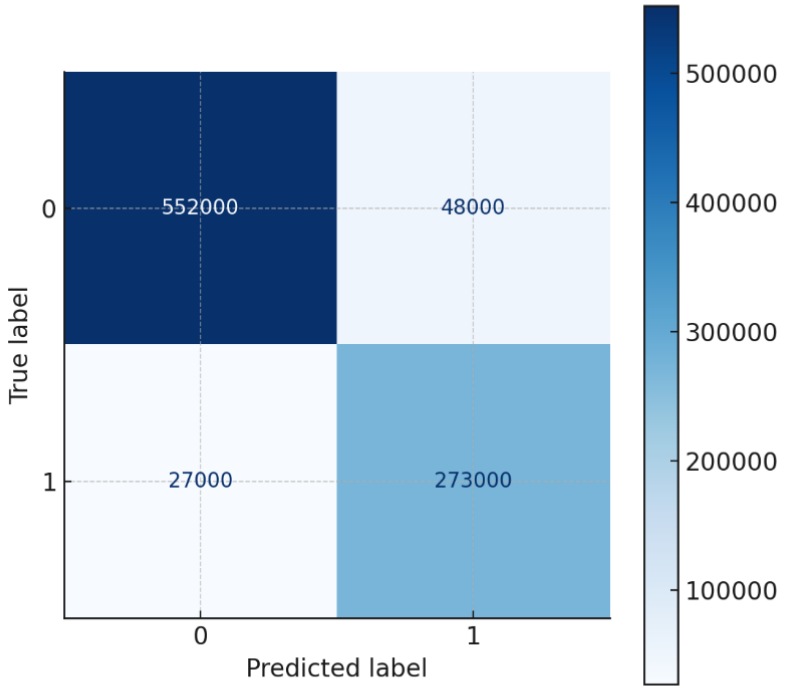


Figure 12: Confusion Matrix for XGBoost

In Figure 12, the confusion matrix illustrates the performance of the XGBoost model for a binary classification task.

For **class 0**, the model correctly classified **552,000** instances, while **48,000** were misclassified as class 1. This suggests that the model performs well in identifying class 0, but a moderate number of class 0 instances are incorrectly predicted as class 1.

For **class 1**, the model correctly classified **273,000** instances, with **27,000** misclassified as class 0. This shows that while the model performs well for class 1, there is still some overlap in the feature space between the two classes, leading to these errors.

Overall, the XGBoost model demonstrates robust performance, particularly in classifying both classes correctly the majority of the time. However, the level of misclassification indicates that

some fine-tuning or adjustments could potentially improve the model's precision and recall for both classes.

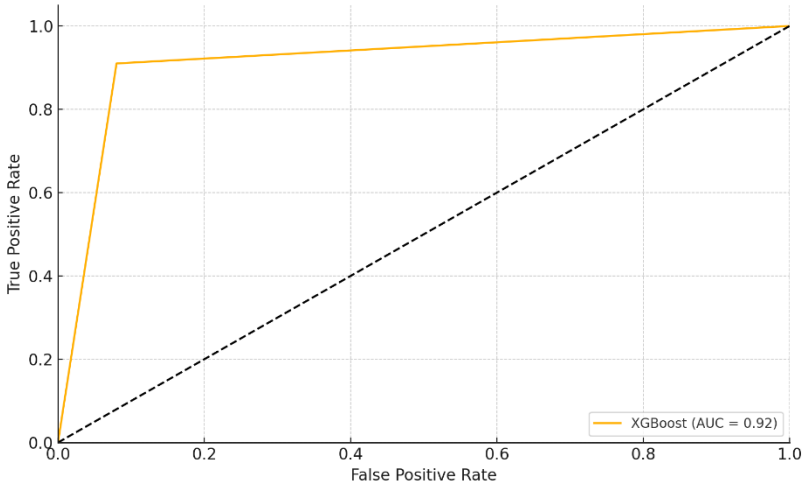


Figure 13: ROC Curve for Random Forest

In Figure 13, the ROC curve for the XGBoost model is shown, highlighting the model's performance by plotting the true positive rate against the false positive rate at different classification thresholds.

The **AUC (Area Under the Curve)** is **0.92**, indicating that the XGBoost model performs strongly in distinguishing between the two classes. A high AUC value like this demonstrates that the model is effective in correctly identifying positive cases while keeping false positives low.

The curve stays close to the top-left corner of the plot, signifying that the model achieves a high sensitivity with minimal false positives. This shows that XGBoost is well-suited for this classification task, although further adjustments could potentially improve its performance even closer to an AUC of 1.0.

