A Machine Learning Approach To
Network Security Anomaly Detection

by

PRATEEK VERMA

A Project Report Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF ENGINEERING

In the Department of Electrical and Computer Engineering

University of Victoria

We acknowledge and respect the Lək̓ʷəŋən (Songhees and Xʷsepsəm/Esquimalt) Peoples on whose territory the university stands, and the Lək̓ʷəŋən and W̱SÁNEĆ Peoples whose historical relationships with the land continue to this day.

A Machine Learning Approach To

Network Security Anomaly Detection

by

PRATEEK VERMA

SUPERVISORY COMMITTEE

Prof. Hong-Chuan Yang, Supervisor
(Department of Electrical and Computer Engineering)

Prof. Fayez Gebali , Departmental Member
(Department of Electrical and Computer Engineering)

ABSTRACT

Supervised machine learning has emerged as a highly effective technique for classification in anomaly-based cyber-threat detection systems due to its predictability, and high accuracy. This work utilizes the CICIDS2017 dataset which is widely recognized as a benchmark for anomaly detection research. The work begins with the idea to implement a two-layered ML-based detection model. The proposed system's first layer performs binary classification to differentiate benign from malicious traffic, while a secondary, multi-class classification system identifies specific attack types to implement targeted countermeasures. Incremental Principal Component Analysis (PCA) technique and Synthetic Minority Oversampling (SMOTE) is applied to balance the dataset, critical for both binary and multi-class classification tasks. Among all evaluated machine learning models, LightGBM achieved superior performance with 99% accuracy, 98.1% F1-score, and minimal resource usage, outperforming traditional methods like SVM, KNN, Random Forest and Decision Trees. Further feature reduction, guided by feature importance scores, led to an even more lightweight model while performance metrics such accuracy, recall, and F1-score, remained consistent or improved slightly within a margin of ±0.5% highlighting the stability and efficiency of the proposed approach. This proposed system demonstrates that advanced, resource-efficient supervised ML models such as LightGBM can significantly improve real-time threat detection while offering a scalable and cost-effective solution for future cybersecurity deployments.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my supervisor, Dr. Hong Chuan Yang, expressing gratitude for his unwavering support, encouragement and guidance throughout my learning journey. I am grateful for his commitment to my academic growth. Furthermore, I would like to thank my supervisory committee member Dr. Fayez Gebali, for their valuable support in the development of this project.

I am incredibly grateful to my family and partner for their unconditional support. Their encouragement and belief have been a constant source of motivation.

LIST OF ACRONYMS

AE Autoencoder

ADS Anomaly Detection System

CV Cross Validation

CNN Convolutional Neural Network

CICIDS2017 Canadian Institute for Cybersecurity Intrusion Detection System 2017 Dataset

DDoS Distributed Denial of Service

DL Deep Learning

DoS Denial of Service

DT Decision Tree

EFS Ensemble Feature Selection

FTP File Transfer Protocol

IDS Intrusion Detection System

MLP Multi-Layer Perceptron

PCA Principal Component Analysis

PSH Push (flag in TCP packets)

RL Reinforcement Learning

ROC Receiver Operating Characteristic

SMOTE Synthetic Minority Oversampling Technique

UDP User Datagram Protocol

URG Urgent (flag in TCP packets)

XGBoost Extreme Gradient Boosting

CHAPTER 1: INTRODUCTION

In the modern, rapidly evolving cyber-security landscape, the complexity and volume of network threats are increasing [1], rendering traditional rule-based systems insufficient [2] and vulnerable. As attack patterns grow more dynamic, a growing need for intelligent and adaptive detection mechanisms is observed. This work presents a machine learning-based approach to anomaly detection using the CICIDS2017 dataset. In addition to deploying advanced models like LightGBM associated with the study, a range of classical machine learning algorithms (including Random Forest, Decision Tree, and Logistic Regression) were also trained and evaluated to ensure fair performance comparison. The result is a highly efficient, accurate, and scalable intrusion detection framework suited to real-world applications. Additionally, this work distinguishes itself from existing research in several innovative ways:

Addressing Key Gaps in Existing Studies: This study helps in bridging critical methodological gaps found in earlier machine learning-based anomaly detection studies by introducing structured feature reduction, class balancing, and detailed performance evaluation.

Efficient Feature Engineering: Unlike previous models that use all features, this study applies Principal Component Analysis (PCA) to reduce dimensionality, eliminate redundancy, and retain meaningful patterns—leading to faster and more generalizable models.

Balanced Learning with SMOTE: To counter severe class imbalance in the CICIDS2017 dataset, the Synthetic Minority Oversampling Technique (SMOTE) is applied on the training data, ensuring fair learning across all the attack categories.

Use of Scalable, High-Performance Algorithms: The model leverages modern, scalable algorithms such as LightGBM, which deliver high classification accuracy and efficiency making the approach computationally practical and efficient.

Dimensionality Reduction for Efficiency: Using feature importance scores, only the most impactful features are retained from the PCA-enhanced dataset, further reducing the input dimensionality. This saves computational resources and reduces training time without compromising performance of the proposed system.

1.1 Motivation

The motivation behind this work is to develop a high-performing anomaly detection system by leveraging optimized feature selection and lightweight yet powerful machine learning algorithms. By incorporating these machine learning tools, organizations and cybersecurity firms can reduce over-reliance on traditional signature-based detection methods and outdated legacy systems. In the context of this work, anomalies are simply presented as network security anomalies, which are behaviours indicating malicious intent resulting in cyberattacks. In a general sense, anomaly is defined as a datapoint or pattern that deviates significantly from the expected behaviour. As a result, anomaly detection is distinct from attack detection because an anomaly broadly refers to any deviation from normal behaviour which could include benign or unusual traffic patterns, whereas attack detection specifically targets malicious activities. With the help of the proposed system, detection accuracy and adaptability is increased which further contributes to reduced operational costs, minimized system downtime, along with lower maintenance overhead.

1.2 Related Work

The work carried out by Elmrabit, N., Zhou, F., Li, F. and Zhou, H. investigates the effectiveness of the anomaly detection model by evaluating and developing twelve machine learning algorithms. The work included both classical models and deep learning methodologies to detect cyber anomalies across three different benchmark datasets, including CICIDS2017. In the context of the benchmark dataset, network data refers to the collection of flow-level information which includes flow duration, packet counts, and statistical features derived from the packets transmitted during each network connection or session. The focus of this work lies on classifying the network data into either binary or multi-class labels based on different attack scenarios. While the results reveal that Random Forest consistently outperforms other models in terms of accuracy, precision, recall, and F1-score, the paper lacks detailed implementation insights [7]. Additionally, no mention of dimensionality reduction, class balancing techniques such as SMOTE or manual balancing, or computation efficiency showcases a few shortcomings of the proposed work. Despite using a high-performance computing system, the work does not quantify model training or processing overhead.

Another work presents several notable advantages, including its focus on enhancing AdaBoost-based Intrusion Detection Systems (IDS). The authors effectively address the problem of imbalanced training data by employing Synthetic Minority Oversampling Technique (SMOTE), Principal Component Analysis (PCA) and Ensemble Feature Selection (EFS) for robust feature selection, displaying improved performance metrics. However, the work exhibits several drawbacks [3]. A significant limitation is the absence of resource utilization metrics which makes it difficult to understand the practical efficiency. Also, the scope of attack detection is narrow as the analysis is confined solely to DDoS attacks. This leads to neglecting other prevalent attack categories present within the CICIDS2017 dataset. The work also lacks a broader comparative analysis against other machine learning classifiers on the same dataset. These limitations raises concerns regarding the generalizability and detection capabilities of the proposed work for a wider range of cyber-threats.

Maseer, Z.K., Yusof, R., Bahaman, N., Mostafa, S.A., and Foozy, C.F.M. benchmark ten machine learning algorithms for Anomaly-Based Intrusion Detection System using the reputed CICIDS2017 dataset, featuring both unbalanced and multi-class traffic. A key advantage is the comprehensive evaluation of various algorithms, with Decision Trees and Random Forest achieving high accuracy of >99% for detecting diverse attack types. its decision to reduce the dataset to only four classes (C1–C4: BENIGN, Brute Force, XSS, and SQL Injection) significantly undermines its effectiveness. The original dataset contains a wide variety of real-world attack types such as DoS, DDoS, Botnet, Infiltration, Heartbleed, and others, which are entirely absent from the evaluation and results [4]. This oversimplification not only limits the real-world applicability of the models but also leads to misleading performance metrics, as high accuracy is achieved on an unrepresentative subset of attacks. Moreover, it underscores the actual challenge of multiclass classification in intrusion detection.

Similarly, the proposed work developed by Ustebay, S., Turgut, Z. and Aydin, M.A. focuses solely on binary classification and relies heavily on a reduced set of just four features selected via Recursive Feature Elimination (RFE) with Random Forest, followed by a Deep MLP classifier [5]. While the study achieves respectable accuracy (89–91%), it exhibits critical shortcomings,

most notably the oversimplification of the original CICIDS2017 dataset. It reduces the rich multiclass nature of the data into a binary setup (Benign vs. DDoS), effectively ignoring a wide spectrum of prevalent threats such as Botnet, Web Attacks, Infiltration, and Heartbleed. The practical value associated with the trained model is thus diminished as it is not robust against varied real-world cyberattacks.

The LSTM-AE work presents hybrid architecture combining Long Short-Term Memory (LSTM) and AutoEncoder (AE) enables effective learning of both temporal and structural patterns in network traffic. Additionally, it utilizes realistic and comprehensive datasets such as CICIDS2017 and CSE-CICIDS2018 [6]. The model demonstrates strong empirical performance, achieving accuracy rates up to 99.6% on CICIDS2017 and 98.9% on CSE-CICIDS2018, outperforming baseline models like CNN (96.2%) and DNN (95.5%). While it reports high accuracy, the results may not be entirely reliable due to a few methodological gaps. The absence of any sort of feature engineering, class balancing or validation techniques raises a few concerns associated with performance on imbalanced dataset. Another notable shortcoming of the LSTM-AE paper is its lack of multi-class classification analysis, despite using benchmarked datasets that contain a wide variety of attack types. The model is primarily evaluated in a binary system, with no breakdown of the performance across individual attack classes such as DDoS, Brute Force, or Web Attacks.

1.3 Objectives of the Work

The primary goal of this study is to develop an effective machine learning-based anomaly detection system that can classify and respond to cyber threats in real-time. The specific objectives are:

- **Develop a Binary Classification Model**: Build a robust ML model to distinguish between *benign* (normal) and *malicious* (threat) network traffic using the CICIDS2017 dataset.

- **Implement Multi-Class Classification for Attack Typing**: Once a certain threat is identified, classify the specific type of cyber-attack (e.g., DoS, DDoS, Brute Force, Port Scan, etc.) to enable targeted defence mechanisms.

- **Compare Performance of Multiple ML Algorithms**: Evaluate and benchmark algorithms such as Support Vector Machines (SVM), Random Forest, Decision Tree, Logistic Regression, and LightGBM to determine the most effective model for anomaly detection.

- **Enable Quick, Data-Driven Decisions**: Integrate advanced ML models that make real-time decisions to ensure proactive threat prevention in dynamic and unpredictable network environments.

1.4 Report Structure

The report structure is outlined below.

Chapter 1: Introduction

This chapter gives a quick overview of the work along with related work and motivation. The unique and innovative methods associated with the work are also discussed.

Chapter 2: Overview of Machine Learning

This chapter provides a comprehensive overview of supervised learning, including both binary and multi-class classification approaches. Additionally, it outlines the machine learning tools and libraries employed in this study.

Chapter 3: Methodology

This chapter details all preprocessing steps, including data cleaning, imputation, and feature engineering techniques such as PCA and SMOTE. It also provides an in-depth discussion of the various machine learning methodologies applied in this study, with a particular focus on the LightGBM algorithm.

Chapter 4: Performance Evaluation and ML Model Analysis

This chapter explains the performance metrics used to evaluate model performance. Also, discussions related to the results of different ML techniques, comparing their effectiveness in detecting cyber-attacks are mentioned.

Chapter 5: Conclusions and Future Work

This chapter summarizes the main findings and conclusions associated with study while suggesting future research for further improvements.

CHAPTER 2: OVERVIEW OF MACHINE LEARNING

2.1 Supervised Learning

The studies and research pertaining the to the domain of anomaly detection employ a variety of methodologies such as supervised learning, unsupervised learning, and semi-supervised learning. Another form of machine learning method is sometimes employed, known as Reinforcement Learning. Although reinforcement learning methods are highly applicable to complex real-world problems, they are not widely sought after due to the extreme resource consumption associated with vast state-action spaces.

Supervised Learning techniques have given out the most stable results along with excellent performance as reflected by a variety of research papers published. In this entire SL process, the classification technique segments the input data into discrete categories and calculates the probability of a test sample to be under each category [9]. The one with the most votes wins, and that becomes the probability of a sample or instance belonging to that class.

One of the key advantages of Supervised Learning (SL) lies in its **predictability and transparency**. Because the model is trained with clear input-output mappings, it is easier to interpret and evaluate the model's behaviour. supervised models generally require **less time to converge** during training compared to RL, which involves trial-and-error interactions with an environment, often resulting in slow learning curves and convergence issues.

2.2 Binary Classification

Binary classification is a fundamental task in machine learning where the goal is to categorize data into one of two distinct classes. This type of classification is particularly useful in scenarios where decisions must be made between two options, such as "yes" or "no" or "positive" or "negative". In a machine learning context, binary classification involves training a model to distinguish between these two classes using labeled data. For example, in medical diagnostics, a binary classification model could be trained to detect breast cancer. Here, the two classes would typically be benign (non-cancerous) and malignant (cancerous). The model

learns from historical data to make predictions about new, unseen cases, helping medical professionals assess the likelihood of cancer in a patient.

In this work, binary classification is extensively used in cybersecurity and anomaly detection based on the CICIDS2017 dataset commonly used for training models to detect cyber threats. The model classifies traffic as either safe/benign or abnormal/threat. If the system identifies unusual patterns or behaviors based on the learned features, it labels the input as a potential threat; otherwise, the traffic is considered safe.

2.3 Multi-class Classification

Multi-class classification is a type of machine learning task where the objective is to categorize input data into one of three or more distinct classes. Unlike binary classification, which deals with only two outcomes, multi-class classification is used when the problem involves multiple possible labels. A working example is image classification of handwritten digits (0–9) using the MNIST dataset [8], where the model learns to assign each image to one of ten-digit classes.

In cybersecurity applications such as the current study, multi-class classification is crucial for identifying various types of threats rather than just detecting whether traffic is normal or malicious. While utilizing in the CICIDS-2017 dataset traffic is categorized into multiple attack types such as Benign, DoS (Denial of Service), DDoS (Distributed Denial of Service), Botnet, Web Attack, Brute Force, and Port Scan.

A machine learning model trained on this dataset can predict the specific nature of an observed network activity, enabling more precise threat analysis and tailored responses. Specific countermeasures can also be enabled to neutralize the cyber-threat in a real-time application or network environment.

2.4 Overview of Machine Learning Tools

For the development and implementation of the machine learning-based anomaly detection system, a wide range of Python-based tools and libraries were utilized. These advanced toolkits provided the foundation for data manipulation, visualization, modelling and deployment.

NumPy

NumPy is a core Python library used for numerical computing. It provides powerful array and matrix data structures, along with a wide range of mathematical functions to manipulate data structures efficiently. In this study, NumPy was extensively used to handle numeric operations, support feature scaling, and interact with other ML libraries. One of its biggest advantages is its speed and efficiency with large datasets due to vectorized operations [10].

Pandas

This is an essential data analysis and manipulation tool in Python. It allows for structured data handling using DataFrames functionality, making it easy to load, clean, explore, and transform datasets. In this work, Pandas was crucial for organizing the original, raw dataset which effectively handled missing values, labelled attack types, and prepared the features for machine learning models. [11] The underlying high-level functions associated with this library is extremely valuable for preprocessing large-scale network traffic data.

Seaborn

Seaborn is a data visualization library built on top of Matplotlib, offering a high-level interface for drawing informative and attractive statistical graphics. In the project, Seaborn was used to visualize feature distributions, correlation heatmaps, and attack patterns. It is particularly useful for exploring relationships in complex datasets and also supports a wide variety of plots. Aesthetic styling is another advantage when presenting results in a polished format.

Matplotlib

This is a fundamental plotting library in Python that supports the creation of static and interactive plots. It was used in this study to generate custom visualizations like pie charts, bar graphs along with performance comparison plots. The fine-grained control over plot elements helps in developing high-quality figures which is beneficial for both exploratory data analysis and presentation.

Scikit-learn (Sklearn)

Scikit-learn is a versatile library specifically used for machine learning in Python. It provides tools for classification, regression, clustering, dimensionality reduction, model selection, and evaluation metrics. In this study, Sklearn was used to train and evaluate models such as Random Forest, Logistic Regression, Decision Trees, and to apply preprocessing techniques [12]. Its simplicity and comprehensive documentation make it an ideal choice for rapid ML development.

