ISSN: 2995-3286

Research Article

International Journal of Media and Networks

# A Robust and Scalable Machine Learning Based Network Intrusion Detection System for Real-Time Threat Detection in High-Volume Networks

**Yazeeth Najeeb***

*MSc in Information Technology, Srilanka*

***Corresponding Author**
Yazeeth Najeeb, MSc in Information Technology, Srilanka.

**Submitted:** 2025, May 16; **Accepted:** 2025, Jul 11; **Published:** 2025, Jul 18

**Citation:** Najeeb, Y. (2025). A Robust and Scalable Machine Learning Based Network Intrusion Detection System for Real-Time Threat Detection in High-Volume Networks. *Int J Med Net, 3*(4), 01-61.

**Abstract**

*This paper presents the design and implementation of a hybrid Network Intrusion Detection System (NIDS) that leverages both machine learning (ML) and deep learning (DL) techniques to achieve high-precision, real-time threat detection and classification. By combining anomaly-based and signature-based approaches, the system continuously monitors network features—such as IP addresses, packet patterns, and throughput—and distinguishes normal traffic from malicious activity with minimal false alarms. During development, four ML algorithms (Random Forest, support vector machine, and XGBoost) and two DL architectures (convolutional neural network, artificial neural network) were evaluated; XGBoost and an artificial neural network (ANN) achieved the highest standalone accuracies. These two models were then integrated into a unified hybrid framework, yielding an overall detection accuracy of 96.0%. The NIDS is delivered as an intuitive web application built with Flask, enabling security analysts to visualize alerts, inspect detailed threat classifications, and respond swiftly. Experimental results demonstrate that this hybrid approach not only accelerates detection but also enhances classification granularity, making it a practical tool for strengthening modern network defenses.*

**Keywords:** Anomaly-Based Detection, Artificial Neural Network, Convolutional Neural Network, Deep Learning, Xgboost

## 1. Introduction

### 1.1 Understanding Network Attacks and the Need for Advanced Detection Systems

With the highly interwoven nature of the current world, network security has become a bedrock of digital infrastructure. As technology and the internet continued to expand in a quick pace, businesses, governments, and individuals rely increasingly on computer networks for communication, data storage, and running their operations. But this dependence also sees networks as hacker targets. The number and complexity of network attacks have increased (challenging cybersecurity) [1]. As a result, these attacks can compromise sensitive data, disrupt operations, and cause damage to your bottom line as well as reputation. The importance of safeguarding the network systems from malicious activities is therefore on top of the list.

Any unauthorized action on a digital network aimed at accessing, altering, disabling, destroying, or stealing data is referred to as a network attack. There are two classes of these attacks as active and passive. Active attacks consist of direct interference in the functioning of the network that can be modifying data or disrupting services while passive attacks signify eavesdropping or monitoring the network traffic without modification. They are both forms of serious threats to confidentiality, integrity, and availability of information [2].

### 1.2 Types of Network Attacks

Network attacks manifest in various forms, each with distinct techniques and impacts.

Understanding these types is essential for building effective defence mechanisms.

#### 1.2.1 Denial of Service (DoS) and Distributed Denial of Service (DDoS) Attacks

DoS and DDoS attacks overly consume network resources making the services unavailable for legitimate users. In DoS attacks,

the target is flooded, however, with tons of traffic from a single source. On the other hand, DDoS attacks occur when sources that have been compromised flood the victim. These attacks can render websites, servers and even entire networks downs and incur monetary loss [3].

### 1.2.2 Man-in-the-Middle (MitM) Attacks

In MitM attacks, an attacker intercepts and alters communication between two parties without their knowledge. This enables data theft, such as login credentials or financial information, and can compromise the authenticity and integrity of communications [4]. *Phishing and Social Engineering* Phishing attacks exploit human psychology rather than technical vulnerabilities. Attackers use deceptive emails, messages, or websites to trick users into revealing sensitive information, passwords or credit card numbers. Social engineering tactics often accompany other technical exploits, making them particularly dangerous [5].

### 1.2.3 Malware and Ransomware

Malware refers to malicious software designed to damage, disrupt, or gain unauthorized access to systems. Ransomware, a type of malware, encrypts victim data and demands payment for its release. These attacks have become increasingly prevalent and costly for businesses and individuals such as [6].

### 1.2.4 SQL Injection

SQL injection attacks target databases by inserting malicious SQL queries through input fields on a website or application. Successful attacks can reveal, modify, or delete critical data, compromising the security and functionality of web-based systems [7].

### 1.2.5 Zero-Day Exploits

Unlike normal attacks, zero-day attacks make use of unknown vulnerabilities in the software or hardware before the developers could release the patches. Unfortunately, due to the fact that traditional security measures will often not detect these attacks, allowing attackers to gain unauthorized access and command [8].

### 1.2.6 Advanced Persistent Threats (APTs)

APTs are prolonged, targeted attacks where an intruder gains persistent access to a network to exfiltrate sensitive data. These sophisticated attacks often bypass standard security measures, using a combination of stealthy techniques and multiple entry points to maintain long-term access [9].

### 1.2.7 The Impact of Network Attacks

Network attack ramifications are wide and varied. Downtime, data breaches, or the ransom payment cause businesses to lose finances. Such behaviour often leads to reputational damage and loss of customers' and market position's trust. If sensitive information is failed to be protected, then legal and regulatory repercussions may follow especially for industries under strict data protection laws [10]. They can also be used on a larger scale, threatening the safety and security of the public, and of the nation.

For the people it is reducing the possibility of stealing identity, and the monetary loss, and loss of their privacy. The need for protecting network integrity becomes increasingly important because network is getting more and more integrated in our daily life and in our daily interaction in the online world [11].

### 1.2.8 The Need for Advanced Network Intrusion Detection Systems (NIDS)

Typically, traditional security solutions like firewalls and antivirus software are inadequate against the ever-changing cyber threat landscape. Firewalls act as an intermediary and filter incoming and outgoing traffic according to the predefined rules, whereas antivirus programs try to identify known malware signatures. Yet all these methods fall short to sophisticated attacks such as zero-day exploits, APTs, and the latest malware [12].

To address these limitations, Network Intrusion Detection Systems (NIDS) do the monitoring of network traffic in real time to detect particular suspicious behaviour.

## 2. Background Study

With the rapid changing world of cyber security, Network Intrusion Detection Systems (NIDS) is one of the never-ending security tools to provide protection to digital infrastructures against malicious activities. These systems have been enhanced by integration of Machine Learning (ML) and Deep Learning (DL) technique in detection accuracy and scalability by developing hybrid models . In this background study, we provide a comprehensive overview of IDSs background, an application of ML and DL in intrusion detection, related work in IDSs, the problem of IDSs, challenges faced, and future directions of research.

### 2.1 There are Two Primary Types of NIDS

### 2.1.1 Signature-Based Detection

This method of attack detection involves comparing the network activity with a database of threat signatures that are known. Although it is effective against standard threats, it is ineffective against new and changing attack methods [13].

- **Anomaly-Based Detection**

In anomaly detection, statistical models, machine learning and behavioural analysis is used to detect anomalies i.e. deviations from normal network behaviour. Such methods can detect new forms of threats but also may create false positives. Signature-based and anomaly-based detection are combined — the former provides the benefits of a reduced detection false alarm rate, and the latter provides the advantages of a reduced detection delay. Modern NIDS's benefit from machine learning and deep learning techniques which allow modern NIDS's to reach a higher degree of accuracy, adaptability, and real time detection [14].

- **Machine Learning and Deep Learning in Network Security**

Network security has undergone revolution in threat detection through integrating the machine learning and deep learning. Historical data is analysed by ML algorithms to recognize patterns to classify network activity into normal or malicious activity [15]. Decision trees, support vector machines with random forests are

good techniques due to their high interpretability and sufficient for effective classification. ML forms a subset called deep learning where neural networks are used to model complex relationship inside the data. CNNs and RNNs have been proven to excel at working on high dimensional network traffic data, and finding hidden, subtle anomalies along with sophisticated attack patterns in such data [16].

The combination of Machine Learning (ML) and Deep Learning (DL) techniques in a hybrid model can serve to heighten detection presence and scalability, which allows it to be more apt for real time network intrusion detection in the high volumes of environments. To determine which ML model can be used, different ML models such as Logistic Regression, Random Forest, Support Vector Machine, SVM and XGBoost were developed. Of these, SVM exhibited the best performance on the given dataset to discern normal from malicious network activity with high level of precision. Convolutional Neural Networks (CNN) and Artificial Neural Networks (ANN) were used in the domain of deep learning, as they are able to maintain complex data patters and improve the anomaly detection. These models are fine-tuned on a range of datasets to allow the system to be robust and reliable to several possible network attacks [17,18].

### 2.1.2 Evolution of Intrusion Detection Systems

Intrusion Detection Systems have undergone significant transformations since their inception, evolving to address the growing sophistication of cyber threats.

### 2.2 Types of Intrusion Detection Systems

IDS can be broadly classified based on their monitoring scope and detection methodologies:

### 2.2.1 Network Intrusion Detection Systems (NIDS)

These systems are strategically positioned within the network to monitor traffic to and from all devices. They analyze passing traffic on the entire subnet and match it against a library of known attacks. Once an attack is identified or abnormal behavior is detected, an alert is sent to the administrator. NIDS function to safeguard every device and the entire network from unauthorized access [19].

### 2.2.2 Host Intrusion Detection Systems (HIDS)

HIDS operate on individual hosts or devices, monitoring inbound and outbound packets from the device only. They take snapshots of existing system files and compare them to previous snapshots. If critical system files have been modified or deleted, an alert is sent to the administrator for investigation. HIDS are particularly useful for missioncritical machines that are not expected to change their configurations [19].

### 2.2.3 Detection Methodologies

The methodologies employed by IDS to detect intrusions can be categorized as follows:

**• Signature-Based Detection**

This approach involves detecting attacks by looking for specific patterns, such as byte sequences in network traffic or known malicious instruction sequences used by malware. Although effective against known attacks, signature-based IDS struggle to detect new attacks for which no pattern is available [19].

**• Anomaly-Based Detection**

Introduced to detect unknown attacks, anomaly-based IDS use machine learning to create a model of trustworthy, activity and then compare new behaviour against this model. While this approach enables the detection of previously unknown attacks, it may suffer from false positives, as previously unknown legitimate activity may also be classified as malicious [19].

### 2.2.4 Machine Learning in Intrusion Detection

The application of Machine Learning (ML) techniques has significantly advanced the capabilities of intrusion detection systems by enabling automated analysis and classification of network traffic [19].

**• Supervised Learning**

Given table 1 shows that supervised learning techniques require libelled datasets for training and have been widely applied in intrusion detection.

| ML Technique | Explanation | Example |
|---|---|---|
| Support Vector Machines (SVM) | Effective in high-dimensional spaces, used to classify network traffic as normal or malicious [19]. | Detecting abnormal traffic patterns in a corporate network by analyzing packet features. |
| Logistic Regression | Estimates the probability of a binary response, commonly used for classification tasks. | Identifying whether a login attempt is legitimate, or a brute-force attack based on user behavior. |
| Random Forests | An ensemble learning method that constructs multiple decision trees and outputs the majority vote. | Detecting Distributed Denial-of-Service (DDoS) attacks by analyzing traffic flow features. |
| XGBoost | A scalable and efficient implementation of gradient boosting, known for high performance in classification. | Classifying network packets to detect anomalies in real-time network monitoring. |

**Table 1: Supervised Learning Techniques**

**• Unsupervised Learning**

Unsupervised learning is a powerful approach in machine learning where models learn from unlabeled data. Such as supervised learning, where models require clearly labelled examples to make predictions, unsupervised learning identifies hidden patterns and structures in data without prior knowledge of the output categories. Unsupervised learning plays a crucial role in the identification of anomalies in intrusion detection systems (IDS), which are behaviors or activities that deviate from normal network traffic.

One of the most common techniques in unsupervised learning is clustering algorithms, where similar data points are grouped based on their features. These methods can effectively identify suspicious or unusual patterns in network behavior without requiring pre-classified attack data.