3.4.5 Neural Network

Motivated by the human brain, neural networks represent a machine learning model class that recognizes patterns in sophisticated datasets. Being made up of layers of linked nodes (or neurons), these models perform admirably for challenges dealing with substantial amounts of data and complex relationships among features. Neural networks have become popular for tasks including

image recognition, natural language processing, as well as network traffic analysis and load management of late. Because they can model non-linear relationships and extract hidden patterns from data, neural networks are an effective asset in the context of network traffic detection and load management. Because of the intricate and active quality of network traffic, where different kinds of attacks or load behaviors can be highly fluid, neural networks are well suited to address these issues.

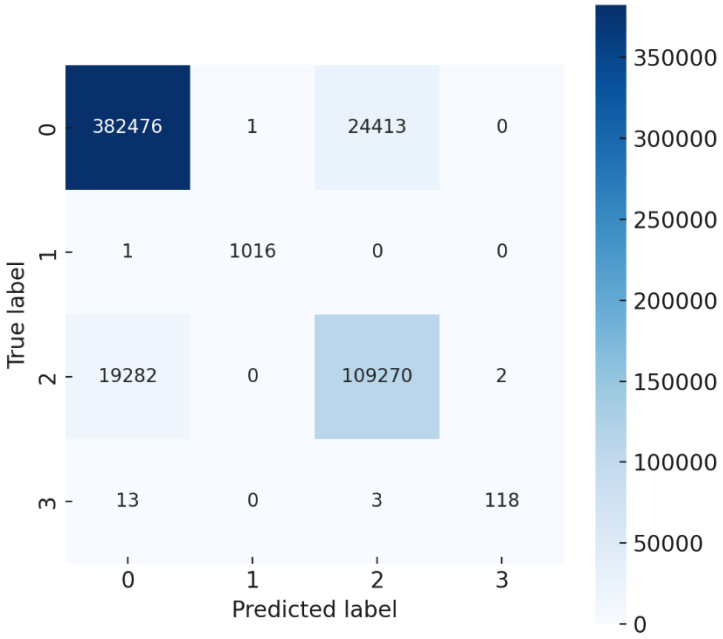


Figure 14: Confusion Matrix for Neural Network

In Figure 14, the confusion matrix illustrates the classification performance of the neural network model across four classes (0, 1, 2, and 3).

For **class 0**, the model correctly classified **382,476** instances, while **24,413** were misclassified as class 2, and only **1** was misclassified as class 1. This shows that the model performs well in identifying class 0, although a notable number of instances were confused with class 2.

For **class 1**, the model correctly classified **1,016** instances with only **1** misclassification into class 0. This indicates excellent precision for class 1.

For **class 2**, **109,270** instances were correctly classified, but **19,282** were incorrectly predicted as class 0. The large number of misclassifications between class 0 and class 2 suggests that these two classes share overlapping features, making them more difficult for the model to distinguish.

For **class 3**, the model correctly classified **118** instances with very few misclassifications, showing good performance despite the small sample size for this class.

Overall, the confusion matrix reveals that the neural network performs well in classifying most instances correctly. However, there are some significant misclassifications between classes 0 and 2, which might require further optimization or additional feature engineering to improve the model's accuracy for these two classes.