LightGBM Toolkit

Light Gradient Boosting Machine is an advanced gradient boosting framework developed by Microsoft. It is optimized for performance and scalability especially with large datasets. In this project work, LightGBM was employed as one of the primary classifiers due to its high speed, low memory usage and ability to handle imbalanced data efficiently. It also supports parallel and GPU learning, making it a strong candidate for real-time network threat detection and anomaly detection.

Google Colaboratory

It is a cloud-based development platform that provides a Jupyter Notebook environment with free access to GPUs and TPUs on the fly. It was used as the primary coding environment in this project, offering flexibility, ease of sharing, and zero setup time. Google Colab also integrates well with Google Drive and also enables execution of heavy ML methodologies without relying on local computational resources.

CHAPTER 3: METHODOLOGY

3.1 Dataset Exploration

The **CICIDS2017** dataset is a widely used benchmark dataset used in the field of cybersecurity, network intrusion and anomaly detection. It was developed by the Canadian Institute for Cybersecurity (CIC), designed to reflect and simulate real-world network traffic conditions [13]. This data incorporated both benign and malicious instances in a controlled environment. The dataset was collected over a span of seven days, each simulating different attack scenarios in a realistic network environment using tools like Zeek, Argus, and Wireshark.

It includes attacks such as Brute Force (FTP and SSH), DoS (Denial of Service), Heartbleed (SSL vulnerability), Web Attacks (like SQL injection and XSS), Infiltration, Botnet behaviour, and Distributed Denial of Service (DDoS). These diverse spectrum of cyber-attacks makes CICIDS2017 a versatile dataset for both binary and multi-class classification tasks. Statistically, the dataset consists of 2,830,743 rows and 79 columns. Out of these 79 features, 78 are numerical features derived from packet-level and flow-level characteristics. These columns capture important statistical behaviours of the traffic flow, crucial for the detection.

Using only the CICIDS2017 dataset did not result in overfitting, as several techniques and processes were undertaken to enhance model generalization. These included dimensionality reduction through Principal Component Analysis (PCA), class rebalancing using the Synthetic Minority Over-sampling Technique (SMOTE), appropriate data normalization, and a 75:25 train-test split. Furthermore, model performance was validated using multiple evaluation metrics and 5-fold cross-validation.

The dataset was employed with a relevant subset extracted for focused analysis. The dataset was split in a 75:25 ratio for training and testing along with 5-fold cross-validation. This was applied to validate model performance and reduce bias. All experiments were conducted on Google Colab using the most recent libraries such as Python 3.10, scikit-learn 1.4.2, pandas 2.2.1, seaborn 0.13.2, and NumPy 1.26.4. These tools were used for model development, data preprocessing, and performance visualization.

FEATURE NAME	DESCRIPTION
Flow Duration	Total time that a network flow remains active, measured from the arrival of the first packet to the last packet within the same flow.
Total Fwd Packets	Total number of packets sent in the forward direction (from source to destination)
Total Backward Packets	Total number of packets sent in the reverse direction (from destination to source)
Total Length of Fwd Packets	Sum of the payload lengths (in bytes) of all forward (source to destination) packets
Total Length of Bwd Packets	Sum of the payload lengths (in bytes) of all backward (destination to source) packets
Fwd Packet Length Max	Maximum payload length (in bytes) among all forward packets in a flow
Fwd Packet Length Min	Minimum payload length (in bytes) among all forward packets in a flow
Fwd Packet Length Mean	Average payload length (in bytes) of all forward packets in a flow.
Fwd Packet Length Std	Standard deviation of the payload lengths of all forward packets in a flow, indicating variation in size.
Bwd Packet Length Max	The maximum payload length (in bytes) among all backward packets in a flow
Bwd Packet Length Min	The minimum payload length (in bytes) among all backward packets in a flow
Bwd Packet Length Mean	The average payload length (in bytes) of all backward packets in a flow.
Bwd Packet Length Std	The standard deviation of the payload lengths (in bytes) of all backward packets in a flow, indicating variation in size.
Flow Bytes/s	The average number of bytes transmitted per second throughout the network flow
Flow Packets/s	The average number of packets transmitted per second during the flow

Flow IAT Mean	The average inter-arrival time between packets in the flow
Flow IAT Std	The standard deviation of inter-arrival times between packets in the flow
Flow IAT Max	The maximum time gap between any two packets in the flow
Flow IAT Min	The minimum time gap between any two packets in the flow
Fwd IAT Total	Total time duration between packets sent in the forward direction
Fwd IAT Mean	Average inter-arrival time of forward packets
Fwd IAT Std	Standard deviation of the inter-arrival time of forward packets
Fwd IAT Max	Longest inter-arrival time between forward packets
Fwd IAT Min	Shortest inter-arrival time between forward packets
Bwd IAT Total	Total time duration between packets sent in the backward direction
Bwd IAT Mean	Average inter-arrival time of backward packets
Bwd IAT Std	Standard deviation of the inter-arrival time of backward packets
Bwd IAT Max	Longest inter-arrival time between backward packets
Bwd IAT Min	Shortest inter-arrival time between backward packets
Fwd PSH Flags	Total number of times the PSH (Push) flag was set in forward packets
Bwd PSH Flags	Total number of times the PSH flag was set in backward packets
Fwd URG Flags	Number of times the URG (Urgent) flag was set in forward packets
Bwd URG Flags	Number of times the URG flag was set in backward packets
Fwd Header Length	Total length (bytes) of headers in forward packets

Bwd Header Length	Total length (bytes) of headers in backward packets
Fwd Packets/s	Average rate of packets per second sent in the forward direction
Bwd Packets/s	Average rate of packets per second sent in the backward direction
Min Packet Length	Length of the smallest packet in the flow
Max Packet Length	Length of the largest packet in the flow
Packet Length Mean	Average packet length in the flow
Packet Length Std	Standard deviation of packet lengths in the flow
Idle Mean	Average time (in microseconds) the flow remained idle between packet arrivals
Init_Win_bytes_forward	Initial TCP window size in the backward direction
Init_Win_bytes_backward	Number of data packets in the forward direction before a PSH flag
act_data_pkt_fwd	Minimum segment size observed in forward packets
min_seg_size_forward	Minimum segment size observed in forward packets
Idle Std	Standard deviation of idle time periods in a flow
Idle Max	Maximum idle time period between packets
Idle Min	Minimum idle time period between packets
Label	Indicates whether the network flow is benign or belongs to a specific attack type

All features in the dataset, except for the 'Label' column, consist of numerical values. These numerical features were standardized as part of the preprocessing phase to ensure consistency and improve the performance of machine learning models. Standardization helps normalize the range of feature values, which is especially important when using algorithms sensitive to feature scale.

The *Label* column, on the other hand, is a categorical binary feature that indicates whether a data instance is classified as *Benign* or *Malicious*. While *Benign* represents normal, non-

threatening network traffic, the *Malicious* category encompasses a wide range of cyberattacks. The data also includes *source* and *destination IP addresses*. These are included as part of the network flow data, which is usually extracted from the raw PCAP (packet capture) files. The CSV files contain labelled flow, with information including timestamps, source and destination IPs, source and destination ports along with attack labels.

Possible limitations emerged after the analysis of individual network flow features from the CICIDS2017 dataset. This approach, while effective, limits the capture of crucial inter-domain dependencies and advanced temporal dynamics that characterize real-world cyberattacks. Future research aims to address these limitations by possibly exploring graph-based network analysis for relational insights. Additionally, leveraging sequential models such as Transformers or LSTMs could better model the temporal progression of attacks.

3.2 Types of Cyber-Attacks

The dataset chosen for this study includes a diverse range of cyber-attacks that reflect common threats faced in real-world network environments [14]. The primary attack categories featured in the dataset are Denial of Service (DoS) and Distributed Denial of Service (DDoS), with the primary objective to overwhelm network resources. Additionally, Port Scanning is used for probing vulnerable network ports; Brute Force attacks targeting login credentials over FTP and SSH; Web Attacks such as SQL injection and cross-site scripting; and Botnet activity, which represents coordinated malicious patterns and abnormal behaviours often used for large-scale attacks [15].

3.2.1 Denial-of-Service (DoS)

This type of attack floods the network or server with excess traffic, making a website or resource unavailable. As the server is flooded with more TCP/UDP packets than it can process, the server might crash or data may become corrupted. This attack happens from a single location and increases the resource's loading time exponentially [34]. DOS attacks are usually easier to detect and mitigate compared to their distributed counterpart. However, their

impact can be severe and fatal for the network system if the target system lacks failsafe mechanisms and proper defense systems.

Figure 3.1. Denial-of-Service [16]

Attackers may use packet flooding, ping of death, SYN flood, or application-layer DoS techniques, depending on the system's vulnerabilities. These attacks can result in data corruption, service unavailability, financial losses, and reputation damage, especially for organizations dependent on constant online availability.

3.2.2 Distributed Denial-of-Service (DDoS)

During this attack, multiple computers and machines try to flood a targeted resource or server. During the execution, remote machines can send larger amounts of traffic simultaneously [24]. The fundamental objective is to compromise and interrupt the existing services. DDoS can be deployed much faster than a DoS attack, making it extremely difficult to detect. Distributed Denial-of-Service (DDoS) is a more advanced and powerful form of Denial-of-Service attack in which multiple compromised systems are used to launch a coordinated flood of traffic toward a targeted server or online service.

The primary objective of a DDoS attack is to disrupt the normal functioning of services, overwhelm bandwidth and typically exhaust server resources [34]. Attack vectors can include volumetric attacks (UDP floods, DNS amplification), protocol-based attacks (SYN floods), and application-layer attacks that target web applications directly. Due to the distributed nature

and speed of deployment and execution, DDoS attacks can bypass traditional defence mechanisms.

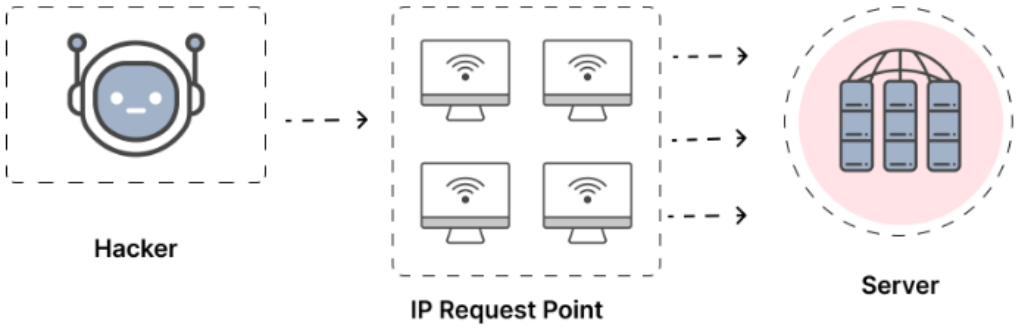

Figure 3.2. Distributed Denial-of-Service [16]

3.2.3 Port Scan

This network intrusion is used to discover weak points in a given network. Port scan allows the perpetrator to analyze the open port and figure out if the port is in receive mode or transmit mode. This attack can help discover the type of network packets sent to the destination port using various techniques [34]. *Several of these include Ping Scans, Vanilla Scan, SYN Scan and Sweep Scan.* Although, a port scan inherently does not cause network disruptions, however, it reveals vulnerabilities in the network.

3.2.4 Brute Force

This attack typically is a hacking technique that utilizes trial and error attempts to bypass passwords, discover login credentials and gain access to encryption keys forcefully. During this type of attack, network packets will exhibit several changes. Some of the changes might include an increase of login requests from same IP address, frequent failed authentication messages and a burst of traffic shortly before a successful brute force attempt. An SSH Brute Force attack typically involves the use of automated tools to bombard an SSH server with thousands of password combinations in a short span of time.

This generates an abnormal and distinct traffic pattern followed by a surge of short-lived connections on one of the ports, each involving a quick connection attempt, an authentication

failure, and a subsequent disconnection [34]. The network logs will reflect a high number of failed login attempts originating from the same IP address, which is often followed by a successful login if a correct credential pair is eventually guessed.

3.2.5 Web Attack

These hacking techniques are malicious attempts to exploit vulnerabilities and weak links in websites and web applications to gain unauthorized access in order to steal information, breach database or damage reputation of an organization. Common web attacks include *SQL injection*, *Cross-Site Scripting* and *Man-in-the-Middle.* During the SQL injection attack, attackers and hackers insert malicious SQL queries into input fields to manipulate backend databases [34]. Another major threat Cross-Site Scripting involves inserting malicious and harmful scripts into webpages viewed by other users, potentially allowing the predator party to hijack sessions or redirect users to a completely different webpage.

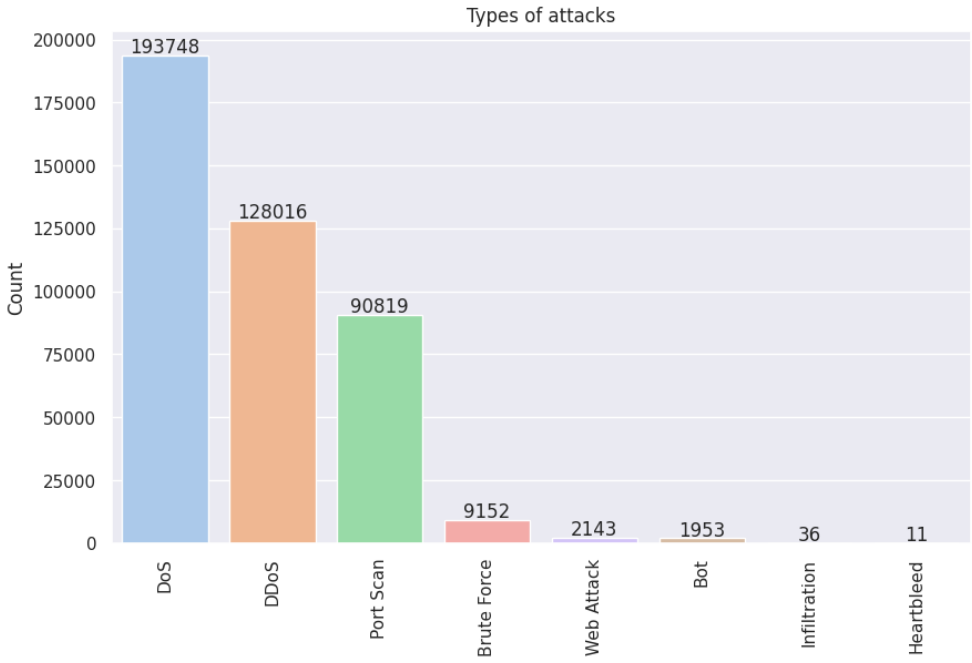

Figure 3.3. Network Threat Breakdown [35]

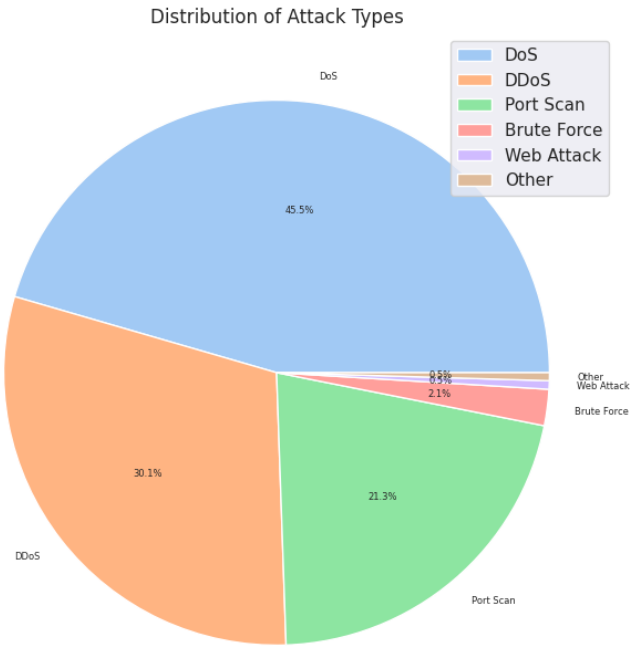

Figure 3.4. Cyber Threat Distribution [35]

The illustrations in Fig. xx3 and Fig. xx4 represents the breakdown and distribution of various cyber-attacks found in the selected dataset. As showcased in the bar chart and pie chart, DoS attacks are the most frequent, accounting for approximately 45.5% of the original dataset, followed by DDoS (30.1%) and Port Scan (21.3%). Other attack types such as Brute Force, Web Attacks, and Botnet represent a much smaller portion of the data.

3.3 Data Cleaning and Imputation

3.3.1 Managing Duplicate Values

The dataset had about 308327 duplicate rows which were later dropped and updated shape of the dataset with 2522362 rows and 79 distinct columns. This led to a reduction of about 10.9% of the original dataset.