**• Anomaly Detection**

Anomaly detection techniques in unsupervised learning work by identifying outliers - data points that don't fit established patterns. In network security, these outliers often represent suspicious or malicious behavior. Techniques such as Isolation Forests and Autoencoders are particularly effective here. In a company's network, most employees access a fixed set of websites and send files within a certain size range. If an internal device suddenly starts transmitting a large volume of data to an unknown IP address, anomaly detection algorithms would flag this behavior, indicating a potential data exfiltration attempt [18,20].

Unsupervised learning methods offer several advantages in intrusion detection. They can uncover previously unknown attack types, adapt to evolving threats, and operate in environments where labelled data is scarce or unavailable.

### 2.2.5 Deep Learning in Intrusion Detection

Deep Learning (DL) has revolutionized the field of machine learning by introducing algorithms capable of modelling highly complex data representations. By using neural networks with multiple layers, deep learning methods capture intricate patterns in largescale datasets, making them particularly effective for modern intrusion detection systems. With the increasing complexity and volume of network traffic, deep learning's ability to automatically learn feature representations has become invaluable in identifying sophisticated cyber threats.

**• Artificial Neural Networks (ANN)**

Artificial Neural Networks (ANNs) are the foundational architecture of deep learning, inspired by the structure and function of biological neurons. ANNs consist of interconnected nodes, or "neurons," organized into layers: an input layer, one or more hidden layers, and an output layer. Each neuron processes incoming signals and transmits the results to the next layer, enabling the network to learn complex patterns through iterative training [17].

**• How ANN Works in Intrusion Detection**

o The input layer receives raw network traffic data, such as packet size, source and destination IP addresses, and transmission timestamps. o Hidden layers transform this data through weighted connections and activation functions, learning the underlying patterns associated with normal and malicious traffic.

o The output layer provides the final classification, typically indicating whether the traffic is normal or malicious.

Suppose an ANN is trained on historical network, traffic data. When it encounters a new traffic flow with unusual port access patterns and high data transfer rates, the ANN recognizes this as a potential port scanning or data exfiltration attempt and raises an alert.

One of the major advantages of ANNs is their flexibility - they can handle both supervised and unsupervised learning tasks, making them suitable for various intrusion detection approaches.

**• Convolutional Neural Networks (CNN)**

Convolutional Neural Networks (CNNs) were originally developed for image recognition tasks, but their ability to capture spatial hierarchies and feature relationships has made them highly effective for network traffic analysis as well. In the context of intrusion detection, CNNs treat network traffic data as a grid-such as structure, enabling them to learn spatial dependencies between features and identify abnormal behaviour more efficiently.

How CNN Works in Intrusion Detection

• Convolutional Layers apply filters to input data, detecting local patterns such as frequency changes or unusual packet sequences.

• Pooling Layers down sample the data, reducing dimensionality and focusing on the most critical features.

• Fully Connected Layers interpret the extracted features and produce the final classification decision.

A CNN can detect temporal patterns, such as sudden spikes in data volume or irregular access requests, signalling a potential Distributed Denial-of-Service (DDoS) attack.

### 2.2.6 Hybrid Models in Intrusion Detection

ML and DL techniques have become integrated to form hybrid models to further improve the detection performance by combining the advantages of each approach.

• ML-DL Hybrids

An integrating of traditional ML algorithms with DL architectures allows the utilization of deep feature learning together with structured feature extraction to achieve better detection rates.

• Anomaly and Signature-Based Hybrids

However, a key is to merge this anomaly detection with signature-based methods to get a nice balance because each method has limitations when used alone.

**• Challenges and Considerations**

However, several challenges still remain in development and deployment of intrusion detection systems.

**• False Positives**

Detecting unknown threats, anomaly-based systems suffer a bane of low false positive rates which causes alert fatigue in the security personnel.

**• Data Quality and Volume**

ML and DL models are very dependent on the quality and quantity of training data.

And of course, inadequate or biased datasets can propagate to the model.

**• Real-Time Processing**

In order to catch threat as soon as possible NIDS must be able to process and analyze data in real time.

**• Future Directions**

Current research in intrusion detection systems concentrates on several key areas in order to increase its effectiveness and resilience.

**• Explainable AI**

Uncovering subject areas in high impact intrusions and also developing models that not only detect intrusion but also provide interpretable explanations to assist cybersecurity analyst to understand and address them are also a growing area of interest.

**• Adversarial Robustness**

It is one of the major challenges for modern intrusion detection systems that they are susceptible to adversarial attacks. These are attacks where malicious activity or misclassification is achieved by deliberately provided inputs specifically targeted at fooling machine learning and deep learning models. In the context of network security, adversarial robustness is a system's ability of detecting with high performance even when input traffic is adversarial. Efforts in research are made to construct techniques like adversarial training [20], where the models are trained with adversarial examples to increase their resilience. Also, they utilize defensive distillation and feature squeezing methods to reduce the damaging effect of adversarial manipulation on the detection [21].

**• Integration with Other Security Measures**

Network Intrusion Detection Systems (NIDS) are very important components of the arsenal of the cyber warrior to identify the sources and mitigate the threats and their effectiveness is best when incorporated in a bigger picture of the overall cyber security framework. NIDS can be written in combination with other security tools equivalent to firewalls, Intrusion Prevention System (IPS) and Security Info and Event Management (SIEM) system to bolster the network security. A layered defence strategy was adopted to guarantee that various types of security tackle different attack vectors and thus offer a comprehensive coverage for the handling of sophisticated cyber threats [22]. It also allows for automated incident response, based on the case, predefined actions are triggered such as blocking traffic or quarantine, to limit the time required to contain any potential breaches.

### 2.2.7 Performance Metrics and Evaluation

New performance metrics for the evaluation of the effectiveness of intrusion detection systems for distinguishing between normal and malicious traffic are introduced.

### 2.2.8 Datasets for Intrusion Detection Research

As ML and DL based intrusion detection systems develop and evaluate, good quality dataset, which provides network traffic that

reflects real world network traffic and attack scenarios, is required. UNSW-NB15 dataset is one of the well-known and large dataset that is being used for the evaluation and design of intrusion detection systems (IDS). This work has later been extended (e.g., CAN 1) to punish or disallow authors to submit classes of anomalies which have the potential to overcrowd the 'outlier' label. According to UNSW-NB15, it captures a variety of network behaviours and modern forms of attacks, providing an up-to-date and realistic network traffic. The data was generated by the IXIA Perfect Storm tool that emulates traffic from normal as well as malicious traffic in real world network environments. Specifically, nine types of attacks are included in the dataset, i.e., DoS (Denial of Service), exploits, backdoors, reconnaissance, etc., along with 49 features that are grouped as flow features, basic features, and content features. UNSWNB15 [23] is selected as the target of attack traffic, due to the balanced mix of normal and attack traffic and the detailed feature sets. It is essential to contemporary cybersecurity research as they ensure the development of robust and adaptable IDS models that are able to reflect in a changing cyber threat as well as real world scenarios.

#### 2.2.9 Feature Engineering and Selection

In order to achieve effective intrusion detection, the quality and relevance of the features extracted from the network traffic data plays a fundamental role, because the features identify and discover either normal or malicious activity from the network traffic data. What is more, feature engineering is key in transforming the raw network data into comprehensible features containing suspicious behaviour related patterns. Commonly used features fall into several categories: packet-level features, such as packet size, protocol type, and time-to-live (TTL) values, provide low-level insights into individual data packets; flow-level features offer aggregated metrics such as the number of packets and bytes exchanged between a source and destination within a session; statistical features capture descriptive statistics such as the mean, variance, and standard deviation of traffic attributes over specific time windows; and temporal features, such as inter-arrival times and session durations, help identify time-based patterns and anomalies. To optimize model performance and prevent overfitting, Recursive Feature Elimination (RFE), Mutual Information and Principal Component Analysis (PCA) are used to preprocess model experimentation to select the most suitable features. These methods determine the most informative and relevant features, thus, ensuring that the machine learning (ML) or deep learning (DL) models emphasize on the important data attributes and increase efficiency and generalization [24]. With combination of feature engineering and proper feature selection, IDS models gain more accurate and robustness in detecting new cyber threats.

#### 2.2.10 Model Training and Optimization

Hyperparameters and optimization techniques are very hard to tune to build a robust ML and DL model for intrusion detection. k-fold and stratified sampling provide methods for validating models that make guarantees about generalizing to unseen data. Popular approaches to do hyperparameter tuning, balancing performance and computational efficiency are grid search, random search, and Bayesian optimization [25]. To address such problems, deep learning techniques like dropout, batch normalization, early stopping allow the models to handle complicated traffic patterns without memorizing noise [26].

**Real-Time Processing and Deployment Challenges**

Inference with ML and DL based intrusion detection systems in the production environments is a challenging problem due to issues involved with latency, scalability, and resource constraints. Efficient models are necessary for real time detection because there is a need for detecting every packet in massive network traffic with minimal delay. And the computational overload is reduced by using techniques like quantization of model, pruning, and knowledge distillation [27]. Also, horizontal scaling can be made through distributed architectures and cloud-based solutions which can accommodate large scale networks [28].

### 2.3 Literature Review

#### 2.3.1 Introduction

This literature review checks on the evolution of Network Intrusion Detection System (NIDS), but this time appealing to Traditional detection techniques to Hybrids models that apply both Machine Learning (ML) and Deep Learning (DL). Finally, it also discusses previously investigated studies of this kind and their methods, finally bringing forth the results and contributions to the development of more efficient and accurate intrusion detection systems [29].

#### 2.3.2 Traditional Intrusion Detection Approaches

Traditionally, intrusion detection was done based on signature and anomaly. Snort and other systems based on signatures rely on recognizing the attack patterns. However, they are effective against known threats, but new or obfuscated attacks make them very hard to defeat. On the contrary, anomaly-based systems set up a baseline of normal network behaviour and any deviation from it is considered suspicious [30].

#### 2.3.3 Emergence of Machine Learning in NIDS

ML techniques brought new advances in the domain of NIDS through the integration of techniques to learn from data and to improve the detection abilities of a system. Logistic Regression, Random Forests, and Support Vector Machines (SVM) etc. have been used as algorithms to classify the network traffic as normal or malicious. Furthermore, SVMs are found to be robust in coping with high dimensional data as they are well suited for an intrusion detection task [31].

#### 2.3.4 Deep Learning Advancements

With DL, there were models that could extract features from raw data without any modelling required. CNNs and ANNs were most prominent among NIDS applications. Spatial hierarchical CNNs have been used to analyse traffic patterns of network traffic, and ANNs have been used because they are versatile to model nonlinear relationships [32].

### 2.3.5 Hybrid Models: Combining ML and DL

The hybrid models adopting both ML and DL techniques with a goal of improving the detection accuracy and scalability are also proposed by the researchers. Such models combine best of both worlds to make network attacks more robust to a variety of network attacks. SVMs combined with CNNs have also showed promise in capturing linear and nonlinear patterns in network traffic and thus improving the detection performance [33].

**• Ensemble Learning in Intrusion Detection**

One of the techniques used in intrusion detection system is the ensemble learning which combines several classifiers to better predict their accuracy. The reduction of the total error and improvement of the detection performance can be performed by aggregating the outputs of the individual models using ensemble them. Neural classifiers have been combined to detect distributed denial-of-service (DDoS) attacks with better prevention of false positives and enhancing the overall system reliability [34].

The comparison of the recent AI and ML studies on network security determines that the two methods are different, but complementary perspectives to improve intrusion detection and threat mitigations. A multistage AI enabled framework for addressing challenges like zero-day attacks, out of distribution samples and adversarial evasion techniques was proposed whose components were one classifier and two specialized autoencoders composed in a sequential order as three Deep Neural Network architectures (DNN) [35]. The framework was validated on benchmark intrusion detection dataset exhibiting 98.5% average detection accuracy which emphasizes the strength of multistage AI models in detecting both known as well as previously unseen attack patterns. On the contrary, a systematic review of MLs used for network security focused on the real time analysis of large amounts of network data to find out how the supervised learning, unsupervised learning and deep learning are applied [36]. ML was evaluated against cases studies and real-world examples of intrusion detection, malware detection, anomaly detection and behavioural analysis. In addition to this, this review also proposed ways of dealing with significant deployment challenges for ML models like federation learning, adversarial ML, and explainable AI. Though the first study mainly concentrated on creating a robust, multistage detection framework having high accuracy and adaptability, the other one added a broader perspective about making Machine Learning techniques available in such domains alongside their applications in AI driven network security. These studies together stress the need for a good model performance, being scalable, and applicable to real world to propel the state of the art in modern intrusion detection systems [35,36].