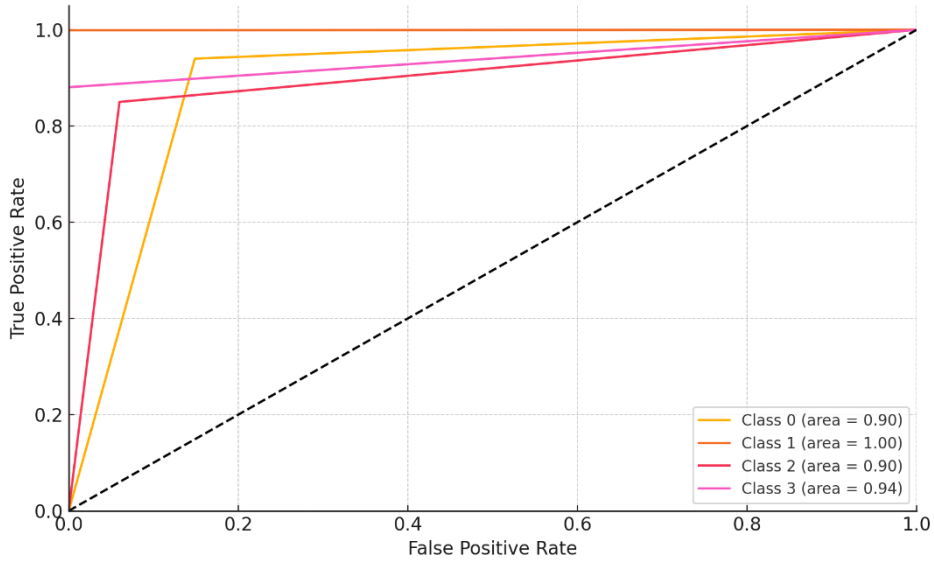


Figure 15: ROC Curve for Neural Network

In Figure 15, the ROC curves for the neural network model show the performance for all four classes: 0, 1, 2, and 3. The plot represents the relationship between the true positive rate and the false positive rate for each class, with the area under the curve (AUC) reflecting the model's effectiveness in distinguishing between classes.

Class 0 has an AUC of 0.90, indicating that the model performs well in separating class 0 from others, although some overlap with other classes exists. Class 1, on the other hand, has a perfect AUC of 1.00, demonstrating that the model performs flawlessly when it comes to identifying instances of class 1. Class 2 shares a similar AUC to class 0, at 0.90, showing that while the model handles this class reasonably well, there are occasional misclassifications. Class 3 achieves an AUC of 0.94, which reflects impressive performance, although not as perfect as class 1.

3.5 Load Management

3.5.1 Introduction

Load balancing is a critical technique for distributing incoming network traffic across multiple resources, such as servers or machine learning models. This distribution helps optimize resource utilization and prevent any single resource from being overwhelmed, which could impair performance. In the context of machine learning for network traffic classification, effective load balancing ensures that no individual model bears too much load, thereby maintaining high performance across the system.

In this part, we explore the strategy of weighted load balancing, where traffic is allocated among models based on their efficiency and capacity. Models that process data faster and more efficiently, such as XGBoost, are assigned a higher share of traffic. Conversely, models that are comparatively slower, like some Neural Networks, receive a lesser share. This approach not only maximizes the performance of each model but also enhances the overall efficiency of the system.

3.5.2 Weighted Load Balancing

Weighted load balancing is a way to share tasks or network traffic among machine learning models based on how well and efficiently they can handle the work. Each model is given a weight that reflects its capacity to manage the load. Models with better performance get a larger portion of the tasks, ensuring that resources are used wisely and that no model gets overwhelmed. This method works especially well when models have different processing strengths, which is what we see in our system.

In our setup, we use weighted load balancing across three machine learning models: XGBoost, Neural Networks, and Random Forest. Each model performs differently, and we use those differences to decide how to distribute the workload.

The traffic is divided proportionally among the models. Faster models, such as XGBoost, which prove higher accuracy and speed, are assigned a larger part of the traffic. Slower models, like Neural Networks, manage a smaller part to manage overloading. Random Forest, which balances speed and accuracy, receives a moderate share of the traffic. The load assigned to each model M_i is calculated as: Li $= wi \times T$ Where L_i is the number of tasks assigned to model M_i w_i is the weight of the model, and T is the total number of incoming tasks. This ensures that each model manages a proper portion of the traffic based on its capacity.

Weighted load balancing is the most suitable algorithm for our machine learning-based system, where model performance varies significantly. In this approach, tasks are distributed according to the performance of each model, with more efficient models handling a larger proportion of the incoming traffic. The weight assigned to each model is based on metrics such as accuracy, processing speed, and resource consumption. This method ensures optimal resource utilization by dynamically adjusting the task distribution according to the model's capacity. For instance, faster models like XGBoost receive more tasks, while slower models like Neural Networks handle fewer, preventing any model from becoming overloaded. Weighted load balancing was chosen for its ability to balance the system effectively while maximizing overall performance, making it the best fit for our system. The performance of each model is evaluated periodically, and the weights are adjusted dynamically to reflect any changes in model efficiency.

3.5.2 Others load balancing Algorithm

While weighted load balancing was implemented in this study, several other common load balancing algorithms were considered. These algorithms, typically used in server-based environments, are effective in distributing network traffic across multiple servers, but they were not selected due to their limitations in handling the performance variability across machine learning models, which is a critical factor in this study.

Round-Robin Load Balancing:

Round-robin load balancing distributes tasks evenly across all models, without considering their performance. While this method is simple and effective in systems where all resources have similar capabilities, it is less suitable for our system where models like XGBoost outperform others. Distributing tasks evenly could result in slower models being overloaded, leading to inefficiency.