3.3.2 Managing Missing Values

The dataset had 3128 missing values which were spread across two different columns: *Flow Byte/s, Flow Packet/s.* These two columns are continuous variables and were not following normal distribution [17]. As a result, the missing values were filled in by using the median as

it does not create any new categories. This further reduces any disruption in the original distribution of the dataset.

Table 3.1. Statistics Related to Missing Values

Feature	Count of Missing Values	Percentage of Total	Median Value
Flow Byte/s	1564	0.06	3714.098
Flow Packet/s	1564	0.06	69.876

3.4 Data Preprocessing

Data preprocessing refers to the implementation of several key techniques and processes to ensure that the dataset is well-structured and suitable for efficient machine learning methodologies. Correlation analysis was first used to identify and eliminate highly correlated features, helping reduce redundancy and improve model generalization. Principal Component Analysis (PCA) was applied to further reduce dimensionality by transforming the data into a set of uncorrelated principal components [36].

Furthermore, SMOTE (Synthetic Minority Oversampling Technique) was used on the training set to synthetically generate samples for underrepresented attack categories, ensuring balanced class representation [37]. Lastly, outlier analysis was conducted to detect and shed some light on the importance of anomalous data instances for the network intrusion and anomaly detection method.

3.4.1 Correlation Analysis

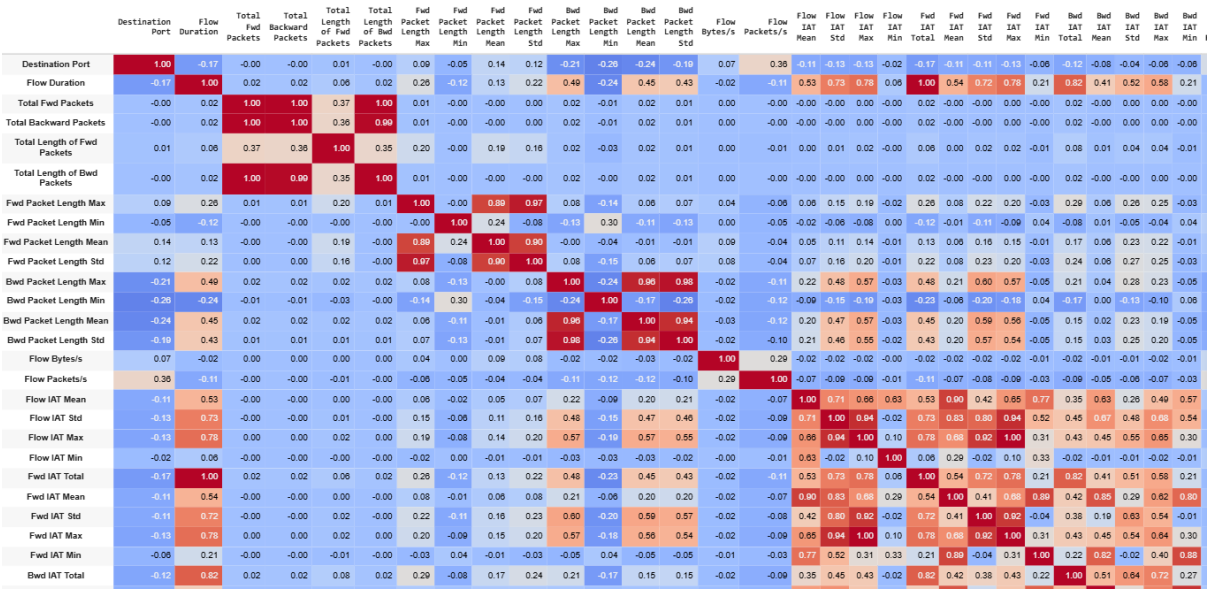

Fig 3.5. Correlation Matrix for 78 Features

The dataset clearly showcases several clusters of highly correlated features, particularly packet-related statistics such as *Total Fwd Packets, Total Length of Fwd Packets, Fwd Packet Length Max, Fwd Packet Length Mean*. Also, these correlations can be seen between various IAT (Inter-Arrival Time) metrics. These strong linear relationships indicate that many of the 78 original features are redundant or overlapping in the information they provide, as displayed in figure 3.5.

These redundant metrics provide some challenges to the development of machine learning models. For instance, the models trained on overlapping features may learn noise or overly specific patterns during the training phase. This could lead to less generalizability capabilities towards unseen network traffic. Additionally, it could introduce elements of overfitting which would be detrimental to the overall performance [18].

Apart from the overfitting concerns, the 78 features may require substantial processing and computation especially for models like K-NN, Logistic Regression or ensemble methodologies. This happens because high-dimensional datasets usually lead to higher training times, larger memory usage, and possess difficulties in debugging or visualization [19]. Thus, reducing the number of features to a smaller uncorrelated dataset could reduce the computation overhead while maintaining key information.

3.4.2 Normalization and Standardization

Standardization of the current dataset is carried out using *StandardScaler* from the *sklearn.processing* module. Standardization is a type of normalization technique that transforms the features so that they have a mean of zero and a standard deviation of one. This process is essential when dealing with machine learning models that are sensitive to the scale of input features.

The dataset is first separated into two parts: the feature set and the target variable, where 'Attack Type' column label is dropped and the attack classes are stored separately. Additionally, the *StandardScaler* is applied to the feature set using a function, which calculates the mean and standard deviation for each feature and uses these statistics to scale the data accordingly.

	Destination Port	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean
0	49188	4	2	0	12	0	6	6	6.00000
1	49188	1	2	0	12	0	6	6	6.00000
4	49486	3	2	0	12	0	6	6	6.00000
5	49486	1	2	0	12	0	6	6	6.00000
8	88	609	7	4	484	414	233	0	69.14286

Table 3.2. Non-Standardized Original Data

In the end, all features are brought to the same scale without distorting differences in the ranges or affecting any relationships between features. This process generally leads to faster convergence during the training phase and often results in better model performance.

	PC1	PC2	PC3	PC4	PC5	PC6	PC7
0	-2.357605	-0.054980	0.577367	0.734454	3.730371	0.234648	-0.015694
1	-2.883854	-0.069938	0.911321	1.763147	8.846136	0.620454	-0.056858
2	-2.416928	-0.056683	0.615335	0.850869	4.304213	0.275779	-0.020438
3	-2.884705	-0.069978	0.912183	1.765263	8.851561	0.618717	-0.057028
4	-1.505248	0.080886	-0.504396	0.290414	-0.539267	0.746103	0.100443

Table 3.3. Normalized Dataset

The original, unscaled network flow features where values differ significantly in both range and units utilized as illustrated by table 3.2. This variation can negatively impact machine learning models, especially those sensitive to feature scale. Subsequently, Table 3.3 shows the transformed features after applying normalization and PCA which resulted in a standardized scale that enhances training stability.

3.4.3 Principal Component Analysis (PCA)

Principal Component Analysis is a powerful dimensionality reduction method often used in machine learning and data preprocessing. The objective of this technique is to simplify a complex dataset by transforming it into a new set of variables, known as principal components. This transformation is achieved by identifying the *eigenvectors* associated with data's *covariance matrix*. These newly generated components are linearly uncorrelated because the eigenvectors are orthogonal, and they are ordered by their corresponding eigenvalues representing the amount of variance captured by each component.

In real-world datasets, it is common for features to carry redundant or overlapping information, which can complicate model training and reduce performance [20]. Principal Component Analysis (PCA) addresses this by reducing the number of features while retaining most of the data's variance and important information through the selection of components with the largest eigenvalues. By transforming correlated features into uncorrelated principal components, PCA effectively eliminates multicollinearity, which helps prevent overfitting by simplifying the model's structure.

The process of PCA can be broken down into simple and straightforward steps as illustrated in Fig 3.6. The correlation patterns are identified by computing either the covariance or correlation matrix, given the dataset is standardized. Positive covariance values indicate variables tend to increase or decrease together. Additionally, negative values indicate that variables move in the opposite directions while zero value indicates no linear relationship.

Eigenvectors: These define the directions of the new principal components.

Eigenvalues: Quantify the amount of variance experienced by each eigenvector.

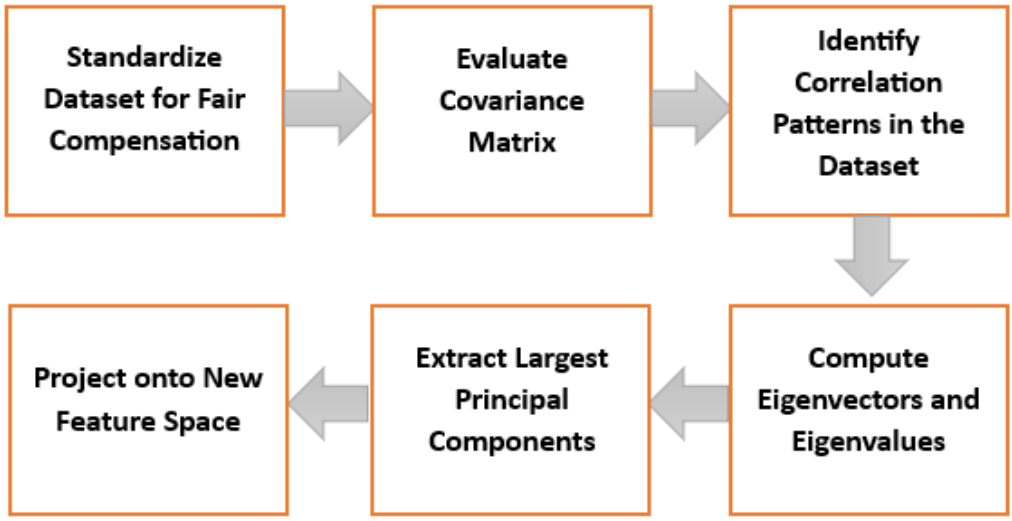

Figure 3.6. PCA Implementation Process

After applying the PCA method, original 78 features were successfully reduced to 35 independent, uncorrelated principal components. These components are statistically independent and completely uncorrelated, capturing the majority of the variance present in the original dataset while removing redundancy and noise. To simplify further analysis and model development, the original feature names were replaced with standardized aliases such as PC1, PC2, PC3, and so on, where each PC (Principal Component) represents a unique linear combination of the original features progressively ranked by their contribution to explaining variance in the data.

3.4.4 Synthetic Minority Over-sampling Technique (SMOTE)

The SMOTE technique at its core fabricates and improves a new 'synthetic' minority-class samples by the process of linear interpolation between a real minority point and one of its k-nearest minority neighbours. The new points are present inside the minority manifold rather than completely duplicating the same set of records and data instances. This leaner set of instances displays a richer and enhanced surface decision, which learns and adapts boundaries that are not biased towards the majority class.

This technique helps in reducing any sort of biases underlying in the original datasets. In multi-class classification tasks, the Synthetic Minority Oversampling Technique (SMOTE) is adapted to address class imbalance across multiple categories rather than just a single minority class. Instead of focusing on one underrepresented class as in binary classification, SMOTE identifies all classes with relatively low sample counts and treats them as minority classes [21,22]. To generate synthetic samples, it typically employs a one-vs-all strategy, ensuring that new points are interpolated only between samples of the same class, which helps maintain label integrity. Various balancing strategies can be applied, such as equalizing all class sizes, proportionally increasing minority class counts, or using a hybrid method that combines oversampling with under-sampling of dominant classes.

These adaptations ensure that SMOTE remains effective and flexible when applied to complex multi-class problems. During the development of this work, the utilization of SMOTE technique played a crucial role in addressing the highly imbalanced nature of the original CICIDS2017 dataset. The original data is heavily skewed, with over 2 million records associated with 'Benign' samples compared to only a few thousand in certain attack categories such as Web Attacks or Botnet. This kind of imbalance usually leads to a biased classifier that performs well on the dominant classes but poorly detects and typically fails while working with the critical but rare threats.

Attack Type	Instances (Count)
BENIGN	5000
DoS	5000
DDoS	5000
Port Scan	5000
Brute Force	5000
Web Attack	5000
Bot	5000

Table 3.4. Balanced Dataset for Multi-Class Classification

To tackle this problem, SMOTE was applied selectively to the training data which boosted the representation of minority classes through synthetic sample generation while down-sampling the majority classes such as BENIGN and DoS. This resulted in a balanced training dataset with 5,000 instances per class, as illustrated in Fig xx. As a result of this implementation, the classes with over 9000 instances were sampled down to 5000, whereas classes with less than 2500 instances were increased to 5000. This curated a balanced dataset for multi-class categorization.

Importantly, **SMOTE** was only applied to the **training set**, while the test set was deliberately kept untouched and imbalanced for the process of multi-class classification. This is a best practice as modifying the test set would distort performance evaluation. Preserving the original class distribution in the test set ensures that the model is evaluated under real-world conditions. This selective training sampling also prevents information leakage, where synthetic samples influenced by test data could falsely inflate performance metrics. Thus, metrics like accuracy, precision, recall, and F_1-score truthfully reflect how the model would perform in real-world deployment scenarios.

3.4.5 Outlier Analysis

This method gives an initial insight into the distribution of outlier dataset which is essential for development of effective anomaly and intrusion detection classification models. In the benchmarked cybersecurity datasets, outliers often represent abnormal or malicious patterns and behaviours. [23] Detecting and understanding these deviations reveals underlying patters that are often overlooked by the traditional feature-based analysis. Another important insight is related to the characteristics that distinguish cyber-threat traffic from normal traffic. High outlier percentages in certain features are valuable for anomaly detection, making them valuable for both binary and multi-class classification tasks.

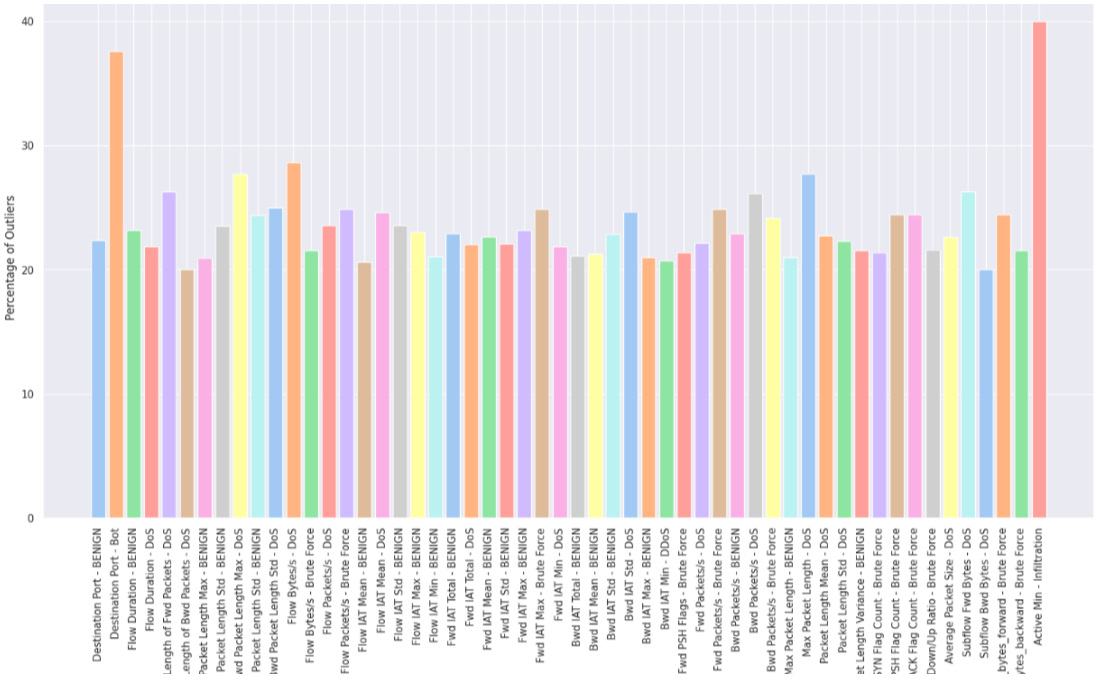

Fig 3.7. Outlier Analysis of Features based on Cyber-Attack Types

The bar graph in Fig 3.7, visualizes the percentage of outliers, usually greater than 20%, across multiple features present in the CICIDS2017 dataset. With the help of this feature engineering technique, some features were dropped reducing the initial set of 78 features to 70 features. The fact that many features have more than 20% outliers (even in BENIGN traffic) suggests strong variance, possible data imbalance or noise in the dataset.

3.5 Machine Learning Overview

A wide range of machine learning techniques were employed in this work, spanning from foundational models like *Logistic Regression* and *Decision Trees* to more advanced algorithms such as *LightGBM* and *XGBoost*. Different models were used for binary and multi-class classification tasks to best suit the nature and complexity of each problem. While binary classification aimed to distinguish between normal and malicious network traffic, multi-class classification required separating multiple specific attack types. Notably, algorithms such as Random Forest, LightGBM, K-Nearest Neighbours, and XGBoost were effectively applied in both settings due to their flexibility and strong generalization capabilities.