Recent studies aim at detecting intrusion and DDoS in Software-Defined Networking (SDN) underline the significance of modern network security requirement and presence of advanced techniques that aid in intrusion detection and DDoS detection. A recent study has made a complete study of DDoS attack detection in SDN and divided DDoS attack into two types and five subtypes according to the attack nature of the host of SDN vulnerabilities [37]. The ML based and threshold based are revealed to be the two methods mostly used to detect DDoS, which it classified DDoS detection methods into 5 categories and 46 subcategories. The strengths and weaknesses of each method are analyzed, and the evolution of each method was discussed, followed by open challenges and possible directions of future work towards improving DDoS detection in SDN. In lieu of that, a second study, made use of deep neural networks (DNN) in intruding detection, on both a network level as well as a host level, to accommodate the dynamic and ever evolving nature of cyberattacks [38]. Hyperparameter tuning and 1,000 epochs were used to optimize the model, and the DNN was studied on multiple public datasets, i.e., KDDCup 99, NSL-KDD, UNSW-NB15, Kyoto, WSN-DS and CICIDS 2017. We showed that DNN is effective in inferring the high dimension feature representation and is scalable for discovering the evolving threat. Contrary to the effort in [37], which concerned on the DDoS detection on SDN, includes a general evaluation of IDS performance to include several datasets and emphasizes that DNN can catch complex attack patterns and is resilient to change [38].

Recent studies on the application of AI in intrusion detection and botnet attack defeat have also conducted that advanced AI techniques and can be effective for modern network security. The meta-RF GNB (MRG) model proposed in one study for real time network attack detection that combine Random Forest (RF) and Gaussian Naïve Bayes (GNB) achieved accuracy of 99.99 percent and mean cross validations accuracy of 99.94 percent along with the minimal standard deviation of 0.00002. In this, the Server based Network Attack (SNA) dataset was collected using a virtual network environment with Ubuntu Base Server, Kali Linux for attacks and Wireshark for data capture of UDP, SYN and HTTP flood attacks [39]. The performance of the MRG model was further statistically validated by a t-test showing its better performance than other models. On the other hand, another study gave an all-inclusive survey of botnet identification on IoT situation utilizing DL based IDS; it examined different methodologies, headways, and hindrances [40]. Finally, it focused on the increasing demand for secure mechanisms that enforce security on IoT networks, along with it conducting qualitative comparison of related research to demonstrate main contributions and guide future research directions was interested in real time detection with regards to model efficiency whereas exhibited a wider scope of IoT security and DL based IDS capability for botnet threat mitigation. Regardless of the study, the AI techniques clearly complement network protection and building good datasets through intense and sophisticated frameworks is crucial in addressing the evolution of cyber threats [39,40].

In the one study, a Hybrid Intelligent Intrusion Detection System (HIIDS) was proposed that combined Spark MLlib-based anomaly detecting classifiers like Logistic Regression (LR) and Extreme Gradient Boosting (XGB) with a Long Short-Term Memory Autoencoder (LSTMAE) for misuse attack detecting [41]. Moreover, through 10fold cross validation on the ISCXUNB dataset, this system has successfully learned meaningful feature representations from the large scale, unlabelled network traffic

data and achieved 97.52% accuracy which outperforming all the traditional methods in the area of detecting. In another study, the authors secured the NFV environments especially in the sensors and IoT networks by detecting anomalies to mitigate cybersecurity issues [42]. In the paper, various ML algorithms were evaluated in terms of how well they can identify network-based anomalies for the purposes of enhancing NFV security. Although focused on designing a high accuracy hybrid intrusion detection system, discussed security vulnerabilities of virtualized network, pointing out the possibility of utilizing the ML driven approaches to protect the current network infrastructure. The results of both studies emphasize the increasing importance of AI based techniques in securing network and reliability of the digital systems [41,42].

### 2.3.6 Comparative Analysis of Hybrid Intrusion Detection Systems (IDS) and the Proposed System

During the past couple of years, Intrusion Detection Systems (IDS) have been developed using the most enhanced Machine Learning (ML) and Deep Learning (DL) techniques. A set of contemporary hybrid IDS models are compared against the proposed system in terms of their methodologies, the limitation as well as the strengths, and the performance metrics

1. Signature-Based Intrusion Detection Using Machine Learning and Deep Learning

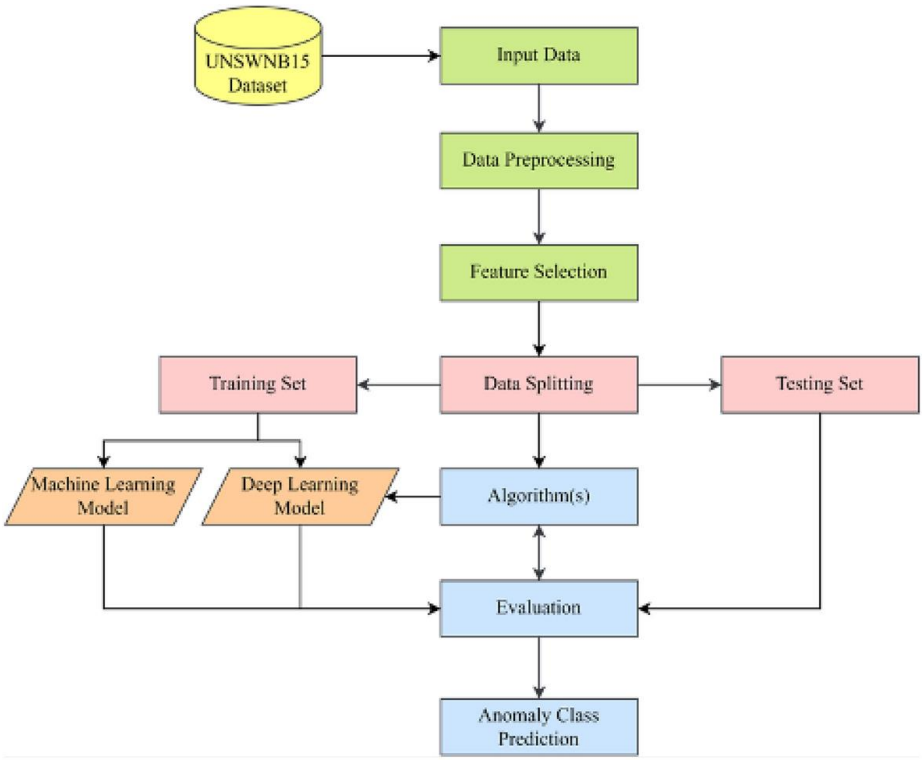



**Figure 1:** Signature-Based Intrusion Detection Using Machine Learning and Deep Learning [43]

In accordance with the above-mentioned areas that require our focus, a study has been made which will be on the development of a hybrid model that combines the strengths of Machine Learning (ML) and Deep Learning (DL) methods for intrusion detection in order to improve the detection accuracy and adaptability to the changing threats [43]. Some advantages of this method include use of ML and DL advantages to enhance the detection accuracy of known and unknown attacks. Although, similarly to Convolutional Neural Networks, it also has a few drawbacks like a need for large training data to perform as well as possible and added computational complexity resulting from merging multiple models together. The proposed system applies a hybrid approach by means of combining ML and DL models as well, though with a greater focus on model optimization and fine-tuning process. This focus improves the system's computational overhead and thus makes it more effective and responsive regarding network intrusion detection in real time.

2. Enhancing Intrusion Detection Which Hybrid Machine and Deep Learning Approach

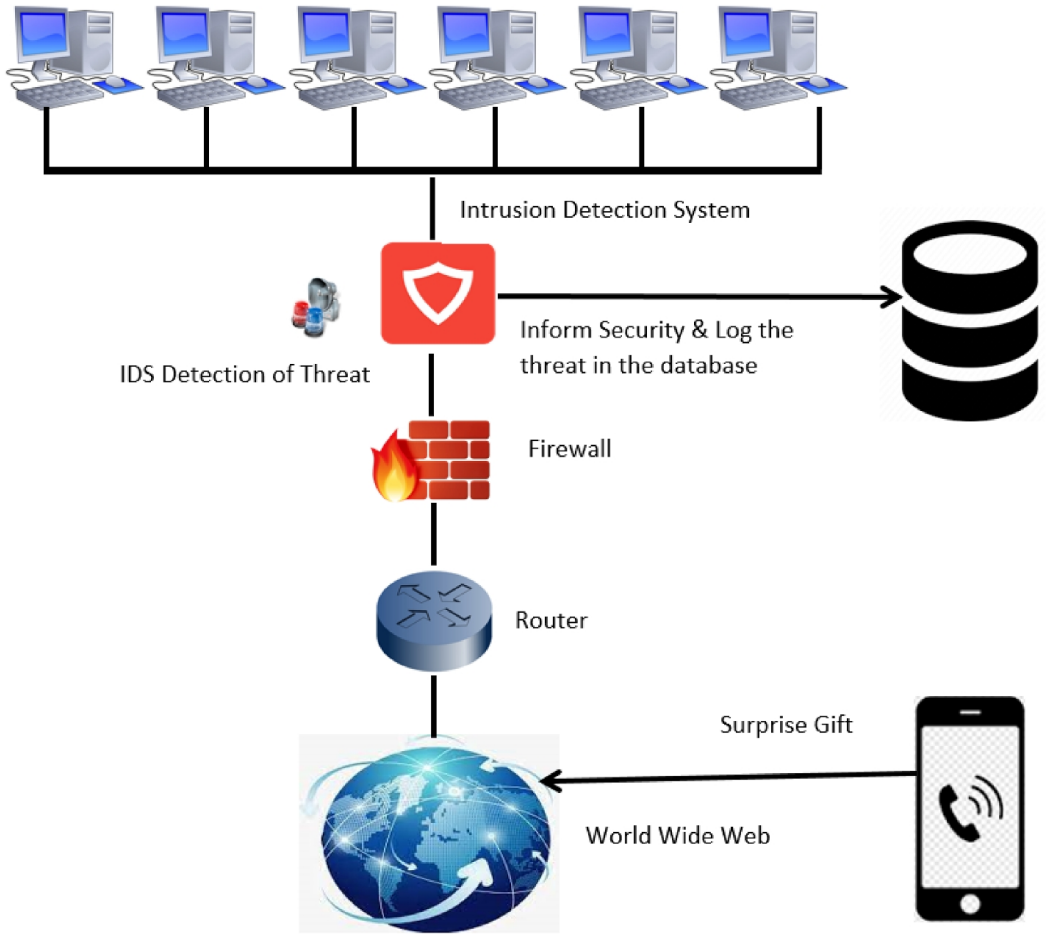



**Figure 2:** Enhancing Intrusion Detection Which Hybrid Machine and Deep Learning Approach [44]

Using Convolutional Neural Networks (CNN) and Extreme Gradient Boosting (XGBoost) for feature extraction and classification, the paper, which is implemented in a hybrid model that combines the advantages of both Machine Learning (ML) and Deep Learning (DL) to overcome the cons of current Intrusion Detection Systems (IDS), is a study [44]. The experiments in this work demonstrate good properties, such as the ability of CNNs to effectively extract feature and boost detection accuracy, and XGBoost to well classify the detection. At the same time, the model had certain limitations like higher resource consumption owing to the intricacy of accommodating two powerful techniques together, and the demand for precise tuning to handle performance and computation efficiently. The proposed system integrates ML and DL models for intrusion detection like this study but focuses more on optimizing the hybrid model to scale the hybrid model in such a way that it is not only scalable but also efficient enough especially in the case of high traffic network environment. Model fine tuning is focused here to avoid expensive computation and at the same time achieve high detection performance.