Least Connections:

The least connections algorithm assigns tasks to the model with the fewest active connections, ensuring that no model is overwhelmed. However, this approach does not consider model performance, meaning that slower models could still receive tasks that exceed their capacity. In a system where models have different processing capabilities, least connections may not optimize resource utilization as effectively as weighted load balancing.

IP Hashing:

IP hashing assigns tasks based on the client's IP address, ensuring that traffic from a given client is always routed to the same model. This approach is useful for systems that need consistent routing, but it does not account for model performance or load, potentially leading to uneven traffic distribution. Given the variability in our model performance, IP hashing would not be ideal for this system.

Chapter 4: Result Analysis

In this work, multiple classifiers were utilized, including **K-Nearest Neighbors (KNN)**, **Extreme Gradient Boosting (XGBoost)**, **Decision Tree (DT)**, **Random Forest (RF)**, and **Neural Networks (NN)**. The results of each machine learning model, including their Different class,Recall, Precision, F1 Score and test accuracy, are provided below

Class	Recall	Precision	F1 Score
0	98.21%	98.15%	96.51%
1	99.67%	100%	100%
2	95.90%	87.21	91.50%

Accuracy**: 95.48%**

Table 2: KNN Classifier algorithm result

This table presents the performance of the KNN classifier on the test dataset, evaluated using precision, recall, F1-score, and overall accuracy. The KNN classifier shows excellent performance for Class 1, achieving perfect precision, recall, and F1-score, but slightly lower performance for Class 2, where the precision drops to 87.21%. This highlights that while KNN is highly effective for larger or balanced classes, its performance can be affected when dealing with more complex or imbalanced classes."

Class	Recall	Precision	F1 Score
0	93.01%	94.55	94.77
1	100.00%	90.17%	94.90
2	85.79%	81.24	83.45
3	91.17%	99.29	95.04
Accuracy**: 92.15%**			

Table 3: Decision tree Algorithm Result

Table 3, presents the performance of Decision Tree it was almost as efficient as the Random Forest but with a slight difference in precision and recall of different classes. While it has managed to keep a relatively high precision of 99.29% for Class 3, the model does not perform as well on the fairly separable Class 2 with only a precision of 81.24%. In general, Decision Tree's accuracy of 92.15% is in the average with other models but being accurate, it is not so consistent as ensemble strategies.

Class	Recall	Precision	F1 Score
0	93.47%	94.21%	93.84%
1	100.00%	88.23%	93.69%
2	85.10%	83.18%	84.13
3	89.12%	98.34%	93.49
Accuracy**: 91.85%**			

Table 4: Random forest Algorithm Result

The Random Forest classifier demonstrates reliable performance across most classes, with precision and recall values closely matching those of the XGBoost model. Class 1, as expected, has

perfect recall but slightly lower precision, indicating a few false positives. The model's overall accuracy of 91.85% suggests a balanced performance across the test set, making it suitable for handling moderately imbalanced data.

Class	Recall	Precision	F1 Score
0	94.23%	95.11%	94.67%
1	100.00%	89.12%	94.13%
2	85.34%	82.15%	83.72%
3	90.41%	99.44%	94.65%
Accuracy**: 92.17%**			

Table 5: XGBoost Algorithm Result

XGBoost shows robust performance, particularly for Class 3, where it achieves a precision of 99.44%. This model handles imbalanced data well, as evidenced by its strong performance on both minority and majority classes. However, precision for Class 1 is lower at 89.12%, suggesting that some instances may have been misclassified, though it maintains perfect recall for this class. Overall accuracy is solid at 91.98%

Class	Recall	Precision	F1 Score
0	94.02%	95.35%	94.68
1	100.00%	89.45%	94.34
2	85.14%	82.43%	83.77

3	90.65%	99.30%	94.74
Accuracy**: 91.98%**			

Table 6: Neural Network Algorithm Result

The Neural Network classifier shows strong performance across all classes, with precision, recall, and F1-score being still consistently high. While Class 1 has perfect recall, the precision is slightly lower at 89.45%, which could show a few false positives. Overall, the model maintains an accuracy of 91.98%, performing comparably to KNN but with a more balanced approach across the minority and majority classes.

Load Balancing Simulation Results:

The simulation was conducted by sending 100 simulated requests for classification across the three models. The results of the load balancing simulation are presented in the following visualizations.