Logistic Regression

This is a supervised machine learning technique that is primarily utilized for classification problems. Unlike linear regression which predicts continuous values, this technique pushes out the probability that an input/input value belongs to a specific class [25]. This algorithm is usually used for binary classification where the output can be of the true possible categories. In this study, those two cases would be 'BENIGN' and 'THREAT' (for binary classification). The *Sigmoid* function is the important feature of logistic regression which is used to convert the raw output of the model into probability values between 0 and 1. In this regression technique, a threshold value of 0.5 is used to decide the class label.

- If the sigmoid output is same or above the threshold, input is classified as Class 1.
- If it is below the threshold, input is classified as Class 0.

1. WEIGHTED SUM

   For each network instance, the model computes a mathematical function:

$$z = w_1 x_1 + w_2 x_2 + \ldots + w_{35} x_{35} + b \tag{3.1}$$

   In this function, x_i is the value of the i^{th} feature while w_i is the learned weight for that particular feature. Additionally, the bias term *b* is also learned during the training process and contributes to the final output of the model. The bias acts as an offset that allows the model to make accurate predictions even when all input features are zero.

2. SIGMOID ACTIVATION

This process helps in converting the inputs into appropriate probabilities [26]. The input values are passed through the above-mentioned function *z* in order to get a value between 0 and 1.

$$\hat{y} = \frac{1}{1 + e^{-z}} \tag{3.2}$$

The output is interpreted as the probability of being a threat. For instance, if the output value is greater than 0.5 then network instance is classified as 'Threat'. Else, classification would be 'Benign'.

3.5.1 Random Forest

This technique combines predictions from multiple decision trees to provide a higher accuracy and stable results. During the classification tasks, Random Forest method predicts categorical outcomes based on the input values. This is facilitated by multiple decision trees which outputs the label that has maximum votes among all the individual tree predictions and outcomes. A process known as Bootstrap Aggregation is used where random rows are picked (with replacement) to train each decision tree. Due to this, each tree sees slightly different distribution of data which leads to increased variance and further reduces overfitting.

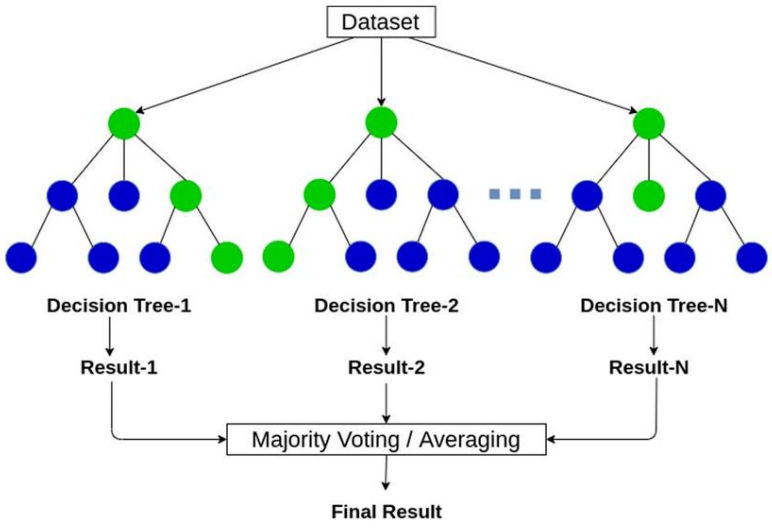

Fig 3.8. Multiple Decision Trees Prediction Model [27]

This diagram illustrates the working of a Random Forest model where multiple decision trees are trained on different subsets of the dataset. Each decision tree provides a prediction and the final result is determined by majority voting (for classification) or averaging (for regression) which further improves accuracy and reduces overfitting.

Each decision tree is developed using a process function called *feature subsampling,* where a random subset of features (typically square root of the number of features) is considered at every node split. This method introduces variance and diversity among trees which further helps in reducing overfitting conditions by preventing trees from becoming and growing similar. While a tree is being constructed, each tree splits the data at nodes based on a single feature that best separates the classes, using either Gini or Entropy based methods.

The trees are allowed to grow deep, with stopping criteria such as maximum depth or minimum number of samples per split to control complexity. Once all trees are trained, the Random Forest makes predictions by aggregating the outputs of individual trees. The final output is predicted using either a major voting technique or averaging predicted probabilities. This results in stable learning and predictions.

3.5.2 Decision Trees

This method is a graphical representation of different choices and possibilities with probable results that help in making decisions and final outcomes. The method follows a hierarchical tree structure with the root node at top which leads to branching out into different possible outcomes [28]. In this machine learning technique, *Root Node* is the starting point of the entire dataset.

- **BRANCH**: These help connect the nodes that shows the flow of one decision from another.
- **INTERNAL NODES:** These are the points where decisions are formulated based on the input features.

- **LEAF NODES:** These terminal nodes lie at the end of the branches that display final outcomes and predictions.

Decision Trees work on the dataset by evaluating all the PCA-enhanced 35 features such as *FlowDuration, PSHFlagCount, AvgPacketSize.* The features are chosen and a certain threshold is applied that separates the 'BENIGN' from 'THREAT' traffic based on Gini impurity or entropy. Recursive Splitting is applied where once a child node is formed, the tree again looks for the best feature and threshold to further separate the dataset. This continues recursively leading to a formation of decision tree where each step narrows down the possibilities. Once a certain node contains mostly one class, for instance 98% 'THREAT', the node changes to a leaf node. Any new data reaching that leaf during prediction is classified accordingly.

3.5.3 K Nearest Neighbour

This technique can also be termed as a lazy learning algorithm because it simply stores all the data instead of building a model from ground-up. Whenever a new unseen data instance is encountered by the algorithm, KNN predicts the class based on the classes of the most similar *'K'* instances present in the training set [29]. In this method, 'k' is a number that tells the model how many nearby points or neighbours to assess when making a prediction or decision. Additionally, this technique uses distance metrics to measure and identify the nearest data instance (neighbour) which would be utilized for classification tasks. Distance metrics such as *Euclidean (1.1), Manhattan (1.2)* and *Minkowski* are used to develop the final predictions.

$$\text{distance}(x, X_i) = \sqrt{\sum_{j=1}^{d} (x_j - X_{i,j})^2} \tag{3.3}$$

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i| \tag{3.4}$$

$$d(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}} \quad (3.5)$$

The *Minowski* distance (1.3) generalizes both the Euclidean and Manhattan distances as special cases. As displayed in the formula, for p=2 the distance becomes the same as the Euclidean distance and when p=1, it turns into the Manhattan distance formula. The KNN method starts off by a feature preprocessing step where all the features are scaled using a normalization function after feature reduction is applied on the initial data.

During the training phase, the algorithm stores all the labelled training points with their future values and class labels. Subsequently, during the prediction phase, for every new traffic instance the KNN method computes the distance between the given point and all the training points. The predicted class for that instance would be the majority class among those *'K'* neighbours. Since KNN is highly sensitive to irrelevant and redundant features, the dimensionality reduction from 78 features to 35 PCA-enhanced features helps the model arrive at improved outcomes.

3.5.4 Support Vector Machines

SVM is a supervised machine learning methodology that is used for both classification and regression tasks. This method lays on the foundation of finding the best boundary (also known as Hyperplane) that separates different classes in the dataset. It is highly useful in binary classification tasks and works best if only two classes are present. The primary objective of SVM is to maximize the margin between classes and all the associated data instances.

SVM can be divided into *Linear SVM* and *Non-Linear SVM*. Linear SVM works the best when data is clearly separable without any major effects from the outliers. Whereas, Non-Linear SVM is utilized in a dataset which is not easily separable. During the non-linear process, kernel tricks are used to map the data into higher dimensions such that a separation is introduced between the two classes.

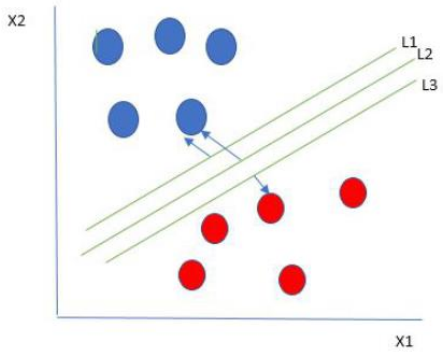

Fig 3.9. Hyperplane Separation using SVM Technique [30]

The dataset utilized in this work was first feature-reduced and normalized using only 35 features, since SVM is sensitive to the scale of features. During the training phase, the algorithm searches for the maximum-minimum hyperplane such that distance between the chosen hyperplane and the closest points from both classes (support vectors) are maximized. Also, a kernel function was utilized to project the data instances to higher dimensions where linear separation was easily achieved. The SVM method is well suited for handling outliers and anomalies because it defines a clear boundary around normal (BENIGN) behaviour.

3.5.5. XGBoost

This technique is a high-performance algorithm optimized for speed and efficiency. These boosting techniques are suitable for large datasets and tasks that require fast predictions and robust outcomes. This algorithm can also handle missing values and data points without requiring any special pre-processing, thus simplifying data pre-processing pipeline. The methodology is predominantly built on gradient boosting architecture which provides high accuracy and scalability. It can also be understood as a framework which supports multiple languages for machine learning processes [31]. This technique usually outperforms other boosting methodologies and ensemble learning algorithms both on execution speed and performance parameters.

For the current learning process associated with model training, data preprocessing is applied on the CICIDS dataset. Feature Reduction, Label Encoding and Sparse Data Handling techniques are applied which successfully converts categorical labels [32] such as ‘BENIGN’, ‘DoS’, etc. into numeric classes (1,2,3...).

Execution Speed	Performance Parameters
Parallelization: XGBoost is designed to utilize maximum computational resources during model training through efficient parallel processing, significantly reducing training time.	*Regularization:* This parameter activates automatically when model is in the training phase, leading to generalization and reducing the risk of overfitting.
Cache Optimization: Intermediate calculations and statistics are stored in cache for frequent access leading to lower execution times and faster model convergence.	*Auto-Pruning:* This feature halts the learning process of the algorithm at a certain threshold. Bias and variance are evaluated effectively during the training process.
Out of Core: This feature enables the algorithm to process datasets larger than the available RAM by efficiently splitting and streaming data from disk.	*Sparse Values Handling:* If some values are missing in the dataset, XGBoost uses different statistical imputations to ensure robust performance even with incomplete data.

Table 3.5. Enhanced Features Supporting XGBoost Learning

During the subsequent steps, XGBoost builds an ensemble of decision trees in a sequential manner where every new tree tries to correct the errors of the previous ones. After each round, the overall model is updated by fitting to the residuals (errors) of previous predictions instead of the original labels.

$$\mathcal{L} = \sum_i l(y_i, \hat{y}_i) + \sum_k \Omega(f_k) \tag{3.6}$$

To prevent overfitting, XGBoost constantly applies L1 and L2 regularization (1.4) to the primary objective function.

3.6 LightGBM: An Optimized Approach for High-Performance Anomaly Detection

This is a gradient boosting technique where an ensemble of multiple weak learners is used to improve the performance of the model. The basic hypothesis involves filtering out the instances which are difficult to accurately predict and replace those with new weak learning methods to handle them effectively. Fig. 3.12. highlights the core attributes that set LightGBM's methodology apart from conventional supervised learning techniques

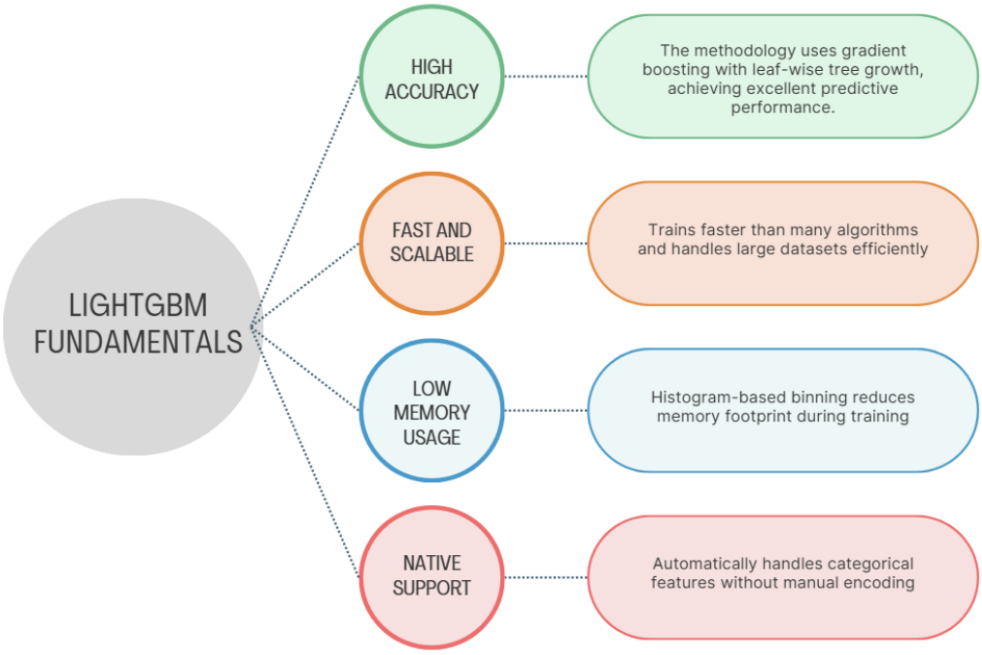

Fig. 3.10. Key Advantages of LightGBM for Classification

LightGBM is a superior learning method due to the histogram-based learning technique which performs bucketing of values. The methodology is highly compatible with extremely large, complex datasets and is much faster during the training phases. LightGBM performs a leaf-wise vertical growth that results in better loss reduction, which leads to higher performance both in terms of accuracy and precision. It primarily uses two novel techniques which helps in maintaining good accuracy levels.

LightGBM's algorithmic design and efficiency make it especially well-suited for high-dimensional and complex cybersecurity datasets. The following core features plays a critical role in its success:

1. Leaf-Wise Tree Growth Strategy

LightGBM builds trees by expanding the leaf with the maximum information gain, rather than growing level-by-level This process is clearly illustrated in the Fig. 3.13. The leaf-wise growth strategy is particularly beneficial when working with CICIDS2017, where attack patterns such as Botnet or Infiltration are rare and subtle.

- **Impact:** This method allows the model to focus more deeply on complex threat patterns, leading to higher accuracy.
- **Example:** Instead of asking every student inside a classroom the same set of questions, a smart way would be to identify which students are confident and which students are already struggling. Then, the teacher could focus their time and resources on the students who are unsure or making mistakes. Similarly, LightGBM prioritizes learning from the most confusing or error-prone data points (those with high gradient values), allowing the model to improve faster by concentrating on the areas that need the most attention.

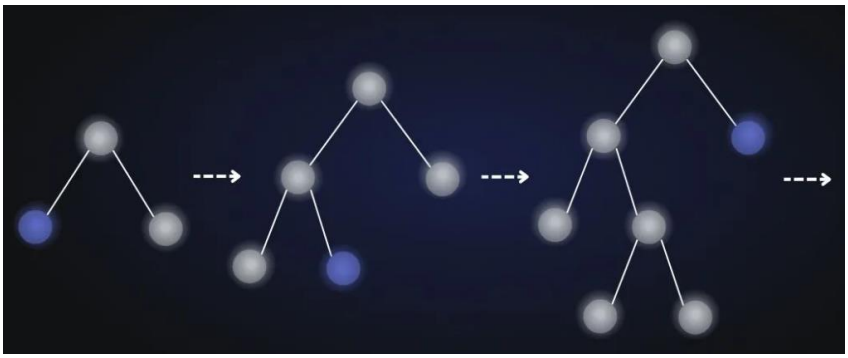

Figure 3.11. Horizontal Tree Growth (Leaf-Wise) [33]

2. Gradient-Based One-Side Sampling (GOSS)

The CICIDS2017 dataset is heavily imbalanced, with millions of benign instances and relatively few samples for certain attacks. GOSS addresses this by retaining all “difficult” samples, those with large gradients, while randomly sampling the “easier” ones.

- **Impact:** The model spends more time learning from ambiguous or rare threat data, such as Web Attacks or Botnet, while boosting precision and recall without unnecessary computation.
- **Example:** In a classroom, a teacher is preparing students for an exam. Some students already understand the material well and consistently get the answers right. Whereas, others are making mistakes and clearly need more help. Instead of spending equal time with every student, the teacher focuses revision sessions on those who are struggling which helps them improve faster. On a similar note, LightGBM, through Gradient-Based One-Side Sampling (GOSS), spends more time learning from difficult or confusing samples (e.g., rare attacks in the dataset) and less time on easy-to-classify threats such as Benign traffic.