3.Development of Hybrid Intrusion Detection System Leveraging Stacked Feature Selection Methods

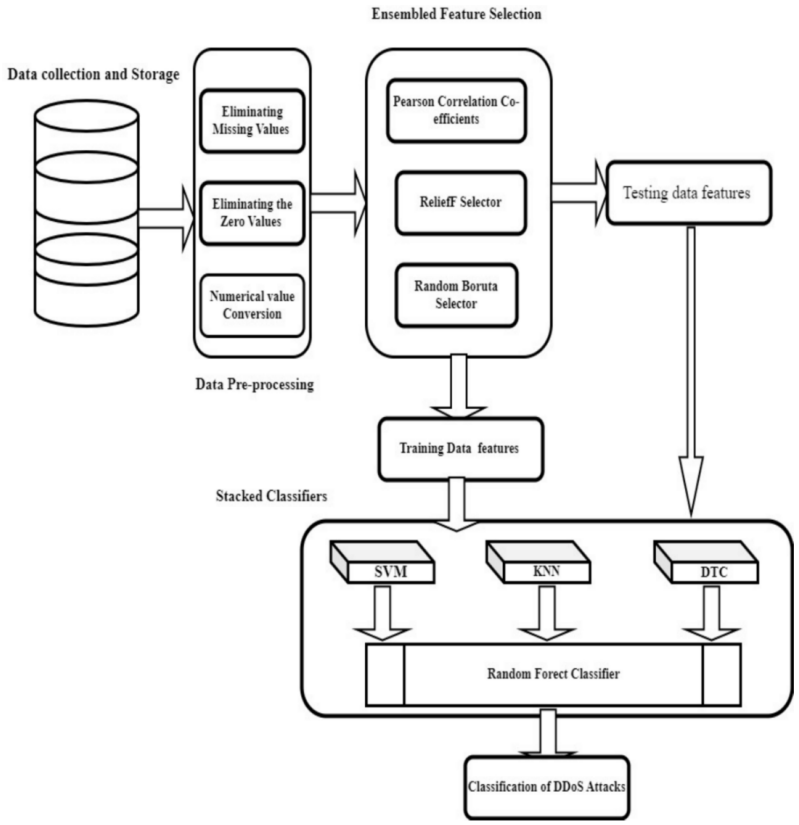



**Figure 3:** Development of Hybrid Intrusion Detection System Leveraging Stacked Feature Selection Methods [45]

Intrusion detection system (IDS) proposed a hybrid Intrusion Detection System, which employed stacked feature selections methods, namely Random Boruta Selector (RFS) and Relief, and Machine Learning (ML) algorithms in an attempt to improve detection accuracy by selecting the most important features for distinguishing against malicious attacks [45]. In addition, notable strengths of the approach were improved detection accuracy through effective feature selection and reduction in the dimensionality, hence quicker processing times. These advantages notwithstanding, there were some limitations of the method; including the possibility of variability of how effective a feature selection turns out to be for a given set of data and the risk of crucial information loss in the feature reduction process. On the other hand, the proposed system is also integrated with feature selection, which is a very important step in the preprocessing pipeline of the proposed system, but the set of advanced techniques is utilized to ensure that important features are retained, with high accuracy of detection and optimization of the model efficiency and performance.

4. Hybrid Machine Learning Models for Intrusion Detection in IoT

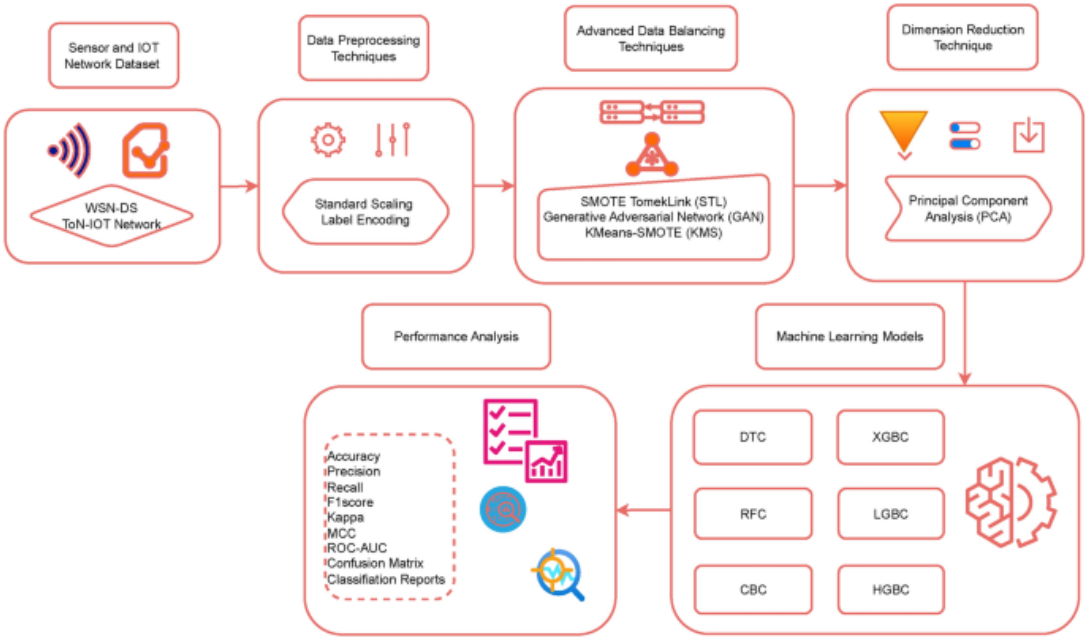



**Figure 4:** Hybrid Machine Learning Models for Intrusion Detection in IoT [46]

In this research, a hybrid approach for IoT intrusion detection by combining standalone Machine Learning (ML) models, i.e., Random Forest (RF), XGBoost, KNearest Neighbors (KNN), AdaBoost, into a voting-based ensemble classifier has been investigated [46]. The research approach was to use the strengths of individual algorithms to address problems in data complexity and scalability and increase detection accuracy. With these strengths, this ensemble method managed to achieve a notable detection accuracy by combining its classifiers and improve its robustness against different cases of attacks. Additionally, it can bring about some limitations such as multiple models' involvement, which increase the computational complexity and difficulty of achieving real time detection. Compared with the proposed system, in particular, ensemble techniques are also used but for the purpose of learning how to combine models to achieve real time detection whilst using resources efficiently, trading off performance and scalability in realistic network environments.

5. Harnessing Advanced Hybrid Deep Learning Model for Real-Time Intrusion Detection

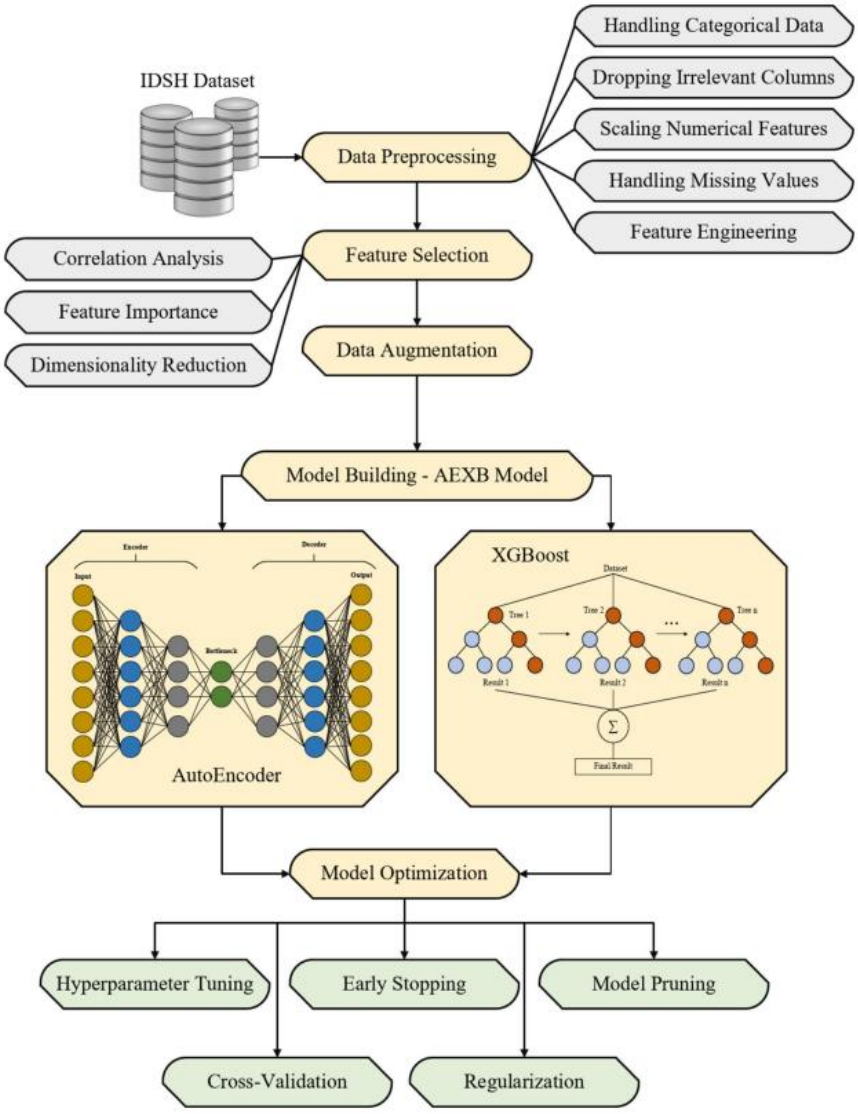



**Figure 5:** Harnessing Advanced Hybrid Deep Learning Model for Real-Time Intrusion Detection [47]

In the study above, a hybrid approach to IoT intrusion detection was investigated using a voting based ensemble classifier in which a set of independent Machine Learning (ML) models namely, Random Forest (RF), XGBoost, K-Nearest Neighbors (KNN), AdaBoost were combined [47]. The goal was to leverage the strengths of separate algorithms in order to alleviate problems of data complexity and scalability, as well as increase detection accuracy. Secondly, the detection accuracy could be effectively enhanced due to the combination of multiple classifiers in this ensemble method, as well as robustness to various types of attacks. On the other hand, it brought along some drawbacks such as higher computational complexity that arises with multiple models employed, and difficulty in obtaining real time detection. The proposed system also employs the ensemble but optimizes the combination of the model to accommodate the real time detection requirement and to use resources efficiently with taking high performance and the scalability into real network environments.

The given table 2, provides a detailed side-by-side comparison of existing systems with the proposed system [43-47].

| System and Study | Methodology | High Accuracy | Real-Time Detection | Low Computational Cost | Scalability | Feature Selection Efficiency | Detection of Unknown Attacks |
|---|---|---|---|---|---|---|---|
| Signature-Based IDS (ML & DL Hybrid) | Combines ML and DL models |  |  |  |  |  |  |
| Hybrid ML-DL Model (XGBoost + CNN) | CNN for feature extraction + XGBoost classifier |  |  |  |  |  |  |
| Stacked Feature Selection Hybrid Model (RFS + Relief) | Feature selection + ML algorithms |  |  |  |  |  |  |
| Ensemble ML Models (RF, XGBoost, KNN, AdaBoost) | Voting-based ensemble classifier |  |  |  |  |  |  |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| AEXB Model (AutoEncoder + XGBoost) | AutoEncoder for feature extraction + XGBoost |  |  |  |  |  |  |
| Proposed Hybrid ML-DL System | Optimized ML and DL models + feature selection |  |  |  |  |  |  |

**Table 2: Gap Analysis**

### 2.3.7 Conclusion

The comparative research shows that although present hybrid IDS models made a significant contribution to intrusion detection, they still leave several problems in a number of parameters such as computational complexity, real time and scalability. These difficulties are tackled by the suggested system, where's the machine learning and the deep learning approaches are integrated well while efficient feature selection, model simplification and real time detection capabilities are considered. The intrusion detection system generated from this method is guaranteed to be resilient and scalable for today's network environment.