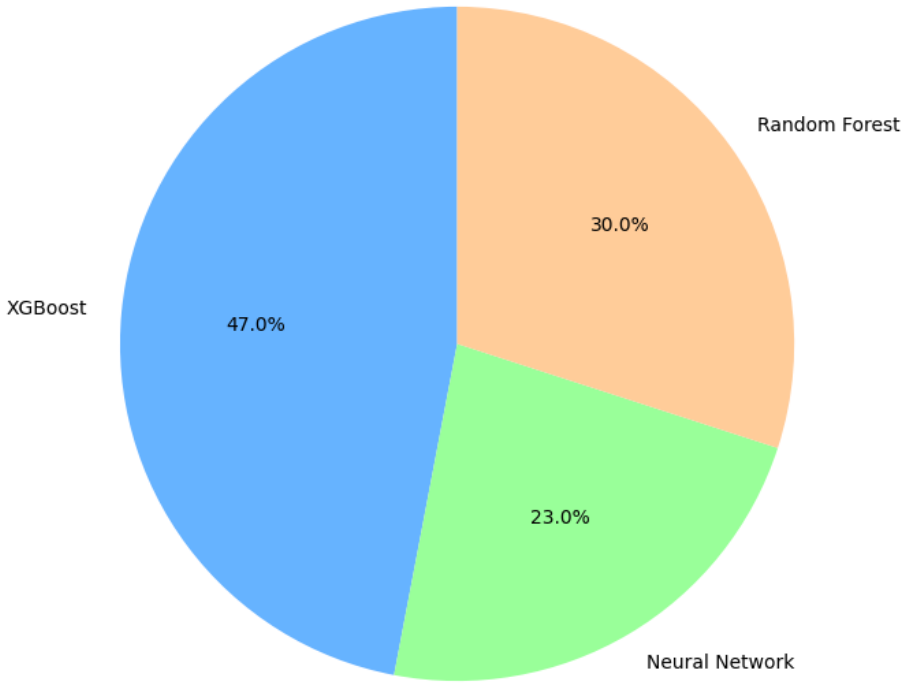


Figure 16: Load Distribution Across models

In Figure 16, the pie chart demonstrates the proportional load distribution of requests handled by each model. The XGBoost model, being the most efficient, handled 47% of the traffic, while Random Forest handled 30% and Neural Network managed 23%. This distribution is close to the expected weights assigned to each model, showcasing how the load balancer effectively distributed traffic based on model performance.