3. Histogram-Based Binning

LightGBM uses a novel technique called *histogram-based decision tree learning* to optimize memory efficiency. Instead of working with raw continuous numerical features directly (like Flow Duration or Packet Length Std from the dataset), it first discretizes these values into a fixed number of bins, formulating a histogram.

- **Impact:** This binning approach significantly reduces computational complexity and memory usage, allowing LightGBM to train faster and scale to larger datasets. Additionally, it helps smooth out minor variations or noise in the data.

- **Example:** While analyzing monthly bank statements, a person could examine every individual transaction, to understand the spending behaviour. While this gives a detailed breakdown, the process is time-consuming and inefficient. Instead, a more practical approach is to group your expenses into broad categories like Rent, Groceries, Transportation, Utilities, and Miscellaneous.

  This is similar to what LightGBM does with histogram-based binning. Rather than evaluating every raw value of a continuous feature (like packet duration or byte count), it divides the feature into a fixed number of bins (e.g., 255 intervals). Each data point is then mapped to its corresponding bin. This technique reduces the number of unique values to be processed by the algorithm, which further facilitates the focus towards broader patterns rather than minor variations.

These are the core features and improvements of LightGBM which when combined together explain why this advanced methodology consistently delivers the best performance, especially on large-scale, high-dimensional datasets like CICIDS2017.

CHAPTER 4: PERFORMANCE EVALUATION

In this section, the performance of supervised machine learning (ML) algorithms and methodologies is evaluated and thoroughly discussed. The chapter begins with a brief overview of key performance evaluation metrics, providing foundational understanding, and then delves into individual ML algorithms. ML algorithms' performances are analysed across both binary classification and multi-class classification tasks, highlighting strengths, weaknesses, and comparative effectiveness.

Table 4.1 System Specifications

COMPONENT	DETAILS
Brand	Hewlett-Packard (HP)
Model Number	HP 14s-dk0xxx
OS Version	Microsoft Windows 10 Home 10.0.019045
Processor	AMD Ryzen 5 3500U @2.1GHz
Memory	8 GB DDR4
CPU Speed	2.1 GHz Base, upto 3.5 GHz
GPU	AMD Radeon Vega 8 (2 GB)
Core Count	4

4.1 Evaluation Metrics

Accuracy: This metric gives an insight into how the model makes the correct predictions. It considers both correctly predicted network attacks (True Positives) and correctly predicted benign instances (True Negatives) divided by the total number of instances present in the dataset.

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{True Positive} + \text{False Positive} + \text{True Negative} + \text{False Negative}} \quad (4.1)$$

Precision: This metric helps in tracking and identifying the predicted attacks that are actually correct. This displays the accuracy of the model when it predicts something as a network attack. In other words, precision measures the accuracy of model's positive predictions.

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \quad (4.2)$$

F1 Score: This metric combines precision and recall values into a single value based on harmonic mean. This is highly important for an imbalanced dataset which further emphasizes accuracy in predictions. This method is useful when trying to find an optimal balance between false positives and false negatives.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (4.3)$$

Recall: This metric showcases the number of real attacks successfully found by the model. This metric is calculated by dividing the total number of attacks that were correctly identified by the total number of actual attacks. This includes both detected attacks (True Positives) and attacks missed by the model (False Negatives).

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \tag{4.4}$$

Mean Cross Validation: This is the average value of the evaluation score extracted from multiple rounds of cross-validation. CV means that a certain dataset is split into 'k' equal parts (folds). Furthermore, the model is trained on k-1 folds and validated on the remaining part. The process is repeated k times, with each fold used once as a validation set. In this study, *cv=5* is utilized that means the training data is split into 5 equal folds where 4 are the training folds and the remaining one is the validation fold.

$$\text{Mean CV Score} = \frac{1}{k} \sum_{i=1}^{k} \text{Score}_i \tag{4.5}$$

There is no single metric that can evaluate a machine learning algorithm perfectly on its own. As a result, multiple evaluation metrics are always used together to get a clearer picture of underlying performance of the machine learning techniques. In the case of **F1 Score**, it balances precision and recall, making it extremely useful when in situations with imbalanced classes. It also helps in avoiding misleading conclusions and prevents false high accuracy. Similarly, **Precision** helps in shedding some light on the false positives and could be utilized in a system where false alerts are costly and **resource-heavy**.

Recall is highly important from the perspective of cybersecurity and anomaly detection, such as where misclassifying a cyber-threat could be costly and lead to serious consequences for the system. Lastly, the **Mean Cross-Validation score** gives a sense of how stable and generalizable the model is across different folds of training and validation. It's usually treated as a supporting metric, rather than a standalone measure of class-wise performance

4.2 Performance Evaluation for Binary Classification

For detection of network attacks and possible threats, six different machine learning algorithms were used and evaluated against different performance metrics. Additionally, ROC curves describing True Positive Rate against False Positive Rate were plotted for technical evaluation. Correlation matrices were employed to effectively capture the performance and exact values of different machine learning methodologies. The train-test split ratio utilized for this model was 75-25.

4.2.1 Logistic Regression

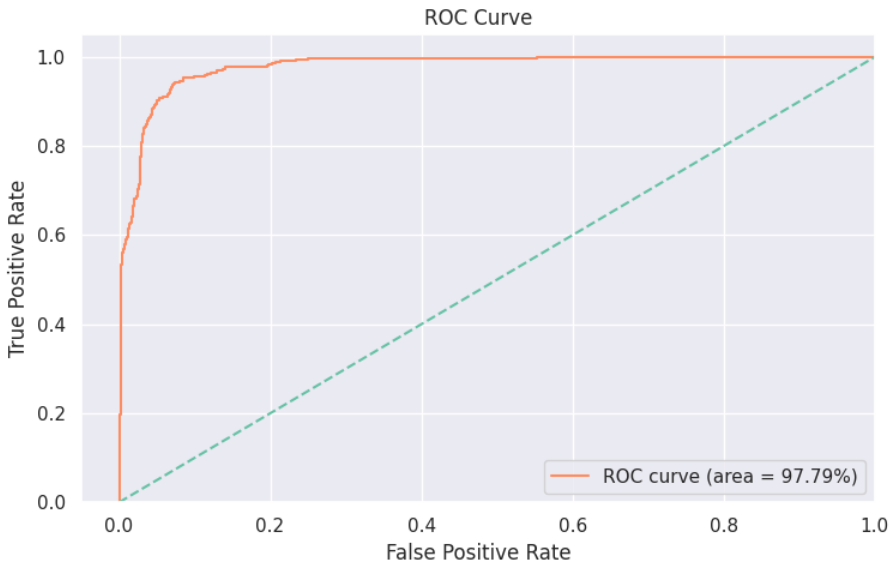

Fig 4.1. ROC Curve for Logistic Regression (Binary Class)

The ROC curve shows a steep rise toward the top-left corner as illustrated in Fig. 4.1, which indicates that the model has a high True Positive Rate (TPR). The Area Under the Curve (AUC) is 97.79%, which is an excellent score and reflects strong classification prowess between the two classes. The ROC curve was generated using Python (Scikit-learn and Seaborn libraries) based on the performance of the Logistic Regression model on the test data.

Precision	Accuracy	F1-Score	Recall	Cross-Validation
93.2%	92.2%	91.7%	92.6%	92.2%

Table 4.2. Performance Metrics for Logistic Regression (Binary Class)

4.2.2 Support Vector Machine

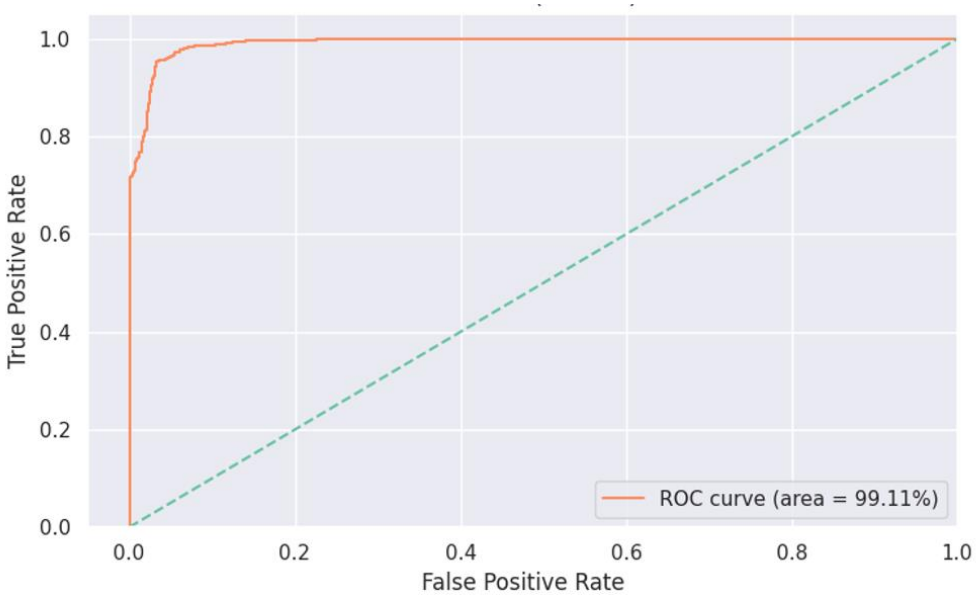

Fig 4.2. ROC Curve for SVM (Binary Class)

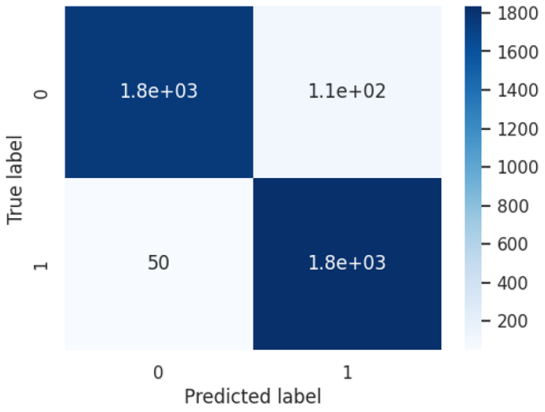

Fig 4.3. Confusion Matrix for SVM (Binary Class)

Precision	Accuracy	F1-Score	Recall	Cross-Validation
95.3%	95.8%	96.1%	95.8%	95.9%

Table 4.3. Performance Metrics for SVM (Binary Class)

The ROC curve analysis confirms strong performance of the SVM classifier. This value demonstrates the model's ability to distinguish between benign and malicious traffic across

various thresholds. As illustrated in Fig. 4.3, The confusion matrix highlights the strong performance of the SVM classifier. Out of all samples, the model correctly identified 1800 benign instances (true negatives) and 1800 malicious instances (true positives), while misclassifying only 110 benign samples. The ROC curve was generated using Python (Scikit-learn and Seaborn libraries) based on the performance of the Support Vector Machine model on the test data.

4.2.3 K-Nearest Neighbor

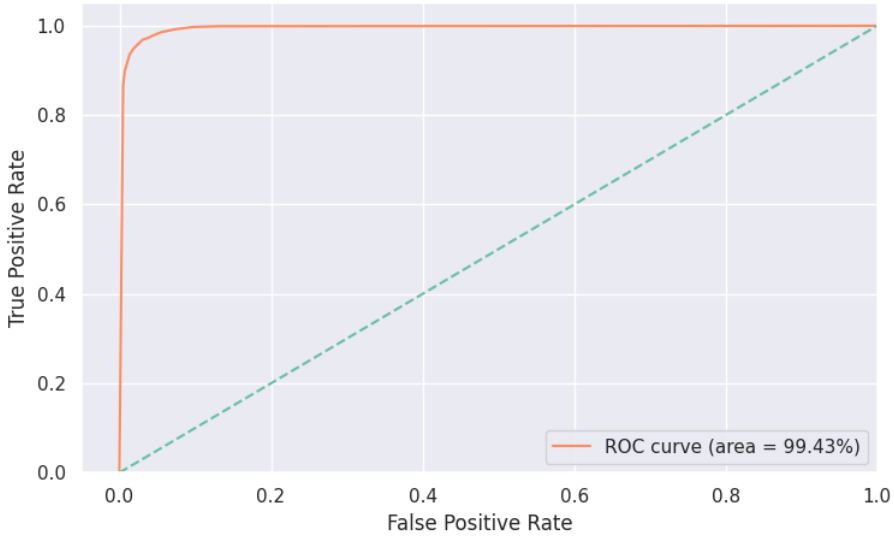

Fig 4.4. ROC Curve for KNN (Binary Class)

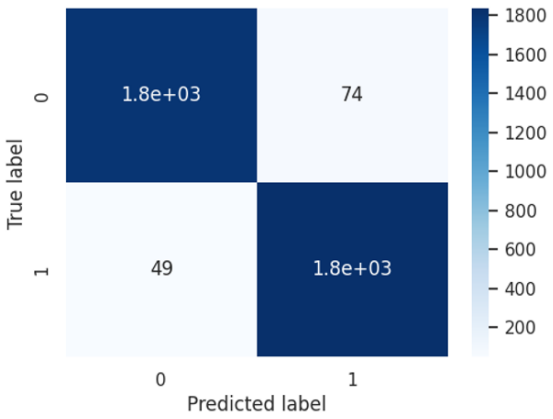

Fig 4.5. Confusion Matrix for KNN (Binary Class)

Precision	Accuracy	F1-Score	Recall	Cross-Validation
95.6%	96.7%	96.0%	95.7%	96.2%

Table 4.4. Performance Metrics for K-NN (Binary Class)

The ROC curve demonstrates strong performance for KNN covering over 99% of the area. The confusion matrix also indicates good results with only a few misclassifications observed during model testing. Although KNN performs well, it does not surpass the benchmark set by LightGBM. Despite achieving over 95% in both accuracy and precision, KNN is computationally intensive and requires significantly more time to train. The ROC curve was generated using Python (Scikit-learn and Seaborn libraries) based on the performance of the KNN model on the test data.

4.2.4 Decision Trees

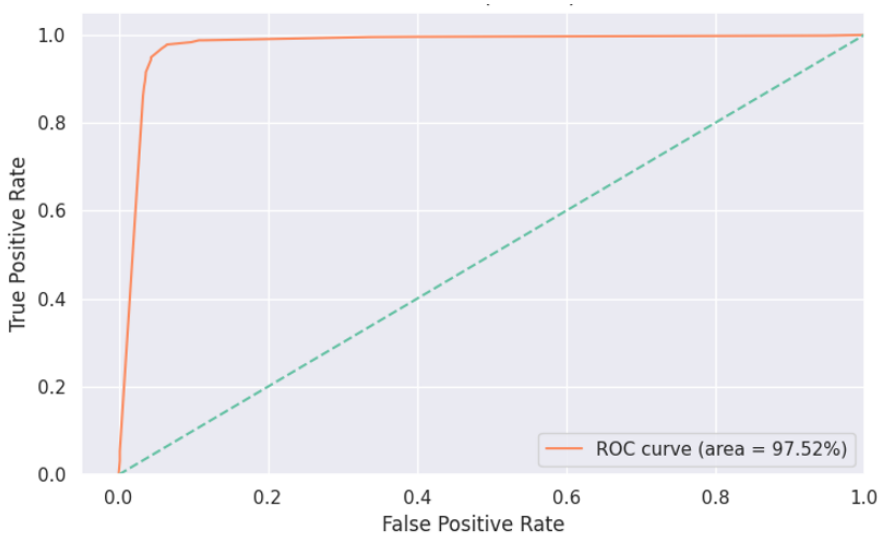

Fig 4.6. ROC Curve for Decision Trees (Binary Class)

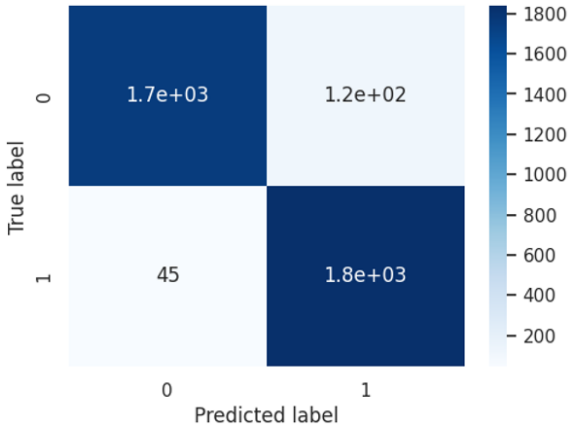

Fig 4.7. Confusion Matrix for Decision Trees (Binary Class)

Precision	Accuracy	F1-Score	Recall	Cross-Validation
95.2%	95.6%	96.1%	95.8%	96.0%

Table 4.5. Performance Metrics for Decision Trees (Binary Class)

As illustrated in Fig. 4.8, the Decision Tree model shows a strong ROC curve, indicating a high True Positive Rate. The ROC curve area is approximately 97.5% suggesting a good distinction between benign and malicious network traffic. However, the confusion matrix reveals that many benign instances are misclassified as malicious which eventually results in a slightly elevated false positive rate. Therefore, Decision Trees may pose challenges in real-world applications due to their tendency to generate an excessive number of false positives. The ROC curve was generated using Python (Scikit-learn and Seaborn libraries) based on the performance of the Decision Tree model on the test data.