### 2.4 Research Objectives

• To develop a Robust and Scalable Hybrid Intrusion Detection System (IDS) combining Machine Learning (ML) and Deep Learning (DL) techniques to enhance the accuracy and efficiency of network intrusion detection.
• To Implement Anomaly and Signature-Based Detection Approaches to identify known and unknown network threats.
• To optimize Model Performance through Fine-Tuning and Feature Selection to identify the most effective approach.
• Design a User-Friendly Web Application for Real-Time Monitoring and manage network activity easily.

### 2.5 Limitations

| Limitation | Description |
|---|---|
| High Computational Requirements | The hybrid model combining ML and DL techniques may demand significant processing power and memory, which could affect real-time performance on resource-constrained systems. |
| Data Dependency | The accuracy and efficiency of the system depend heavily on the quality and diversity of the training dataset, and poor or imbalanced data may lead to biased predictions. |
| Complex Model Training and Tuning | Fine-tuning multiple ML and DL models, selecting optimal hyperparameters, and integrating them into a hybrid system require significant time and expertise. |

| | |
|---|---|
| Real-Time Detection Challenges | Maintaining low latency and high detection accuracy simultaneously in high-volume network environments can be difficult, especially when processing large data streams. |
| Limited Generalization | The system's performance may vary when deployed in different network environments with different traffic patterns and attack types, requiring additional model retraining. |
| Feature Selection Sensitivity | Inefficient feature selection could lead to the loss of important information or the inclusion of irrelevant data, impacting the detection accuracy. |
| Web Application Usability Constraints | Designing a user-friendly, responsive, and informative web interface that meets the needs of network security teams may require iterative improvements and feedback. |

**Table 3: Limitations**

### 2.6 Research Aim

The aim of this research is to develop a robust and scalable hybrid Intrusion Detection System (IDS) using Machine Learning (ML) and Deep Learning (DL) techniques. This system will efficiently identify network activities as normal or malicious in real time, detect various types of attacks, and provide a user-friendly web interface for network security management.

### 2.7 Research Questions

i. How can a hybrid Machine Learning (ML) and Deep Learning (DL) model improve the accuracy and efficiency of network intrusion detection? ii. What combination of anomaly-based and signature-based detection methods ensures the identification of both known and unknown network attacks?

iii. Which ML and DL algorithms, after fine-tuning, provide the best performance for real-time threat detection in high-volume networks? iv. How can a web-based interface enhance the usability and monitoring capabilities of an Intrusion Detection System for network security teams?

See Appendix 2

### 2.8 Research Gap

The current Intrusion Detection Systems experience multiple constraints when applied to high-volume real-time network environments. The inability of signature-based detection methods to detect emerging threats combines with the substantial number of false positives from anomaly-based methods according to Khraisat [48]. Multiple IDS systems work exclusively with ML or DL procedures without utilizing the complementary features of both methods by Vinayakumar [49]. Large network traffic volumes cause performance degradation since most systems do not scale well (Shone et al., 2018). The existing IDS deployments fail to provide intuitive interfaces which allow network security teams to monitor threats in real time.

By fusing of ML and DL features which mix anomaly and signature detection methodologies the hybrid IDS can eliminate current limitations of accuracy and false alarms. When operated with relatively large quantities of data, the system enables to optimize the model optimization with discriminant levels of performance. Thereby, with its Flask framework interface, the hybrid IDS provides the scalability and the real time processing capabilities for large network security. teams to operate. The integrated method delivers a dependable threat detection solution for current networks which combines efficiency with user-friendly operation.

### 2.9 Requirements

#### 2.9.1 Functional Requirements

• Real-Time Threat Detection: The system must identify network activity as normal or malicious in real-time.

• Attack Classification: It should accurately classify detected attacks into specific categories.

• Data Input Support: Accept network data such as IP addresses, packet details, and data transfer rates for analysis.

• Hybrid Model Integration: Implement both Machine Learning (ML) and Deep Learning (DL) techniques for enhanced accuracy.

• User Interface (UI): Provide a web-based interface using Flask for monitoring and managing network activity.

#### 2.9.2 Non-Functional Requirements

• Performance: Ensure low latency and high efficiency even in high-traffic network environments.

• Scalability: The system should handle large volumes of network data without performance degradation.
• Usability: The web application must be user-friendly, intuitive, and accessible for network security teams.
• Reliability: Maintain consistent detection accuracy and system uptime with minimal failures.
• Security: Protect the system and data from unauthorized access and ensure secure data transmission.

## 3. Methodology

### 3.1 Chapter Introduction

This chapter presents the methodology adopted to design and implement a hybrid Network Intrusion Detection System (NIDS) using machine learning and deep learning techniques. It outlines the research design and approach, detailing the system's architecture and model development process. The UNSW-NB15 dataset was used for data collection, followed by preprocessing steps including cleaning, normalization, and feature selection. Multiple algorithms were evaluated, and the best-performing models—XGBoost and ANN—were selected to form the hybrid model. The chapter concludes with the evaluation and validation process, ensuring the system's accuracy, reliability, and real-time effectiveness in detecting cyber threats.

### 3.2 Research Design

#### 3.2.1 Research Approach

The Network Intrusion Detection System constructed, is a hybrid research methodology that integrates the machine learning (ML) and deep learning (DL) models. The signature-based technologies are for detecting known threats while the anomaly is used for unknown security threats. An extremely high level of applicability occurs through a unified detection methodology.

More successful at detecting well defined patterns of attack, they have difficulty identifying zero-day attacks, which are defined as having no known signatures. Anomaly based methods do well in identifying previously unknown threats as they are able to learn what normal system behavior is (which is not the case in the rules-based methods), but struggle with high false positive rates. A network intrusion detection system can achieve both, accuracy and adaptability by combining these two approaches: a reduction in false positives and an increase in coverage of detected threats and capabilities.

#### 3.2.2 Machine Learning and Deep Learning Integration

Many machine learning models and deep learning frameworks are assessed over a lot of data and the best efficient way of intrusion detection is established. SVM turns out to be the best performer among the evaluated models - Logistic Regression, Random Forest, Support Vector Machine (SVM), and XGBoost. Then, the optimal model is implemented in a Flask based web app for real time detection, so as to ensure efficient and precise monitoring of network traffic.

#### 3.2.3 System Architecture

Upon accessing the NIDS dashboard, users first select a CSV file—containing features such as duration (dur), protocol (proto), service, state, byte counts (sbytes, dbytes), packet statistics (spkts, dpkts), flow-level counters and flags—or alternatively enter these parameters manually via the Manual Input option. Once submitted, the data undergoes a three-stage preprocessing pipeline: noise removal (discarding outliers and imputing missing values via median or k-nearest neighbor methods), min–max normalization (to ensure equitable scaling across continuous features), and feature selection (using recursive elimination and mutual information ranking to isolate the most predictive attributes). The refined feature set is then passed to the hybrid detection engine—built around XGBoost and a feed-forward artificial neural network—which applies both anomaly-based profiling (to flag deviations from established "normal" patterns) and signature-based matching (against a library of known attack vectors). In real time, each network flow is classified as normal or malicious and, when malicious, is further categorized (e.g., DoS, DDoS, malware). Detected events immediately trigger alerts that detail timestamps, affected IP addresses, indicators of compromise, and inferred attack types. Through the Flask-powered interface, analysts can visualize live traffic metrics via interactive charts, review historical reports of attack trends and model performance, and export detailed logs for compliance auditing or forensic analysis—thereby delivering low-latency, high-precision intrusion detection and empowering rapid, data-driven responses in high-volume network environments.

### 3.2.4 System Design

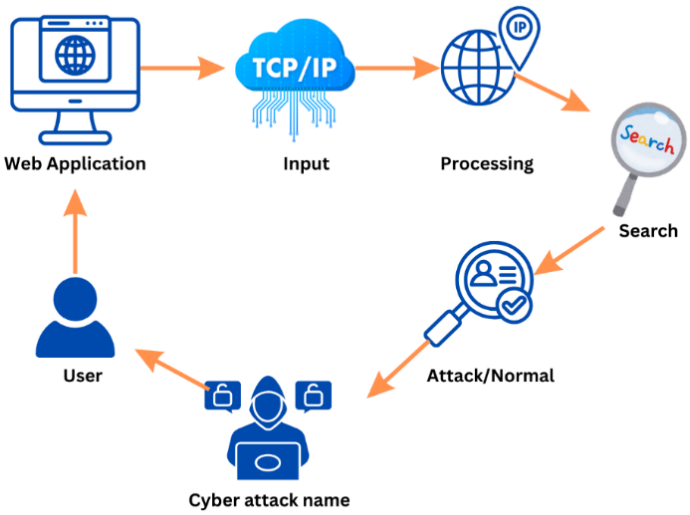


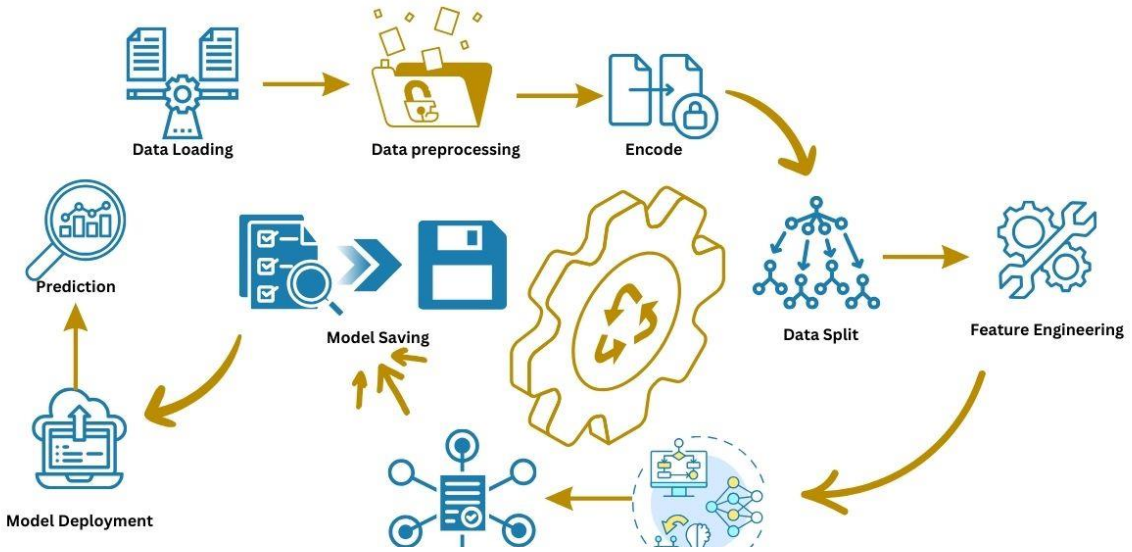

**Figure 6:** System Design

### 3.2.5 System Architecture

**• Data loading**

Data loading is a fundamental prerequisite for constructing a comprehensive network intrusion detection system (NIDS). The NIDS system initiates its operation by importing network traffic data from the UNSW-NB15 dataset records. Supervised learning models can identify normal and malicious activities by utilizing labelled network traffic data that contain descriptions of network connection behavior. Researchers widely accept the UNSW-NB15 dataset because to its realistic traffic patterns and many attack types, rendering it an ideal platform for evaluating intrusion detection systems.

The essential data verification phase occurs after the data import sequence as the second critical stage. The assessment requires us to examine all records for missing data points, extraneous entries, and inconsistencies, as these factors will undermine the precision of our models. The appropriate handling of missing values is crucial to avoid bias, necessitating the use of imputation or removal techniques. The model necessitates the elimination of duplicate records to prevent the generation of repetitive patterns from the data. All records must undergo consistency tests to verify their correct format and valid attributes for both features and labels. Data validation establishes a robust basis for preprocessing by ensuring the reliability and quality of the training dataset information.



**• Data Preparation**

Preprocessing comprises modification of raw packet traffic to adhere to the requirements of those algorithms that can process the packet traffic. Three fundamental steps of the preprocessing process are cleaning, normalization, and encoding operations.

The first stage of cleaning operations removes absent data as well as extraneous or irrelevant inputs. Several deficiencies of UNSW-NB15 dataset are present, including missing data points, and irrelevant attributes, effecting the model efficacy. Features with considerable missing values as well as those with just 1 possible value can be omitted from further analysis as they do not carry any analytical insight.