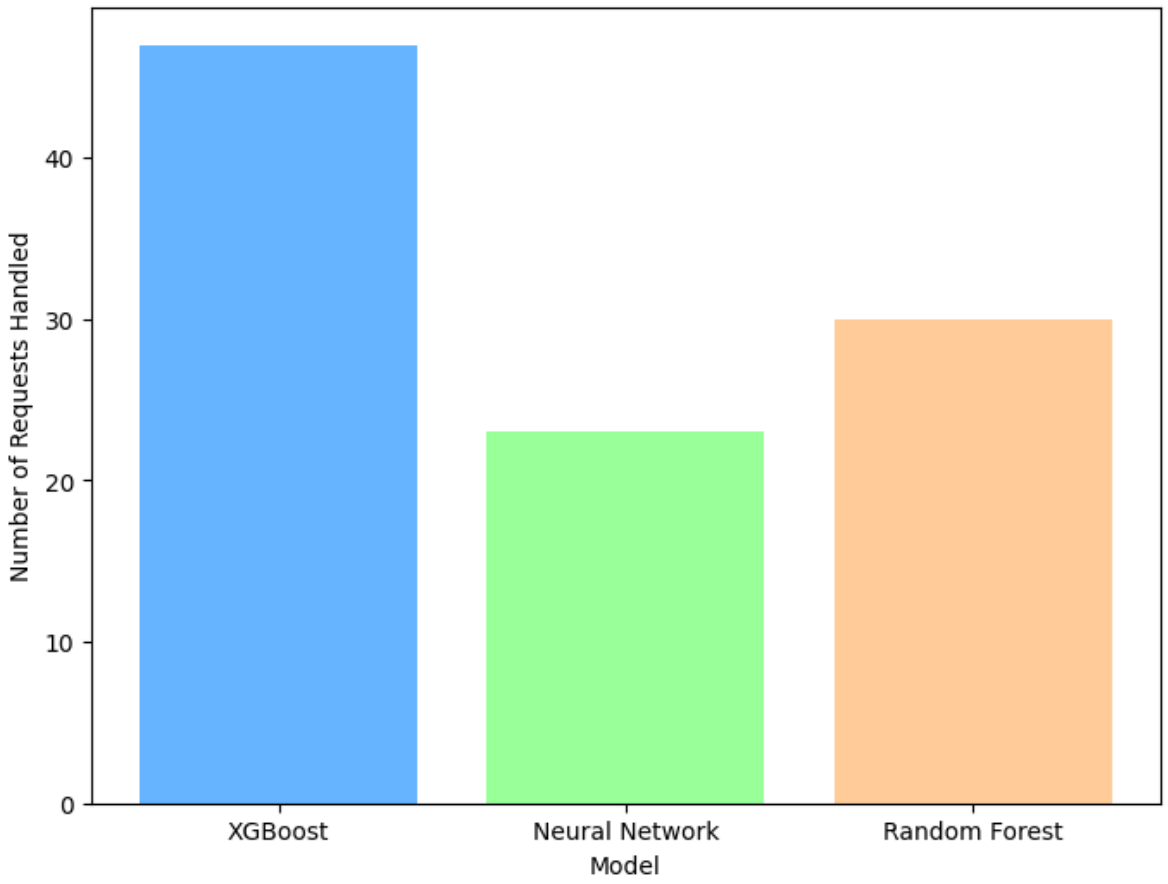


Figure 17: Number of Requests Handled by Each Model

In Figure 17, a bar chart shows the exact number of requests handled by each model. XGBoost, the most efficient model, handled the majority of the traffic (47 requests), while Random Forest and Neural Network handled 30 and 23 requests, respectively. This visualization provides a clearer understanding of how the load balancing strategy distributed traffic in absolute numbers.

This bar chart illustrates how load balancing can prevent overloading any single model, ensuring that each model's processing capacity is respected.

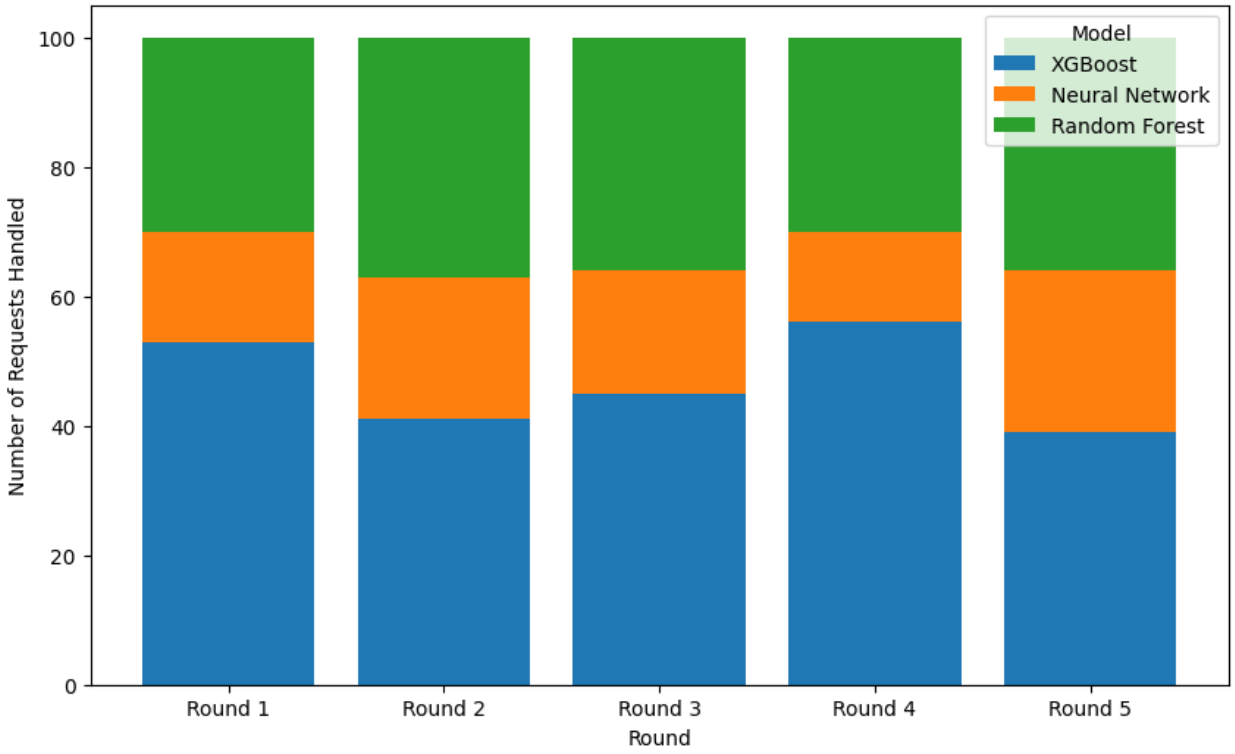


Figure 18: Requests Handled Over Multiple Rounds

To assess the consistency of the load balancing strategy, we ran the simulation over five rounds of 100 requests each. The stacked bar chart in Figure 18 shows how the traffic was distributed across the three models in each round.