4.2.5 LightGBM

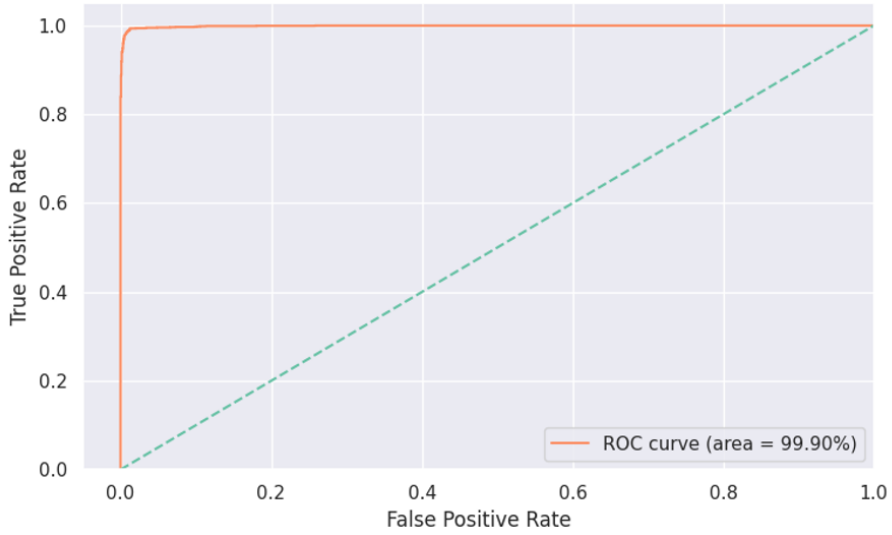

Fig 4.8. ROC Curve for LightGBM (Binary Class)

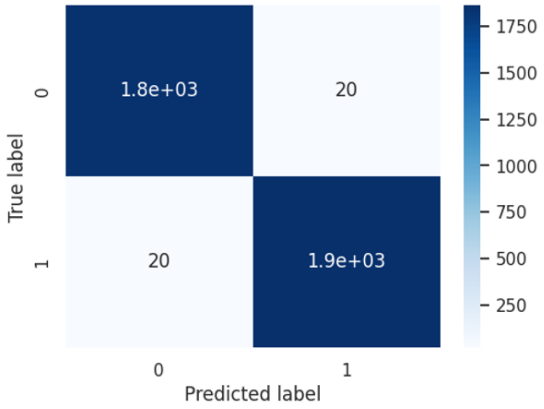

Fig 4.9. Confusion Matrix for LightGBM (Binary Class)

As illustrated in Fig. 4.8, LightGBM has a near-perfect ROC curve, indicating a very high True Positive Rate and a minimal False Positive Rate. The ROC curve area is approximately 99.9%, demonstrating a clear distinction between benign network traffic and malicious traffic flow.

The confusion matrix also highlights excellent classification performance, with only 40 instances misclassified out of the entire dataset. LightGBM exhibits a very high precision score as well as high overall accuracy, indicating outstanding performance over previously discussed

learning methods. The ROC curve was generated using Python (Scikit-learn and Seaborn libraries) based on the performance of the LightGBM model on the test data.

Precision	Accuracy	F1-Score	Recall	Cross-Validation
98.9%	98.8%	98.9%	98.7%	98.9%

Table 4.6. Performance Metrics for LightGBM (Binary Class)

4.2.6 XGBoost

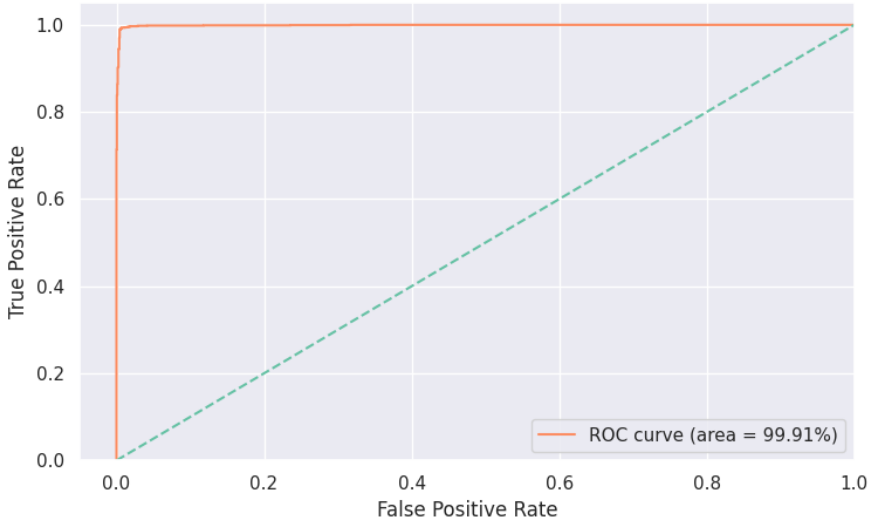

Fig 4.10. ROC Curve for XGBoost (Binary Class)

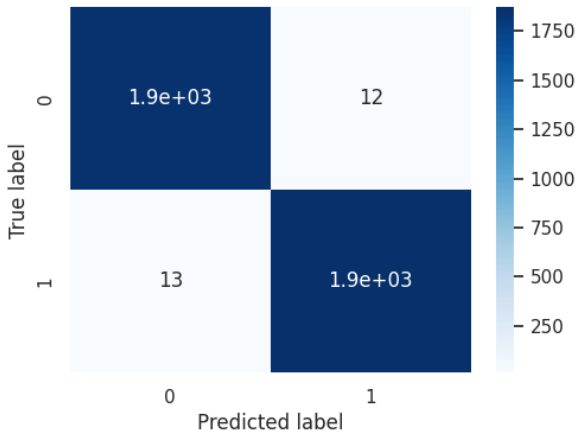

Fig 4.11. Confusion Matrix for XGBoost (Binary Class)

Precision	Accuracy	F1-Score	Recall	Cross-Validation
98.7%	98.7%	98.1%	98.6%	99.1%

Table 4.7. Performance Metrics for XGBoost (Binary Class)

As illustrated in Fig. 4.10, XGBoost demonstrates a near-perfect score and closely mirrors the performance of the previously discussed LightGBM model. This similarity is expected, as LightGBM is built upon the foundational principles of XGBoost which results in almost identical outcomes across all evaluation metrics.

A key distinction lies in processing time where LightGBM achieved approximately ***37% faster performance*** compared to the more computationally intensive XGBoost. Additionally, the confusion matrix highlights XGBoost's strong classification capabilities, with fewer than 40 instances misclassified across the entire dataset—a slight improvement over the LightGBM approach.

4.3 Tabular Comparison: Performance Evaluation for Binary Classification

Classifier	Precision	Accuracy	F1-Score	Recall
Logistic Regression	93.2%	92.2%	91.7%	92.6%
SVM	95.3%	95.8%	96.1%	95.8%
KNN	95.6%	96.7%	96.0%	95.7%
Decision Trees	95.2%	95.6%	96.1%	95.8%
LightGBM	98.9%	98.8%	98.9%	98.7%
XGBoost	98.7%	98.7%	98.1%	98.6%

Table 4.8. Performance Comparison for Binary Classifiers

The performance evaluation of classifiers for multi-class classification demonstrates that LightGBM consistently outperforms other models across key metrics such as precision, accuracy, F1-score, and recall. While both LightGBM and XGBoost show excellent results, LightGBM stands out due to its significantly faster processing time which makes it a more efficient choice for large-scale applications. Compared to XGBoost, LightGBM offers similar or better predictive performance but with substantially lower computational overhead, establishing its position as the most optimized and scalable solution among all the methodologies evaluated.

4.4 Performance Evaluation as a Function of Reduced Features for Binary Classification

This section presents the performance of various machine learning models for binary classification while utilizing a reduced set of features. After applying Principal Component Analysis (PCA) to transform the original dataset, feature importance scores were calculated to identify the most relevant components. These selected features were then used to retrain models like SVM, K-Nearest Neighbors (KNN), Decision Trees, XGBoost and LightGBM. The primary objective of this method is to check if similar or better performance can be achieved using fewer features. This approach helps reduce processing time and memory usage.

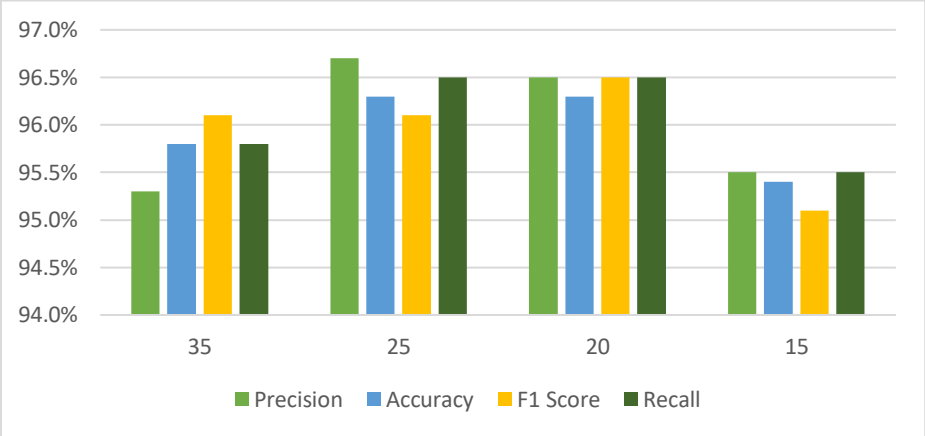

Fig 4.12. SVM: Performance Evaluation with Reduced Features

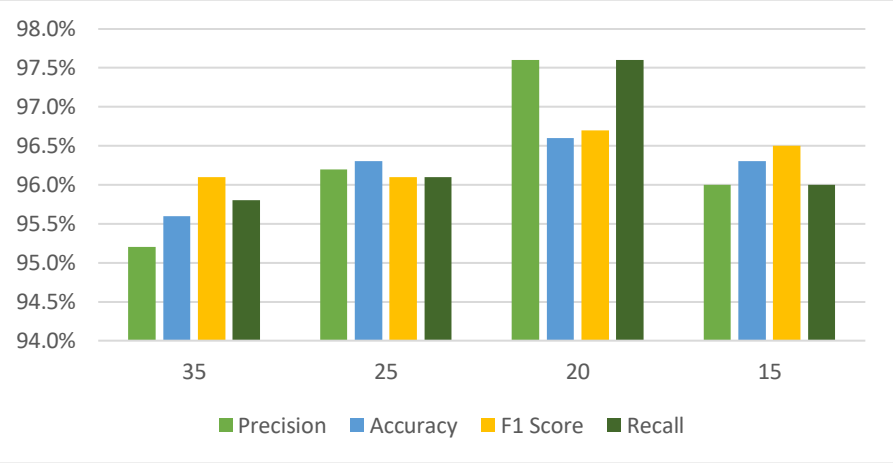

Fig 4.13. Decision Trees: Performance Evaluation with Reduced Features

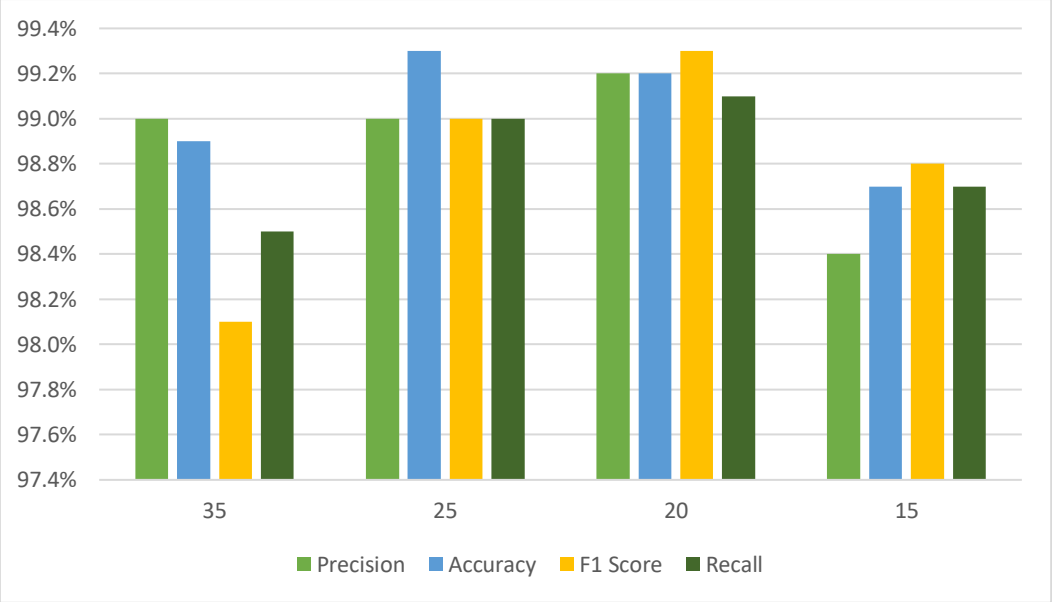

Fig 4.14. LightGBM: Performance Evaluation with Reduced Features

The bar charts illustrated in the figures show that using only 202-25 features gives the best results, with LightGBM achieving over 99% evaluation scores. This suggests that feature reduction, even after PCA enhancement, can improve or stabilise binary network traffic classification.

4.5 Performance Evaluation for Multi-Class Classification

For multi-class detection of network attacks, a variety of machine learning algorithms were used and evaluated against different performance metrics. Additionally, ROC curves describing True Positive Rate against False Positive Rate were plotted for technical evaluation based on different network attack classes. Correlation matrices were also utilized to effectively capture the performance and the associated numerical statistics.

4.5.1 Random Forest

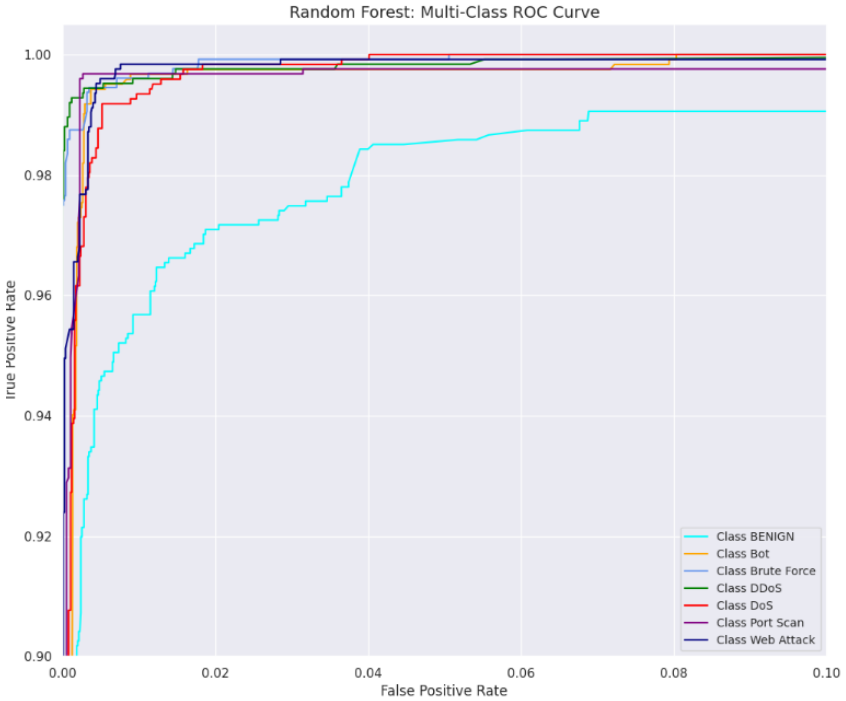

Fig 4.15. ROC Curve for Random Forest (Multi-Class)

As illustrated in Fig. 4.14, all the ROC curves pertaining to different threat-type classes merge into the top-left corner. This indicates high true positive rates (TPR) and low false positive rates (FPR). Also, it suggests excellent separability between classes. The ROC curve for 'BENIGN' is slightly below the others indicating that the model has a slightly higher false positive rate.