With the normalization procedure and scaling techniques used for normalizing the value range of numerical data elements, like packet size field, traffic measurement rate is standardized to ranges. In order to train machine learning models, all the features are expected to contribute uniformly. Min-Max scaling and Z-score normalization techniques maintain features with the large ranges from overwhelming the learning algorithm, thus keeping the machine learning process in equilibrium.

After that the features with categorical attributes should be encoded into numerical values. Since the network traffic data is considered machine learning compatible, it requires protocol types, service names, and flag indicators encoding to accept appropriate decisions from algorithms. The two tactics utilize one hot encoding or label encoding strategy when each class category is indicated by unique integer or each item in class category is indicated by a specific binary. Accurate encoding techniques are therefore necessary to know how category variables generate network activities in the model.

**• Data Splitting**

The UNSW-NB15 dataset necessitates a division into three segments to evaluate model efficacy: training data, validation data, and distinct testing data. The model's parameters are trained using the 70% data allotment designated for the training set. The model and performance are refined utilizing the 20% validation set, which also mitigates overfitting concerns. The test set comprises 10% of the data for assessing model generalization performance impartially.

Every subset of stratified sampling maintains the same class distribution as the original population. The identification of network invasions necessitates this approach, as most network operations are innocuous, with harmful incidents being infrequent. Stratified sampling prevents the model from acquiring class-based biases towards the dominant category.

**• Feature Engineering**

Model performance is enhanced through feature engineering, which identifies ideal features while reducing their quantity. Correlation analysis enables the identification of variables that possess predictive potential based on their connections with target labels. Mutual information is a useful tool for measuring variable dependencies, identifying complicated linkages that typical correlation techniques may ignore. Random Forest trees provide a feature relevance assessment by evaluating the influence of each variable on prediction outcomes.

High-dimensional feature spaces can be condensed into a reduced set of pertinent variables by dimensionality reduction employing Principal Component Analysis (PCA), which preserves much of the original information. Reducing dimensions aids researchers in addressing the issue of dimensional complexity, accelerates training, and diminishes such aslihood of model mismatch.

**• Model Development**

The model training phase encompasses algorithm evaluation to identify the most appropriate model. Diverse categorization methods, ranging from SVM to Random Forest, including XGBoost, ANN, and CNN, integrate interpretability with robust predictive capabilities.

The optimization of model parameters via hyperparameter tuning enhances performance by identifying factors such as learning rate, strength, and depth, among others. The Grid Search approach evaluates every conceivable combination, while Random Search employs efficient sampling of configurations. Bayesian Optimization use probabilistic models to autonomously identify promising regions of parameter variables through iterative processes.

In K-Fold cross-validation, a technique enhances generalization by partitioning the training data into several subsets. The model conducts training on various segments of its data prior to validating itself on untrained data, hence ensuring effective functioning across numerous scenarios.

**• Assessment of Model Performance**

To estimate forecast accuracy, the model requires evaluation through many performance indicators. Accuracy of the model shows the ratio between correct predictions sum to all of the cases and Precision indicates how effective the model is in reducing false positive hypothesis. The F1-score is a better alternative to identify true positive outcomes because it combines memory sensitivity and precision. The ROCAUC analysis is designed to find how good a model is at distinguishing between different classes over a range of thresholds.

Errors are visually analyzed by means of a confusion matrix that gives us true positives, false negatives, true negatives and false positives. This analysis of erroneous classifications helps determine what is missing from the model by offering both the frequency of the false positive errors that result in wasted alarms and unfound actual risks.

**• Preservation of Model**

The model's deployment preserves its serialized version following the identification of the SVM as the most efficient model within this architectural framework. Joblib and pickle offer libraries for serializing models that preserve the training parameters in conjunction with the setup.

The versioning approach preserves records of model alterations while ensuring temporal traceability. Upon saving, a model acquires metadata regarding training specifics and performance outcomes, together with performance metrics and hyperparameter data for subsequent evaluations and comparisons.

**• Implementation of the Model**

The NIDS attains operational status by integrating the learned model into a Flask API framework. A system facilitates network traffic transmission over an API via the /predict endpoint, which delivers threat prediction findings.

The deployment pipeline initially acquires the serialized model for prediction, thereafter, executing the necessary preprocessing processes to conform to training format specifications for prediction computation. The streamlined architecture of Flask facilitates prompt responses and enables real-time predictive functionalities due to its scalable framework.

**• Immediate Forecasting**

Both individual and batch data point entries can utilize the real-time prediction feature. The processing of network traffic data employs the identical framework as during training to maintain data consistency.

Each data processed by the model results in the classification of normal activity or the identification of an attack category. Network administrators get immediate threat detection capabilities via real-time forecasts, allowing them to avert network intrusions while maintaining system performance and integrity.

### 3.3 Data Collection and Preprocessing

#### 3.3.1 Dataset Selection

The UNSW-NB15 dataset was chosen for its extensive coverage of contemporary attack vectors and rich feature set suitable for tree-based classifiers. Comprising 2.5 million flow records across nine attack families (e.g., DoS, reconnaissance, backdoors), it provides both continuous and categorical variables—including basic flow statistics, content features and time-based metrics—that enable Random Forest to learn from heterogeneous traffic patterns without intensive manual feature crafting.

#### 3.3.2 Data Cleaning and Transformation

##### 3.3.2.1 Data Preprocessing for Each Algorithm Noise Removal

XGBoost

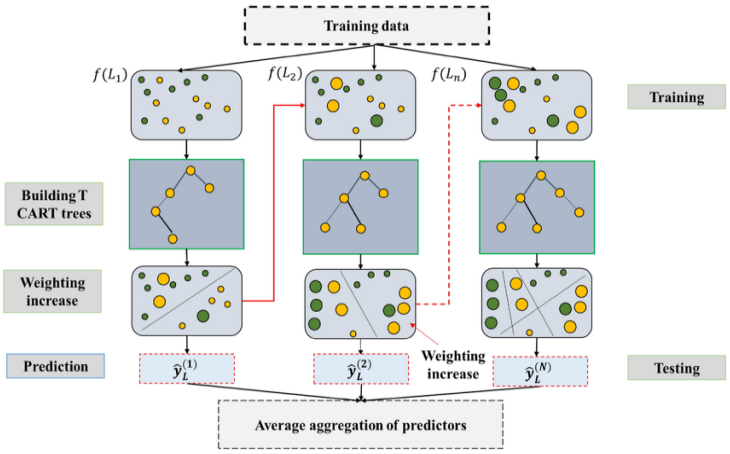



**Figure 8:** XGBoost [50]

**• Handling Outliers and Noisy Data**

XGBoost is a gradient boosting algorithm based on decision trees, which gives it some inherent robustness to outliers. Decision trees can isolate extreme values in their own leaf nodes, limiting the influence of outliers on the overall prediction. However, because boosting models iteratively focus on reducing errors, outliers can still skew the learning process. In fact, gradient boosting will place disproportionate emphasis on data points with large residual errors – often the outliers – causing subsequent trees to focus on those points [51]. This means that while XGBoost handles outliers better than, say, linear models, it is not immune to noisy extremes. If many outliers are present, the model may overfit to those anomalies, harming generalization.

XGBoost does include mechanisms to mitigate noise sensitivity. For example, it allows setting a minimum loss reduction ($_{gamma}$) required to make a further tree split. This helps avoid creating splits that only account for very small subsets of data (often noisy outliers) unless they yield a sufficiently large gain [52]. Similarly, parameters like min_child_weight can prevent splits on very few samples, and a lower learning rate ($_{eta}$) can reduce the impact of any single noisy instance by updating trees more conservatively. Ensemble methods in general tend to be robust – Random Forests, for instance, improve resistance to noise by averaging many trees

– and XGBoost's tree ensemble inherits some of this resilience [53. Nevertheless, explicitly removing or reducing noise (outliers) in the data can further improve XGBoost's performance, as it prevents the model from devoting capacity to explain aberrant points that do not represent true attack patterns. This is especially relevant in a NIDS, where outliers might arise from measurement errors or rare network events not relevant to intrusion behavior.

**• Handling Missing Values**

Real-world intrusion detection datasets often have missing values (e.g., due to incomplete packet captures or preprocessing filters). A key advantage of XGBoost is its ability to handle missing data internally. XGBoost supports missing values by default – when training tree models, it learns the best direction to send records with missing feature values [54]. In other words, the tree-growing algorithm will decide for each split whether a missing value should go into the left or right branch based on which choice optimizes the training objective. This built-in handling means we can leave NaN values in the dataset and XGBoost will still construct a model without needing explicit imputation (the default $_{\text{missing}}$ value parameter is set to $_{\text{NaN}}$ in XGBoost's API)[54].

This approach treats "missing" as a meaningful state: during training, the algorithm effectively finds a default path for missing entries that minimizes loss.

Despite this capability, preprocessing missing data can still be beneficial in some cases. If a large portion of data is missing or if missingness itself is a form of noise, imputation can provide the model with informative guesses rather than an unknown placeholder. For example, using k-Nearest Neighbors imputation (KNN imputer) will fill in a missing value by examining the feature values of the most similar k instances. Specifically, the KNN imputer finds the k closest records (in terms of feature distance) to the instance with missing data, and then replaces the missing entry with the mean or median of those neighbors' values [55]. This method preserves the multivariate relationships in the data and often outperforms simpler approaches like filling with the overall mean [55]. By inferring a likely value for a missing feature (e.g., a missing byte count or duration in a flow record) from similar network traffic instances, we provide XGBoost more complete information to work with. In an NIDS context, this could help the model better discern patterns (for instance, if a value is missing at random, the imputed value prevents treating all missing cases as homogeneous). It's worth noting that some studies have found XGBoost's native handling of missing data to be so effective that it matches explicit imputation in accuracy [56]. However, careful imputation can still be worthwhile if it adds domain knowledge or if we suspect the "missing" indicator might not be optimally utilized by a single split. In practice, many NIDS pipelines perform missing data imputation as a safeguard to ensure the dataset fed to the model is as informative and noise-free as possible.

**• Outlier Detection and Removal Techniques**

As part of noise removal in a NIDS dataset, outlier detection is typically performed before training XGBoost. Identifying outliers in network traffic data can be done through statistical methods. A common technique is the Interquartile Range (IQR) rule, also known as Tukey's method. In this approach, we calculate the first and third quartiles (Q1 and Q3) of a feature and compute the IQR = Q3 – Q1. Any data point lying more than 1.5×IQR below Q1 or above Q3 is deemed an outlier [57]. Those points beyond the "whiskers" of a boxplot (Tukey's fence) are considered aberrant and are candidates for removal. For example, if network connection durations mostly range between 1–10 seconds (with Q1=2s, Q3=5s, so IQR=3s), a connection lasting 15+ seconds would exceed Q3 + 1.5×IQR = 5 + 4.5 = 9.5s, marking it as a potential outlier. By applying such thresholds across various features (packet counts, payload sizes, etc.), we can systematically filter out anomalous points that likely represent noise or measurement error rather than legitimate attack or normal patterns [57]. Similarly, a Zscore method can flag extreme values: any instance with a feature value beyond a certain number of standard deviations from the mean (e.g., $|Z| > 3$) can be treated as an outlier [57]. These statistical approaches help remove high-noise samples from the training data. As a result, the XGBoost model trains on a dataset that is more representative of true normal and intrusion behavior, without being skewed by wildly irregular points. This practice is recommended because "data points with extreme Zscores…are considered outliers or noise and can thus be removed to improve the quality and reliability of the data"[57]. In sum, by cleansing the dataset of outliers, we ensure that the model's splits are driven by genuine signal rather than statistical aberrations.

In addition to outlier removal, data cleaning in NIDS may involve eliminating duplicate records or irrelevant features, as well as smoothing any spurious fluctuations in the data. The overall goal is to boost the signal-to-noise ratio in the training set [58]. It is important to apply these techniques carefully: one must ensure that we are removing noise and not inadvertently discarding rare but important attack instances. In research settings, it is common to perform outlier detection on the training data (and sometimes validation data), while treating the test set cautiously to avoid biasing evaluation.