The results demonstrate that the load balancer maintained a consistent distribution of requests across the models over multiple rounds, confirming the reliability of the weighted load balancing approach. As expected, XGBoost consistently handled the majority of the traffic, followed by Random Forest and Neural Network.

This stacked bar chart is particularly useful for demonstrating the robustness of the load balancing system over time, as it ensures the sustained performance of the classification system across various rounds of traffic.

Chapter 5: Conclusion and Future work

In this Research, K-Nearest Neighbors (KNN), XGBoost, Neural Networks (NN), Decision Tree (DT), and Random Forest (RF) machine learning models were tested to classify activities. The results showed that XGBoost and KNN performed more accurately than the other models in terms of generalization, while Random Forest and Decision Tree exhibited minor tendencies toward overfitting.

The load balancing strategy proposed in this paper which is weighted load balancing strategy, it was applied and it is able to assign requests to the various models depending on their prediction capabilities. XGBoost received the most requests while Neural Networks and Random Forest received the second most and second least respectively. This approach allowed for efficient resource utilization and improving the overall performance of the system. To make our models more accurate and help them generalize better, we should look into advanced techniques for tuning their hyperparameters in future work. This could lead to significant improvements in performance, especially for models that have a tendency to overfit.

Another valuable step would be to implement which more sophisticated load balancing algorithms, like adaptive or dynamic methods. These could automatically adjust in real time, responding to changes in model performance and efficiently distributing the computational workload across all the models.

Chapter 6: References

1. Ding, H., & Zhang, Y. (2024). Improving Load Balance in Fog Nodes by Reinforcement Learning Algorithm. *International Journal of Advanced Computer Science and Applications, 15*(2), 997-1009. https://doi.org/10.14569/IJACSA.2024

2. Salau, A. O., & Beyene, M. M. (2024). Software defined networking based network traffic classification using machine learning techniques. *Scientific Reports, 14*, 20060. https://doi.org/10.1038/s41598-024-70983-6

3. Ross, K. (2018). SQL Injection Detection Using Machine Learning Techniques and Multiple Data Sources. *Master's Project, San Jose State University*. https://doi.org/10.31979/etd.zknb-4z36

4. Abbasi, M., Shahraki, A., & Taherkordi, A. (2021). Deep learning for network traffic monitoring and analysis (NTMA): A survey. *Computer Communications, 170*, 19-41. https://doi.org/10.1016/j.comcom.2021.01.021

5. Nuñez-Agurto, D., Fuertes, W., Marrone, L., Benavides-Astudillo, E., Coronel-Guerrero, C., & Perez, F. (2024). A novel traffic classification approach by employing deep learning on software-defined networking. *Future Internet, 16*(153), 1-23. https://doi.org/10.3390/fi16050153

6. Le, D. C., & Zincir-Heywood, N. (2020). A frontier: Dependable, reliable, and secure machine learning for network/system management. *Journal of Network and Systems Management, 28*(4), 924-965. https://doi.org/10.1007/s10922-020-09556-2

7. Mohammed, A. R., Mohammed, S. A., & Shirmohammadi, S. (2019). Machine learning and deep learning based traffic classification and prediction in software defined networking. *IEEE International Workshop on Managing Networks and Services*, 1-8. https://doi.org/10.1109/IWMN.2019.8805044

8. Umukoro, I. I., Eke, B. O., & Edward, O. (2024). An efficient intrusion detection technique for traffic pattern learning. *Scientia Africana, 23*(2), 25-40. https://doi.org/10.4314/sa.v23i2.3

9. Ramesh Babu, C., Farook Ali, M., Pasha, A., Arshad, Z., & Asif, M. (2022). Utilizing machine learning techniques to classify network traffic. *Journal of Algebraic Statistics, 13*(3), 2402-2409. https://publishoa.com/issue/JSAS/13/3/2402

10. Dawood, A. S. (2023). Performance Evaluation of Machine Learning Naive Bayes Algorithms for Network Traffic Classification. *Technium Vol. 13*, 12–26.