Precision	Accuracy	F1-Score	Recall	Cross-Validation
99.0%	98.9%	98.1%	98.5%	96.7%

Table 4.9. Performance Metrics for Random Forest (Multi-Class)

4.5.2 K-Nearest Neighbors

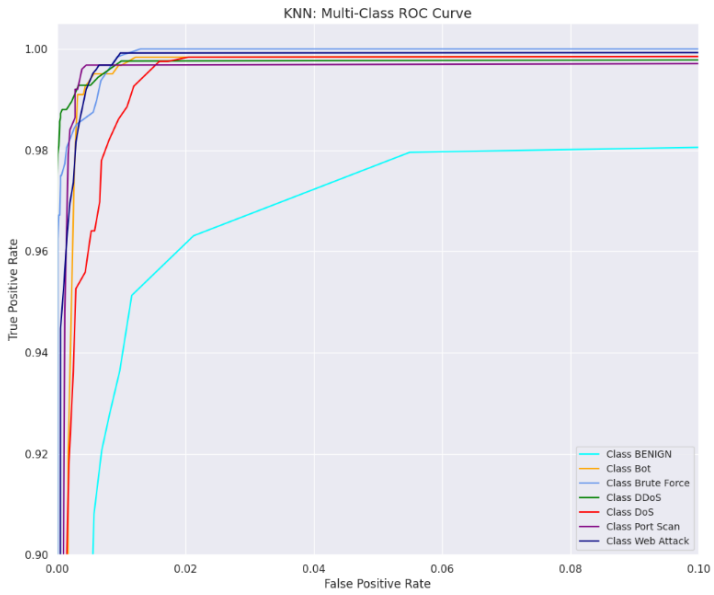

Fig 4.16. ROC Curve for KNN (Multi-Class)

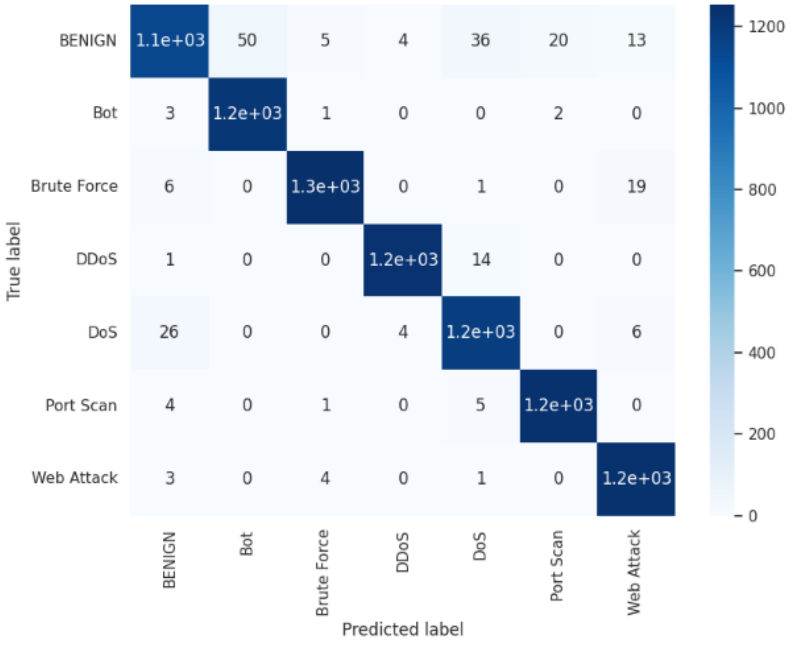

Fig 4.17. Confusion Matrix for KNN (Multi-Class)

The KNN classifier performs exceptionally well for most classes, as indicated by the steep rise in TPR at very low FPR values. Classes like Port Scan, Web Attack, Brute Force, BENIGN, etc., have near-perfect ROC curves. However, Class "Bot" shows relatively lower TPR for the same FPR as illustrated in Fig. 4.15. High accuracy is observed across most classes when referring to

Fig. 4.16, especially Brute Force, DDoS, DoS, Port Scan, and Web Attack. On the other hand, Benign traffic was mostly classified correctly, but misclassified as Bot, Port Scan, or Web Attack, suggesting some overlap in feature space.

Precision	Accuracy	F1-Score	Recall	Cross-Validation
97.1%	97.4%	97.3%	97.4%	96.8%

Table 4.10. Performance Metrics for K-Nearest Neighbor (Multi-Class)

4.5.3 Decision Trees

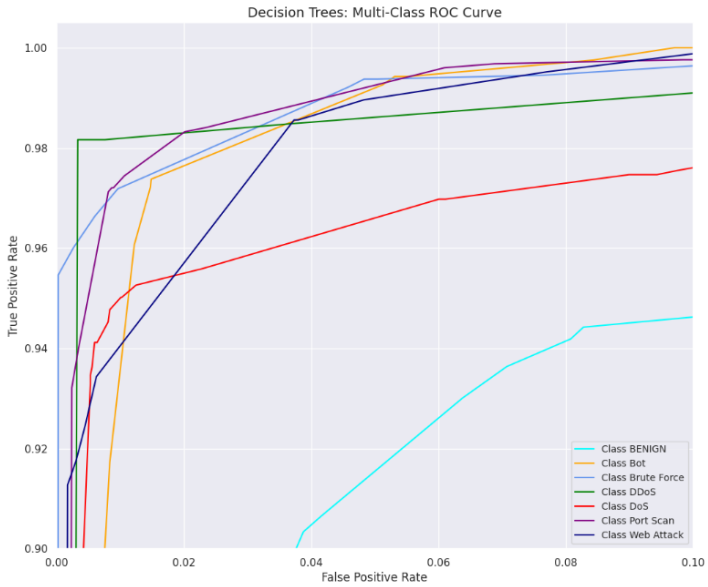

Fig 4.18. ROC Curve for Decision Trees (Multi-Class)

As showcased in Fig. 4.17, Decision Tree classifier performs competently across most attack types, particularly Web Attack, Brute Force, DoS, and Port Scan. The ROC curve for DDoS (red line) is consistently lower compared to other classes, indicating that Decision Tree has difficulty distinguishing DDoS from other traffic. Similarly, the Bot class (light blue) shows a more gradual curve which displays weaker recall

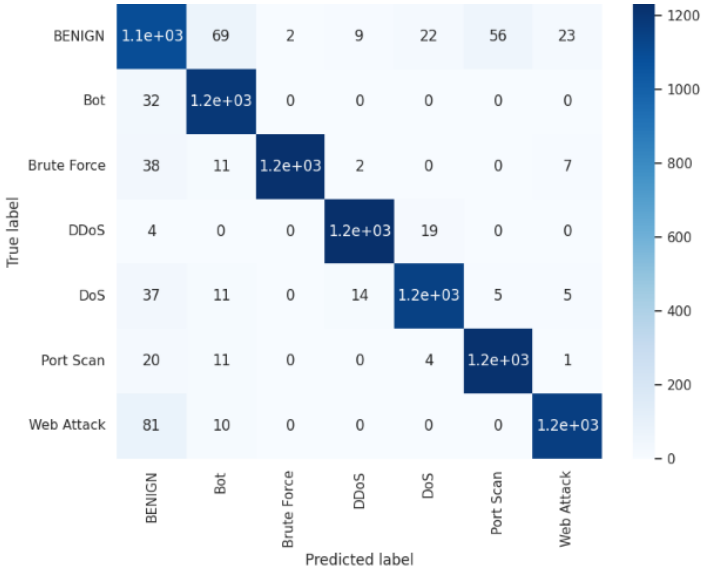

Fig 4.19. Confusion Matrix for Decision Trees (Multi-Class)

Precision	Accuracy	F1-Score	Recall	Cross-Validation
93.3%	94.4%	94.1%	93.7%	94.8%

Table 4.11. Performance Metrics for Decision Trees (Multi-Class)

4.5.4 LightGBM

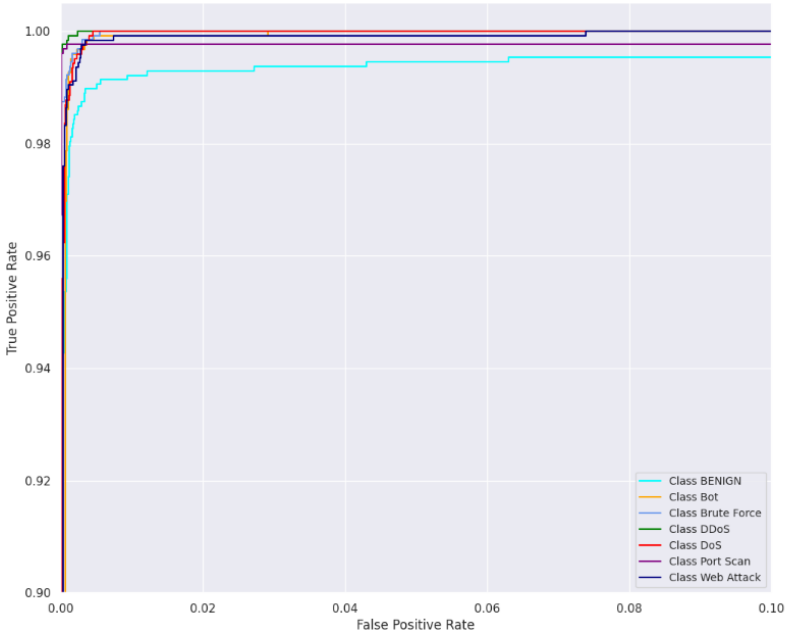

Fig 4.20. ROC Curve for LightGBM (Multi-Class)

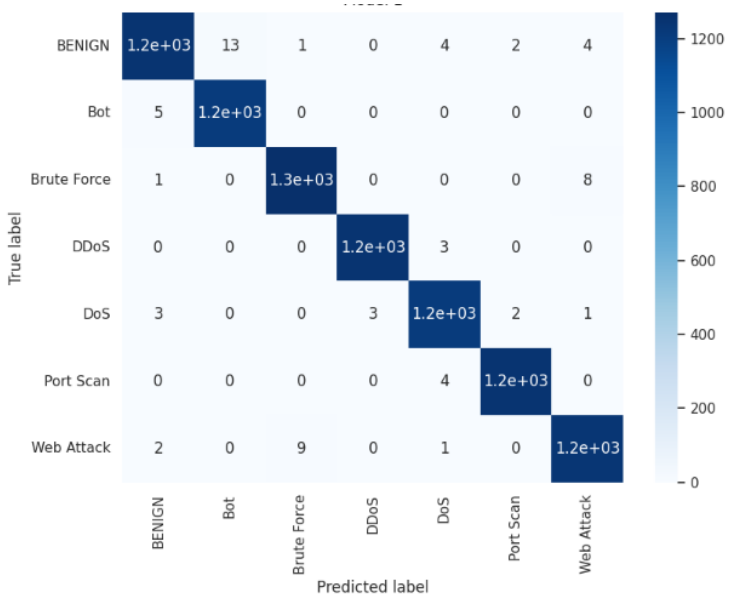

Fig 4.21. Confusion Matrix for LightGBM (Multi-Class)

In Fig. 4.19, all classes have ROC curves very close to the top-left corner, indicating high sensitivity and low false alarm rate. This is true especially for classes like DDoS, DoS, Brute Force, and Web Attack, the curves nearly touch which means near-perfect classification. As a result, ROC curves display excellent discriminative ability across all labels.

The confusion matrix displays that the LightGBM model performs exceptionally well in multi-class classification, with very few misclassifications across all classes. The 'BENIGN' class was occasionally mislabelled with Bot (13 instances), Port Scan (4 instances), and Web Attack (4 instances) indicating a small number of false positives. In contrast, the 'BOT' class had minimal mislabelling with only 5 instances misclassified as 'BENIGN'.

Precision	Accuracy	F1-Score	Recall	Cross-Validation
98.8%	98.8%	98.7%	98.8%	99.1%

Table 4.12. Performance Metrics for LightGBM (Multi-Class)

4.5.5 XGBoost

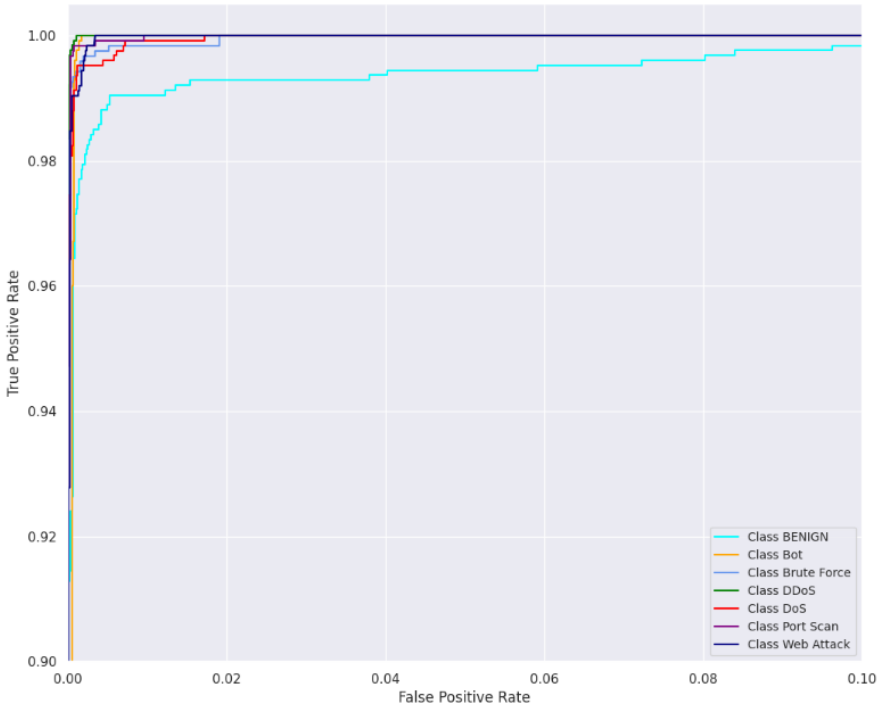

Fig 4.22. ROC Curve for XGBoost (Multi-Class)

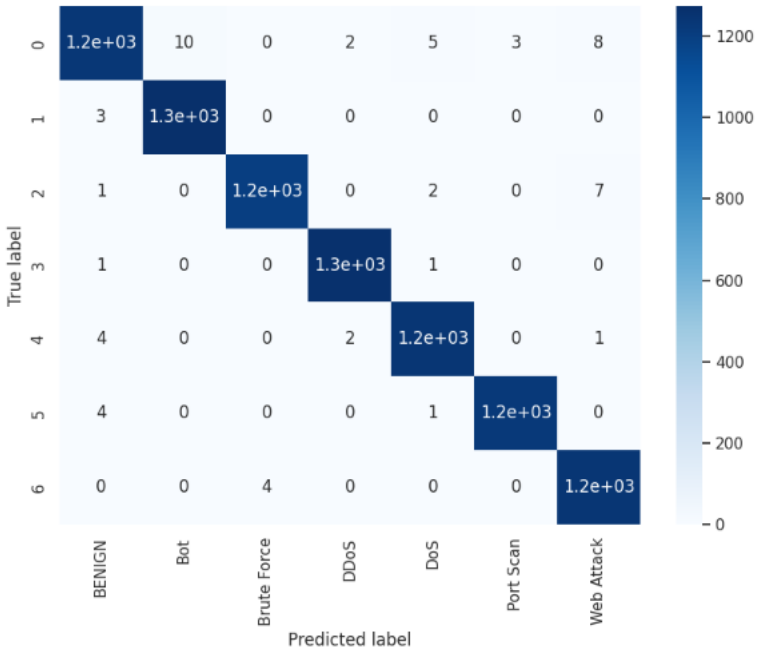

Fig 4.23. Confusion Matrix for XGBoost (Multi-Class)

The figure 4.21(a) describes that XGBoost demonstrates a near-perfect score for the ROC curve for the multi-class classification task. The performance of this methodology is extremely identical and matches the LightGBM's exceptional performance. The primary downside is the processing time, which when compared to LightGBM is approximately ***9% slower.*** On the positive side, the confusion matrix highlights strong classification capabilities with fewer than 100 errors across the entire dataset.

Precision	Accuracy	F1-Score	Recall	Cross-Validation
98.7%	98.7%	98.5%	98.3%	99.2%

Table 4.13. Performance Metrics for XGBoost (Multi-Class)

4.6 Tabular Comparison: Performance Evaluation for Multi-Class Classification

Classifier	Precision	Accuracy	F1-Score	Recall
Random Forest	99.0%	98.9%	98.1%	98.5%
KNN	97.1%	97.4%	97.3%	97.4%
Decision Trees	93.3%	94.4%	94.1%	93.7%
LightGBM	98.8%	98.8%	98.7%	98.8%
XGBoost	98.7%	98.7%	98.5%	98.3%

Table 4.14. Performance Comparison for Multi-Class Classifiers

The tabular comparison in this section highlights the top classifiers based on multi-class performance metrics. Random Forest achieved the highest precision and accuracy, but it suffers from significant computational inefficiency, being nearly 100% slower than LightGBM. XGBoost, with similarly high metrics, is also about 10% slower than LightGBM. On the other hand, LightGBM delivers consistently high performance across all metrics while maintaining superior computational efficiency which makes it most balanced and scalable choice among all classifiers evaluated.