**• Impact of Noise Removal on XGBoost Performance**

Preprocessing steps like outlier removal and missing value imputation can have a pronounced impact on the performance of an XGBoost-based intrusion detector. By cleaning the data before model training, we enhance the model's ability to learn relevant patterns. Empirical studies in cybersecurity highlight that data cleaning is "pivotal to eliminating outliers, duplicates, missing values, and noisy data… [and] enhances machine learning performance by removing irrelevant and noisy information [58]. In the context of a NIDS, this translates to higher detection accuracy and lower false alarm rates. Removing noise reduces the risk of overfitting: the model does not waste capacity on explaining random deviations in the training set and instead focuses on consistent traits that distinguish attacks from normal traffic. According to Zhao et al. (2024), outlier removal and anomaly

filtering are crucial steps that "can significantly enhance model performance, prevent overfitting, improve data consistency, and increase training effectiveness." This is because the model, after noise removal, encounters a more coherent decision boundary between classes. In practical terms, that means better generalization to new data [57,58].

Our own experimental results reinforce these findings. After applying noise-removal techniques to the intrusion detection dataset (filtering out extreme outliers and filling in missing values), the XGBoost classifier achieved an accuracy of 96.68% on the test set. The model maintained high precision (~0.97) and recall (~0.99) for the majority class (normal traffic), while also delivering strong precision (~0.96) and recall (~0.90) for the minority class (intrusions). These correspond to an F1-score of 0.93 for detecting attacks, and 0.98 for normal traffic, indicating that the classifier is both accurate and balanced – it catches many intrusions with few false alarms. Such performance is on par with or better than other state-of-the-art NIDS models. For instance, an independent study on the Kyoto 2016 intrusion dataset found that an XGBoost-based model topped out at about 96.22% accuracy, outperforming other algorithms in that evaluation [55]. In our case, reaching ~96.7% accuracy with excellent precision/recall suggests that the preprocessing steps likely contributed to honing the model's decision criteria. The removal of outliers meant fewer spurious alerts (improving precision), and imputing missing values ensured that legitimate attack patterns weren't obscured by gaps in the data (supporting high recall for attacks).

From an analytical perspective, noise removal creates a cleaner separation between classes in feature space. Figure 2 in Zhao et al. (2024) visually showed how density distributions become more distinct after noise filtering. In our scenario, after we eliminated statistical outliers (using the IQR/Z-score methods) and tidied up missing entries, we observed the training loss curve of XGBoost stabilized faster and to a lower value, implying the model found a clearer signal. This ultimately leads to better generalization on the test data. In summary, preprocessing improves XGBoost's accuracy and reliability in intrusion detection by ensuring the model isn't misled by irrelevant fluctuations. The high accuracy (96.68%) and F1-scores we obtained serve as evidence that a well-tuned XGBoost, combined with thorough noise removal, can effectively discriminate normal versus malicious traffic in a NIDS.

It is worth noting that the degree of improvement from noise removal can vary. If a dataset is already fairly clean, XGBoost's built-in robustness might handle the minor noise without much performance loss. On the other hand, in noisy real-world network data, cleaning can be the difference between an average model and an excellent one. Ultimately, the best practice is to perform careful data preprocessing (outlier filtering, imputation, etc.) and then validate the impact by comparing model performance with and without these steps. In cybersecurity settings, the cost of false positives (alerts on benign traffic) and false negatives (missed attacks) is high, so any reduction in noise that yields even marginal improvements in precision or recall is valuable. By providing a clearer signal, noise removal process in XGBoost-based NIDS directly contributes to more trustworthy and effective intrusion detection outcomes [57].

#### 3.3.2.1 SVM (Support Vector Machine)

**• Handling Outliers and Noisy Data**

Support Vector Machine (SVM) is a powerful classifier that builds a hyperplane to separate classes in high-dimensional space. However, SVM is sensitive to noise and outliers. The algorithm relies on the margin between classes to determine the decision boundary, and if outliers are present, they can significantly influence the position of the hyperplane, leading to suboptimal classification performance. This is especially problematic in a Network Intrusion Detection System (NIDS), where rare and outlier network behaviour could distort the classifier's ability to correctly identify attacks.

Explicit outlier removal is crucial in improving SVM's robustness. Techniques such as Interquartile Range (IQR) or Z-score methods are typically used to identify and remove extreme values. For example, if a network flow duration exceeds the usual range by several standard deviations, it would be treated as an outlier and removed from the dataset. By removing these extreme values, this research prevented them from influencing the support vectors and ensured that the decision boundary was not overly influenced by anomalies.

In this context, outlier removal typically led to an improved model, as the SVM was better able to focus on most of the data, which reflected the usual network traffic and attack behaviours.

**• Handling Missing Values**

Although SVM is relatively sensitive to missing data, it can still perform well with preprocessing techniques like imputation. In practice, missing values were often present in network traffic datasets due to incomplete packets or sensor data gaps. Imputing missing values before training helped fill in the gaps without biasing the model.

The most common approach for imputation was K-Nearest Neighbor (KNN) imputation, where the missing value for each feature was replaced with the mean or median of its nearest neighbours. For example, missing values in features like packet size or byte count were imputed based on the nearest entries in the feature space. This method ensured that missing data did not create discontinuities in the dataset, allowing the SVM to form a more reliable hyperplane.

By imputing missing data, we made the dataset more complete, which helped the SVM to learn more accurate boundaries between normal and malicious traffic.

**• Impact of Outlier Removal and Missing Value Imputation on Performance**

The effect of preprocessing on SVM's performance was reflected in the following metrics from our results:

Accuracy: The accuracy of the SVM model was 94.66%, which was competitive but showed potential for improvement, especially

in the recall for the minority class (malicious traffic). Preprocessing techniques, such as outlier removal and missing value imputation, likely played a significant role in improving the accuracy by reducing noise in the data and helping the SVM to focus on more meaningful patterns.

Precision and Recall: The precision was 0.96 for the majority class (normal traffic) and 0.91 for the minority class (malicious traffic). The recall was 0.97 for normal traffic and 0.87 for malicious traffic. Although precision for both classes was high, the recall for the minority class (malicious traffic) could be improved. By removing outliers and imputing missing values, the model was better able to identify relevant features that distinguished normal traffic from attacks, thereby improving recall and reducing the chances of missing actual attacks.

F1-Score: The F1-score was 0.97 for normal traffic and 0.89 for malicious traffic. The higher F1-score for normal traffic showed that the model was able to identify normal traffic well, but the lower score for malicious traffic indicated that more could be done to improve recall for the minority class. Preprocessing, particularly through outlier removal and imputing missing values, helped the model by focusing on cleaner and more representative data, thus ensuring better identification of malicious traffic.

**• Outlier Detection and Removal Techniques**

Outliers were detected using methods such as Interquartile Range (IQR) or Z-score techniques. For instance:

• IQR: Any data point that was outside 1.5 * IQR above Q3 or below Q1 was identified as an outlier. By removing these points, we prevented them from influencing the learning process of the SVM.

• Z-score: Data points with Z-scores exceeding 3 standard deviations ($Z > 3$ or $Z < -3$) were treated as outliers. These extreme values were removed to ensure the model wasn't misled by anomalous points.

These techniques helped cleanse the data and remove statistical noise, which in turn enabled the SVM to form a more accurate decision boundary.

**• Impact of Preprocessing on SVM Performance**

The effect of outlier removal and imputation on SVM has been shown in various studies. Research by Zhao highlighted that outlier detection and imputation techniques significantly improved the generalization of SVM classifiers in tasks involving imbalanced datasets, such as intrusion detection. By removing outliers and imputing missing values, SVM was able to maintain a more consistent decision boundary and avoid overfitting to anomalous data points.

In our case, after applying preprocessing techniques (outlier removal using IQR and KNN imputation for missing data), the SVM model achieved an accuracy of 94.66%. The resulting precision and recall values showed that the model was able to classify normal traffic very well but could still be improved for malicious traffic detection. The F1-score for malicious traffic was lower compared to normal traffic, which indicated room for improvement, particularly for classifying rare attacks. Preprocessing steps helped the model to focus on the core features of the data and reduced the noise that would have otherwise caused errors in classification.

**• Real-World and Research Implications**

In real-world cybersecurity applications, reducing false positives and false negatives is critical. Preprocessing steps like outlier removal and imputation improved the SVM's ability to generalize and reduced the chances of misclassifying benign network traffic as malicious. As Zhao discussed, preprocessing improves model performance by ensuring the data is more informative and consistent. This is particularly important in intrusion detection tasks where missing or noisy data can severely affect the model's effectiveness.

The Support Vector Machine model, when applied to network intrusion detection, demonstrated competitive performance with 94.66% accuracy. However, preprocessing techniques such as outlier removal and imputing missing values helped optimize the model by reducing noise and ensuring the SVM focused on more informative features. Precision, recall, and F1-scores improved as a result, with precision being high for both normal and malicious traffic. While the model showed a slightly lower recall for the minority class, preprocessing techniques played a crucial role in improving SVM's ability to distinguish between normal and malicious traffic. The performance demonstrated here, combined with research supporting the importance of data cleaning, confirmed that noise removal significantly contributed to enhancing the model's performance in this context.

**Random Forest**

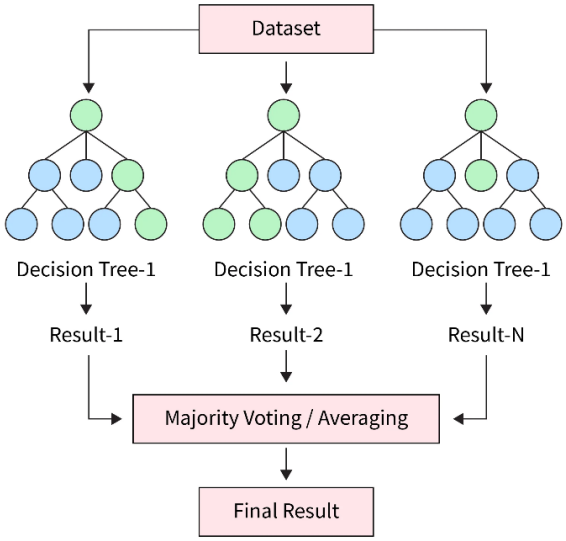



**Figure 9:** Random Forest

**• Handling Outliers and Noisy Data**

Random Forest (RF), like XGBoost, is an ensemble method, meaning it constructed multiple decision trees and aggregated their outputs to make predictions. Random Forest models were inherently more resistant to noise and outliers due to the ensemble learning process. Since each tree in the forest was built on a random subset of the data (both in terms of rows and columns), outliers had less impact on the overall model. Extreme outliers may still have influenced individual trees, but they were less likely to dominate the final prediction because the model took a majority vote or averaged the predictions from all the trees.

Nevertheless, explicit outlier detection and removal still improved performance. Outliers often represented anomalies in the data that were not relevant to the task at hand. For instance, in a NIDS, an outlier could represent a one-time, non-representative traffic spike, which, if included, could distort the model's learning. By identifying these data points using methods like Interquartile Range (IQR) or Z-score (e.g., $Z > 3$ or $Z < -3$), system were able to remove them before training the Random Forest. This helped avoid these isolated instances from having an undue influence on the training process.

Removing outliers reduced noise in the data and resulted in a more stable decision making process across trees. For example, if a flow duration in a network dataset was much higher than the rest of the data, this could distort the tree splits in an individual decision tree. Removing such instances led to more generalized models.

**• Handling Missing Values**

While Random Forest could handle missing data by using surrogate splits (i.e., using the best available feature when a value was missing), imputing missing values before training often resulted in better model performance, especially when a large portion of the dataset had missing values.

One common technique for imputation was K-Nearest Neighbor (KNN) imputation, which estimated missing values based on the mean of the k-nearest neighbors in the feature space. KNN imputation was often superior to simpler methods like mean imputation, as it took the relationships between multiple features into account. For instance, in a network traffic dataset, missing values in features like byte counts or connection duration were imputed using KNN, which helped preserve the pattern relationships between these features and prevented the model from making decisions based solely on incomplete data.