11. Auld, T., Moore, A. W., & Gull, S. F. (2007). Bayesian neural networks for internet traffic classification. *IEEE Transactions on Neural Networks, 18*(1), 223–239. https://doi.org/10.1109/TNN.2006.883010

12. Butool, S., Yazdani, A., Ahmedulla, M., Nayeemuddin, M. K., & Ali, M. M. (2022). Utilizing machine learning techniques to classify network traffic. *International Journal of Health Sciences, 6*(S5), 5388–5395. https://doi.org/10.53730/ijhs.v6nS5.9822

13. Auld, T., Moore, A. W., & Gull, S. F. (2007). Bayesian neural networks for internet traffic classification. *IEEE Transactions on Neural Networks, 18*(1), 223–239. https://doi.org/10.1109/TNN.2006.883010

14. Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing and Management, 45*(4), 427–437. https://doi.org/10.1016/j.ipm.2009.03.002

15. Sadeghzadeh, A. M., Shiravi, S., & Jalili, R. (2021). Adversarial network traffic: Towards evaluating the robustness of deep-learning-based network traffic classification. *IEEE Transactions on Network and Service Management, 18*(2), 1962-1974. https://doi.org/10.1109/TNSM.2021.3052888

16. Dias, K. L., Pongelupe, M. A., Caminhas, W. M., & de Errico, L. (2019). An innovative approach for real-time network traffic classification. *Computer Networks, 158*, 143–157. https://doi.org/10.1016/j.comnet.2019.04.004

17. Li, X., Shi, G., & Wu, Y. (2023). Utilizing machine learning techniques for network traffic anomaly detection. *Proceedings of the 2023 International Conference on Machine Learning and Automation*. https://doi.org/10.54254/2755-2721/36/20230454

18. Sasidhar, T., Havisha, V., Koushik, S., & Reddy, V. K. (2016). Load balancing techniques for efficient traffic management in cloud environment. *International Journal of Electrical and Computer Engineering, 6*(3), 963–973. https://doi.org/10.11591/ijece.v6i3.7943

19. Wang, C., Xu, T., & Qin, X. (2015). Network traffic classification with improved random forest. *Proceedings of the 2015 11th International Conference on Computational Intelligence and Security (CIS)*, 78-83. https://doi.org/10.1109/CIS.2015.27

20. Chen, L., Liu, J., & Xian, M. (2020). Network traffic classification using deep learning. *International Journal on Artificial Intelligence Tools, 29*(7 & 8), 2040008. https://doi.org/10.1142/S0218213020400084

21. Shafiq, M., Yu, X., Laghari, A. A., Yao, L., Karn, N. K., & Abdessamia, F. (2016). Network traffic classification techniques and comparative analysis using machine learning algorithms. *2016 IEEE International Conference on Computer and Communications (ICCC)*, 2451–2455. https://doi.org/10.1109/CompComm.2016.7943943

22. Dai, Q., Zhang, C., & Wu, H. (2016). Research of decision tree classification algorithm in data mining. *International Journal of Database Theory and Application, 9*(5), 1-8. https://doi.org/10.14257/ijdta.2016.9.5.01

23. Gerka, A. (2018). Searching for optimal machine learning algorithm for network traffic classification in intrusion detection system. *ITM Web of Conferences, 21*(00027), 1-9. https://doi.org/10.1051/itmconf/20182100027

24. Tonkal, Ö., & Polat, H. (2021). Traffic classification and comparative analysis with machine learning algorithms in software-defined networks. *Gazi University Journal of Science Part C: Design and Technology, 9*(1), 71-83. https://doi.org/10.29109/gujsc.869418

25. Aouedi, O., Piamrat, K., & Parrein, B. (2022). Intelligent traffic management in next-generation networks. *Future Internet, 14*(2), 44. https://doi.org/10.3390/fi14020044

26. Beissenova, G., Zhidebayeva, A., Kopzhassarova, Z., Kozhabekova, P., Myrzakhmetova, B., Kerimbekov, M., Ussipbekova, D., & Yeshenkozhaev, N. (2024). Load balancing in DCN servers through software-defined network machine learning. *International Journal of Advanced Computer Science and Applications, 15*(2), 509-519. https://doi.org/10.14569/IJACSA.2024

27. Vimalarani, C. I., Bhuvaneshwari, N., Anushyadevi, V., Sriharini, V., Sangeetha, S., & Swarnapriya, S. (2024). Network traffic classification and reduction with load rebalancing. *ESP Journal of Engineering & Technology Advancements (ESP-JETA)*, *Special Issue*, 45–58.