4.7 Performance Evaluation as a Function of Reduced Features for Multi-Class Classification

This section focuses on evaluating the performance of machine learning models for multi-class classification using a reduced number of features. Based on the PCA-transformed dataset, features with the highest importance scores were selected and used to retrain models such as Random Forest, KNN, Decision Trees, and LightGBM. As a result, these retrained models were able to accurately detect different types of attacks while using less data.

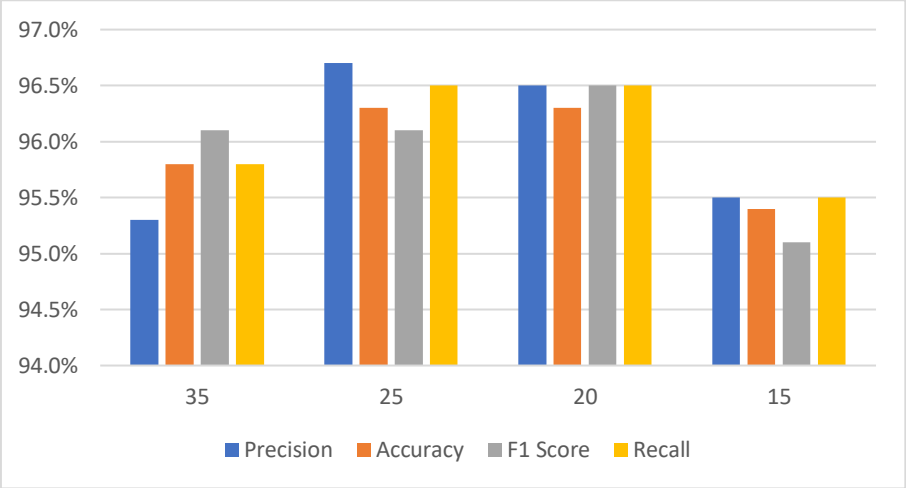

Fig 4.24. Random Forest: Performance Evaluation with Reduced Features (Multi-Class)

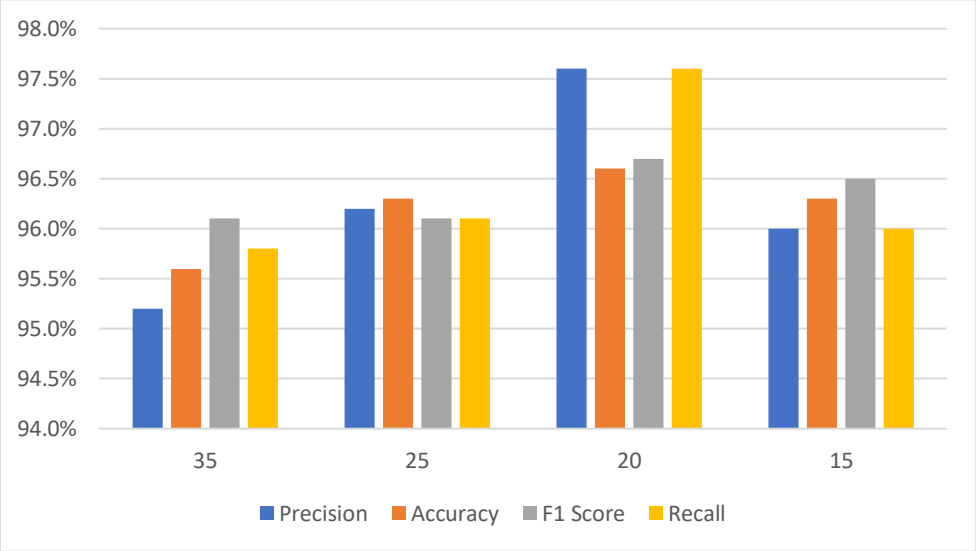

Fig 4.25. Decision Trees: Performance Evaluation with Reduced Features (Multi-Class)

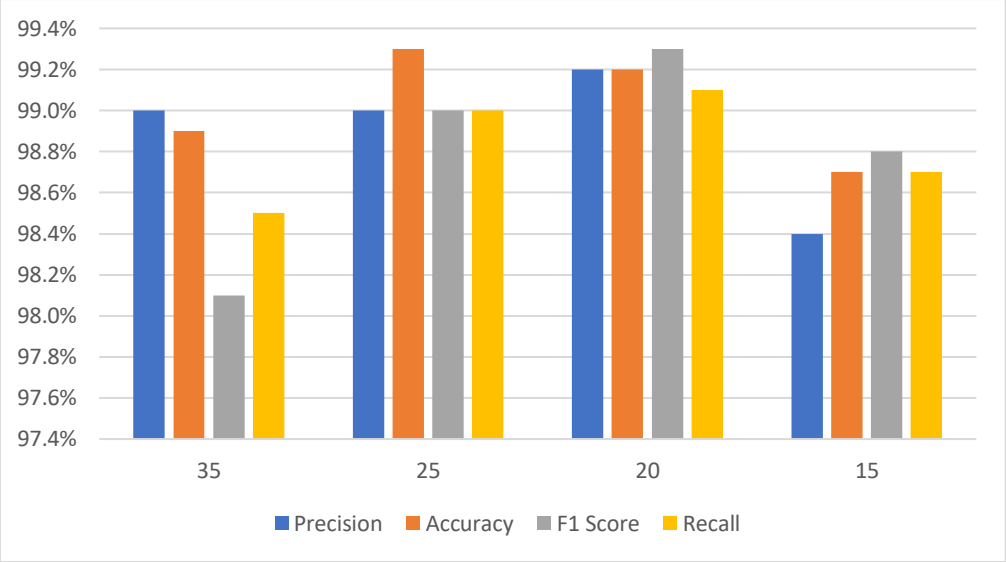

Fig 4.26. LightGBM: Performance Evaluation with Reduced Features (Multi-Class)

For multi-class labelling and classification, using 20-25 features also yields best results, with LightGBM reaching nearly 99% performance evaluation scores. This is confirmed by the bar charts illustrated in the figures which portrays that feature reduction enhances or maintains performance, even in more complex classification tasks.

CHAPTER 5: CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

In this study, machine learning techniques were effectively utilized to develop a robust system for both binary and multi-class classification of network intrusions using the benchmarked (CICIDS2017) dataset. The main objective was to build a binary classification model capable of distinguishing between benign and malicious traffic. The results obtained through this work were highly promising, with all evaluation metrics such as accuracy, precision, recall, and F1-score, ***exceeding 95%*** which reflected the proposed model's strong ability to detect threats in network simulations.

To further improve the utility of the proposed system, a multi-class classification functionality was incorporated, enabling the identification of specific attack types. This functionality not only detects the presence of threats but also provides insights into their nature, allowing security teams to deploy and utilize appropriate countermeasures. Among the algorithms evaluated ***LightGBM*** consistently emerged as the ***top performer*** for both binary and multi-class classification tasks, delivering superior accuracy and generalization performance. Although LightGBM emerged as the top-performing model with the highest evaluation metrics, XGBoost was a close second, demonstrating comparable performance across both binary and multi-class classification tasks. Despite its competitive accuracy, XGBoost fell short primarily due to its slower processing speed.

The performance of the proposed model was further improved through dimensionality reduction and feature selection techniques. The original high-dimensional dataset was refined using Principal Component Analysis (PCA), which reduced redundancy by combining correlated features into a smaller set of uncorrelated components. Feature Importance scores were mainly leveraged to identify and retain the most relevant features. Additionally, **the findings highlighted** that using **only 20–25 features** from the PCA-enhanced dataset led to the highest classification accuracy which also led to significantly reduced computational costs and minimal processing overhead.

This improvement demonstrated the importance of intelligent feature engineering in building scalable anomaly detection systems for cybersecurity and its associated applications.

In a live deployment situations, certain features may be missing due to packet loss, logging failures, or resource constraints. ***Autoencoders*** offer a robust solution in such cases by learning compressed latent representations of the complete input feature space during training. Once training phase is carried out, the encoder portion of the autoencoder can project incoming incomplete data into the latent space by leveraging the ***learned feature correlations***. Since autoencoders are trained to minimize reconstruction loss across all features, they capture inter-feature dependencies which allows them to compensate for missing values during the encoding process. As a result, even when some input features are unavailable, the autoencoder can still produce meaningful ***latent vectors*** that preserve the underlying structure of the data. These vectors can then be passed to the machine learning classifier (such as LightGBM) further ensuring reliable anomaly detection despite partial feature availability in real-time environments.

In this study, PCA was applied initially to reduce the dimensionality of the dataset, improving computational efficiency and minimizing noise. On a similar note, autoencoders can be employed to further encode the PCA-transformed features into a compact latent representation that captures nonlinear dependencies and inter-feature relationships. This ***two-stage reduction*** approach serves multiple purposes where it streamlines the input for classification and also enhances resilience against incomplete data in real-time operations.

5.2 Future Directions

Future development of the proposed anomaly detection system could be focused on deploying effective ML-based anomaly detection model coupled with live network environments to monitor traffic in real-time. A robust data ingestion pipeline could be proposed which handles continuous packet capture, feature extraction and real-time classification. Furthermore, as the network behavior evolves, ML retraining would be necessary. This could be achieved periodically to maintain high detection accuracy.

Additionally, an automated pipeline could be developed such that labelled data is collected, model drift is evaluated and retraining phase is triggered when performance degradation is displayed. Incremental learning process could be utilized to further enhance adaptability without requiring complete retraining from ground zero. To ensure seamless integration with existing security solutions, the anomaly detection system (ADS) can be connected with infrastructures such as firewalls and SIEM tools, creating enhanced defense mechanisms for threat mitigation.

REFERENCES

1. Admass, W.S., Munaye, Y.Y. and Diro, A.A., 2024. Cyber security: State of the art, challenges and future directions. *Cyber Security and Applications*, *2*, p.100031.
2. Liu, R., Shi, J., Chen, X. and Lu, C., 2024. Network anomaly detection and security defense technology based on machine learning: A review. *Computers and Electrical Engineering*, *119*, p.109581.
3. Yulianto, A., Sukarno, P. and Suwastika, N.A., 2019, March. Improving adaboost-based intrusion detection system (IDS) performance on CIC IDS 2017 dataset. In *Journal of Physics: Conference Series* (Vol. 1192, p. 012018). IOP Publishing.
4. Maseer, Z.K., Yusof, R., Bahaman, N., Mostafa, S.A. and Foozy, C.F.M., 2021. Benchmarking of machine learning for anomaly based intrusion detection systems in the CICIDS2017 dataset. *IEEE access*, *9*, pp.22351-22370.
5. Ustebay, S., Turgut, Z. and Aydin, M.A., 2018, December. Intrusion detection system with recursive feature elimination by using random forest and deep learning classifier. In *2018 international congress on big data, deep learning and fighting cyber terrorism (IBIGDELFT)* (pp. 71-76). IEEE.
6. Hnamte, V., Nhung-Nguyen, H., Hussain, J. and Hwa-Kim, Y., 2023. A novel two-stage deep learning model for network intrusion detection: LSTM-AE. *IEEE Access*, *11*, pp.37131-37148.
7. Elmrabit, N., Zhou, F., Li, F. and Zhou, H., 2020, June. Evaluation of machine learning algorithms for anomaly detection. In *2020 international conference on cyber security and protection of digital services (cyber security)* (pp. 1-8). IEEE.
8. Kussul, E. and Baidyk, T., 2004. Improved method of handwritten digit recognition tested on MNIST database. *Image and Vision Computing*, *22*(12), pp.971-981.
9. Wang, S., Balarezo, J.F., Kandeepan, S., Al-Hourani, A., Chavez, K.G. and Rubinstein, B., 2021. Machine learning in network anomaly detection: A survey. *IEEE Access, 9*, pp.152379-152396.
10. NumPy Developers, *NumPy Documentation*, version 1.26, [online] Available at: https://numpy.org/doc/
11. Pandas Development Team, *Pandas Documentation*, version 2.2, [online] Available at: https://pandas.pydata.org/docs/

12. Scikit-learn Developers, *sklearn.ensemble.RandomForest — Scikit-learn documentation*, version 1.4, [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
13. Canadian Institute for Cybersecurity, *CIC-IDS 2017 Dataset*, [online] Available at: http://cicresearch.ca/CICDataset/CIC-IDS-2017/Dataset/CIC-IDS-2017/
14. Sharafaldin, I., Lashkari, A.H. and Ghorbani, A.A., 2018. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, *1*(2018), pp.108-116.
15. Soltani, M., Siavoshani, M.J. and Jahangir, A.H., 2022. A content-based deep intrusion detection system. *International Journal of Information Security*, *21*(3), pp.547-562.
16. Fortinet, "DoS vs DDoS Attacks," [online] Available at: https://www.fortinet.com/resources/cyberglossary/dos-vs-ddos
17. Adhikari, D., Jiang, W., Zhan, J., He, Z., Rawat, D.B., Aickelin, U. and Khorshidi, H.A., 2022. A comprehensive survey on imputation of missing data in internet of things. *ACM Computing Surveys*, *55*(7), pp.1-38.
18. Hall, M.A., 1999. *Correlation-based feature selection for machine learning* (Doctoral dissertation, The University of Waikato).
19. Reddy, M.B. and Reddy, L.S.S., 2010. Dimensionality reduction: An empirical study on the usability of IFE-CF (independent feature elimination-by C-correlation and F-correlation) measures. *arXiv preprint arXiv:1002.1156*.
20. Chowdhury, R., Roy, A., Saha, B. and Bandyopadhyay, S.K., 2022. An intelligent intrusion detection system using a novel combination of PCA and MLP. In *Engineering Mathematics and Computing* (pp. 93-104). Singapore: Springer Nature Singapore.
21. Wu, T., Fan, H., Zhu, H., You, C., Zhou, H. and Huang, X., 2022. Intrusion detection system combined enhanced random forest with SMOTE algorithm. *EURASIP Journal on Advances in Signal Processing*, *2022*(1), p.39.
22. Sayegh, H.R., Dong, W. and Al-madani, A.M., 2024. Enhanced intrusion detection with LSTM-Based model, feature selection, and SMOTE for imbalanced data. *Applied Sciences*, *14*(2), p.479.
23. Uzun, B. and Ballı, S., 2022. A novel method for intrusion detection in computer networks by identifying multivariate outliers and ReliefF feature selection. *Neural Computing and Applications*, *34*(20), pp.17647-17662.S

24. Al-Hadhrami, Y. and Hussain, F.K., 2021. DDoS attacks in IoT networks: a comprehensive systematic literature review. *World Wide Web*, *24*(3), pp.971-1001.
25. Kleinbaum, D.G., Dietz, K., Gail, M., Klein, M. and Klein, M., 2002. *Logistic regression* (p. 536). New York: Springer-Verlag.
26. Zou, X., Hu, Y., Tian, Z. and Shen, K., 2019, October. Logistic regression model optimization and case analysis. In *2019 IEEE 7th international conference on computer science and network technology (ICCSNT)* (pp. 135-139). IEEE.
27. ejable.com, "Random Forest in Machine Learning," [online] Available at: https://www.ejable.com/tech-corner/ai-machine-learning-and-deep learning/random-forest-in-machine-learning/
28. Charbuty, B. and Abdulazeez, A., 2021. Classification based on decision tree algorithm for machine learning. *Journal of applied science and technology trends*, *2*(01), pp.20-28.
29. Kang, S., 2021. K-nearest neighbor learning with graph neural networks. *Mathematics*, *9*(8), p.830.
30. Jiménez-Cordero, A., Morales, J.M. and Pineda, S., 2021. A novel embedded min-max approach for feature selection in nonlinear support vector machine classification. *European Journal of Operational Research*, *293*(1), pp.24-35.
31. Ramraj, S., Uzir, N., Sunil, R. and Banerjee, S., 2016. Experimenting XGBoost algorithm for prediction and classification of different datasets. *International Journal of Control Theory and Applications*, *9*(40), pp.651-662.
32. Dhaliwal, S.S., Nahid, A.A. and Abbas, R., 2018. Effective intrusion detection system using XGBoost. *Information*, *9*(7), p.149.
33. Neptune.ai Team, "XGBoost vs LightGBM: Which Algorithm Should You Use?", [online] Available at: https://neptune.ai/blog/xgboost-vs-lightgbm
34. Aslan, Ö., Aktuğ, S.S., Ozkan-Okay, M., Yilmaz, A.A. and Akin, E., 2023. A comprehensive review of cyber security vulnerabilities, threats, attacks, and solutions. *Electronics*, *12*(6), p.1333.
35. Canadian Institute for Cybersecurity (CIC), 2017. *Intrusion Detection Evaluation Dataset (CICIDS2017)*. University of New Brunswick. Available at: https://www.unb.ca/cic/datasets/ids-2017.html.

36. Abdi, H. and Williams, L.J., 2010. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, *2*(4), pp.433-459.
37. Blagus, R. and Lusa, L., 2013. SMOTE for high-dimensional class-imbalanced data. *BMC bioinformatics*, *14*(1), p.106.