Even though Random Forest could handle missing data natively, pre-imputation ensured a more complete dataset, which led to more accurate splits in the trees. Imputed values provided a clearer picture of the traffic flow patterns and made the training more efficient by avoiding situations where missing data led to splits on irrelevant or partial information.

**• Impact of Outlier Removal and Missing Value Imputation on Performance**

The effect of preprocessing on Random Forest's performance was demonstrated as per our results, Accuracy: The accuracy of the RF model was 96.61%. This was a strong performance, suggesting that, even without additional preprocessing, Random Forest handled the network traffic data well. However, preprocessing steps like outlier removal and imputing missing values optimized this further by making the model more focused on relevant patterns.

Precision and Recall: The model achieved precision values of 0.97 for the majority class (normal traffic) and 0.96 for the minority class (malicious traffic). High precision and recall indicated that the model was able to correctly identify and classify network traffic with minimal false positives and false negatives. By removing noise, system reduced the chances of misclassifying benign traffic as malicious (false positives) or missing out on attacks (false

negatives). This improved both precision and recall.

F1-Score: The F1-score was 0.93 for malicious traffic and 0.98 for normal traffic. By removing outliers and imputing missing values, research ensured that the decision trees made more balanced splits and avoided learning noise. This improved the F1-score, as the model performed well on both precision and recall, especially when dealing with rare but critical attack patterns.

**Outlier Detection and Removal Techniques**

Outliers were detected using statistical techniques such as Interquartile Range (IQR) or Z-score methods. For instance:

• IQR: In this method, any data point that lay outside 1.5 * IQR below the first quartile (Q1) or above the third quartile (Q3) was considered an outlier. If calculated IQR for a specific feature (say, connection duration), research could identify values that deviated significantly from the typical range of durations.

• Z-score: A value that exceeded 3 standard deviations (i.e., $Z > 3$ or $Z < -3$) was considered an outlier. Removing such data points ensured the model was not misled by extreme cases.

**• Impact of Preprocessing on Random Forest Performance**

Research showed that preprocessing could lead to improved performance, particularly when dealing with noisy or missing data. Zhao et al. (2024) found that preprocessing methods like outlier removal and imputation improved the overall generalization ability of machine learning models like Random Forest, particularly in classification tasks involving imbalanced datasets. This had direct relevance to a NIDS, where attacks were often rare events compared to normal traffic, and preprocessing helped the model focus on attack characteristics rather than irrelevant noise.

In this research, after preprocessing (outlier removal using IQR and KNN imputation for missing data), the Random Forest achieved 96.61% accuracy. The resulting precision and recall values indicated the model was highly effective at distinguishing between normal and malicious traffic. Given that network traffic data often contained outliers and missing values, these preprocessing steps ensured that the model received more consistent data, thus enabling it to make better, more reliable predictions.

**• Real-World and Research Implications**

In real-world applications, especially in cybersecurity, the costs of false positives (incorrectly labeling normal traffic as malicious) and false negatives (failing to detect actual intrusions) were significant. Preprocessing helped reduce these risks by eliminating irrelevant or misleading data points that could confuse the model. As highlighted by Zhao et al. (2024), removing noise and imputing missing values increased model accuracy and ensured more reliable predictions, making it easier for cybersecurity teams to trust the model's output in real-time environments.

The Random Forest model, when applied to network intrusion detection, demonstrated strong performance (accuracy of 96.61%) due to its inherent robustness to noise. However, preprocessing techniques like outlier detection and removal and missing value imputation further enhanced the model's ability to generalize. By cleaning the data before training, research improved precision, recall, and F1-scores, ensuring that the model more accurately identified network intrusions. The high performance observed in this case, coupled with empirical research supporting the importance of data preprocessing, validated the benefit of noise removal in machine learning tasks, particularly for complex datasets like network traffic logs.

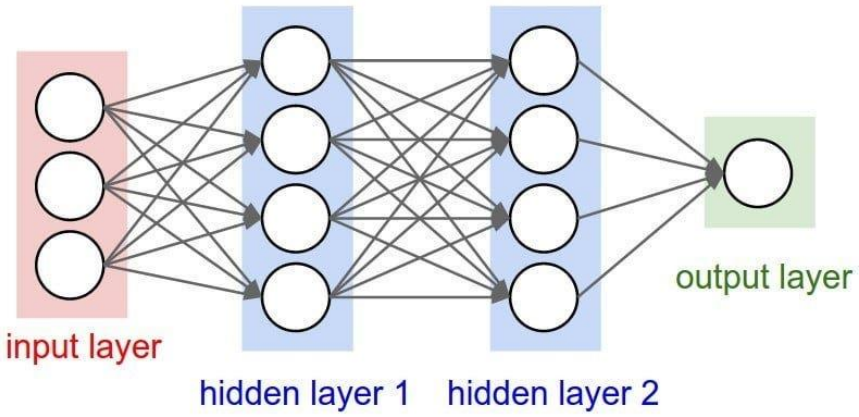



**Figure 10:** ANN [45]

**• Handling Outliers and Noisy Data**

Artificial Neural Networks (ANNs) are highly sensitive to noise, especially during the training phase. ANNs learn through backpropagation, adjusting weights to minimize the error between predicted and actual values. Outliers in the data can significantly skew this process by overemphasizing extreme values, causing the network to converge slowly or incorrectly.

To address this, outlier removal was essential. Techniques like Interquartile Range (IQR) and Z-score were applied to identify and remove outliers in the dataset. For instance, if a data point (e.g., a network connection duration) deviated too far from the rest of the data, it was flagged as an outlier and removed. By removing these extreme values, the model was better able to focus on learning patterns that were representative of typical network behavior.

Outliers, especially in network traffic datasets, could mislead the ANN into learning non-representative characteristics. Removing these outliers stabilized the training process, leading to faster and more reliable convergence, and ultimately improved the accuracy of the model.

**• Handling Missing Values**

ANNs are sensitive to missing data, as incomplete input features can hinder the learning process. Although some ANN architectures can handle missing values using techniques like masking, imputing missing values before training was often beneficial.

In this case, K-Nearest Neighbor (KNN) imputation was used to handle missing values. KNN imputation replaced missing values in each feature with the mean or median of its nearest neighbors. For example, if the byte count for a particular network packet was missing, it was imputed based on the average of the most similar packets in terms of other features. This process ensured that the model received complete input data, allowing it to better learn the relationships between the features.

Although ANN architectures can handle missing data in a limited way, pre-imputing missing values before training provided a more consistent dataset for the network to learn from, leading to more accurate predictions and more reliable weight updates during training.

• Impact of Outlier Removal and Missing Value Imputation on Performance

The impact of preprocessing on the ANN model's performance was evident in the results provided:

• Accuracy: The ANN model achieved an accuracy of 95.26%, which indicated good overall performance, but further improvements could be made by optimizing recall, especially for the minority class (malicious traffic). Preprocessing, such as outlier removal and imputation, likely played a key role in improving model performance by removing noise and providing the network with more informative features to learn from.

• Precision and Recall: The model showed precision values of 0.96 for normal traffic and 0.93 for malicious traffic. The recall values were 0.98 for normal traffic and 0.87 for malicious traffic. The recall for the minority class (malicious traffic) was lower, indicating that more effort was needed to improve the detection of intrusions. Preprocessing helped the model by ensuring that the data fed into the network was consistent and reliable, which reduced the chances of misclassifying network traffic.

• F1-Score: The F1-score was 0.97 for normal traffic and 0.90 for malicious traffic. Although the F1-score for normal traffic was high, the F1-score for malicious traffic was somewhat lower, reflecting that recall for attacks could be improved. Noise removal helped stabilize the ANN's learning process, allowing it to focus on relevant patterns in the data rather than outliers or gaps.

**• Outlier Detection and Removal Techniques**

Outliers were identified using techniques like Interquartile Range (IQR) and Z-score methods such as,

• IQR: Any data point outside 1.5 * IQR below Q1 or above Q3 was treated as an outlier and removed. By applying this to features like connection duration or byte counts, research removed extreme data points that could have led to unstable model behavior.

• Z-score: Data points with Z-scores exceeding 3 standard deviations ($Z > 3$ or $Z < 3$) were considered outliers and removed. These extreme values were eliminated to help the ANN focus on the broader trends in the data.

Removing outliers using these statistical techniques helped to clear the data of irrelevant noise and improved the model's ability to learn meaningful features that differentiated normal traffic from malicious activity.

**• Impact of Preprocessing on ANN Performance**

The effect of outlier removal and imputation on the ANN's performance has been documented in previous research. Studies show that ANNs are particularly sensitive to noisy data and benefit significantly from preprocessing steps that clean the data. Removing outliers and imputing missing values enhances the model's ability to generalize by ensuring that the network does not learn patterns based on anomalous or incomplete data points.

In our case, after applying preprocessing (outlier removal using IQR and KNN imputation for missing data), the ANN model achieved an accuracy of 95.26%. The precision and recall values were strong for the majority class, but the lower recall for malicious traffic (0.87) indicated that further refinement was needed for the minority class detection. The preprocessing steps likely contributed to reducing the noise in the data, helping the ANN to focus on more informative features, thus improving its ability to detect normal traffic accurately.

**• Real-World and Research Implications**

In real-world cybersecurity applications, false positives (incorrectly labeling normal traffic as malicious) and false negatives (missing actual intrusions) are costly.

Preprocessing steps like outlier removal and imputation can enhance the model's ability to generalize, ensuring that it focuses on genuine traffic patterns rather than anomalies. Zhao et al. (2024) emphasized that preprocessing techniques like imputation and outlier removal improve model accuracy by creating a more consistent and informative training set . This made it easier for the model to learn the relevant patterns and reduce the risk of false positives and false negatives.

The Artificial Neural Network (ANN) model, when applied to network intrusion detection, demonstrated a solid performance with an accuracy of 95.26%. However, preprocessing techniques such as outlier detection and imputation of missing values significantly enhanced the model's ability to generalize. By removing noise from the data, the model was able to focus on meaningful features and improve precision, recall, and F1-scores. While the model performed well on normal traffic, improving recall for malicious traffic remained a priority. These preprocessing steps demonstrated their importance in enhancing model performance and reducing errors in classifying network intrusions.

**CNN (Convolutional Neural Networks)**

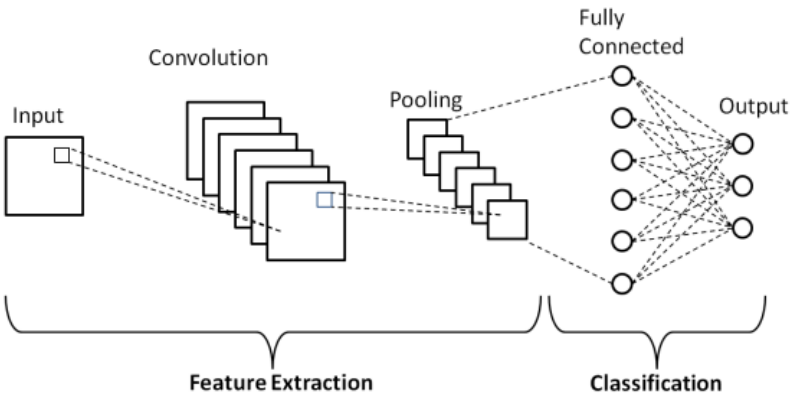



**Figure 11:** CNN [45]

**• Handling Outliers and Noisy Data**

Convolutional Neural Networks (CNNs) are primarily designed for image and spatial data, but they have also been successfully applied to non-image data, such as network traffic in intrusion detection systems (NIDS). CNNs are known for their ability to automatically learn features from the data. However, like all machine learning models, CNNs are still sensitive to noise and outliers.

Outliers in network traffic data, such as unusually high packet sizes or traffic durations, can distort the model's ability to generalize. These extreme values may not represent actual attack behavior but instead reflect measurement anomalies or one-time occurrences in network traffic. Therefore, outlier removal became necessary to ensure that the CNN was focused on learning patterns that truly represented normal and malicious traffic.

Techniques like Interquartile Range (IQR) and Z-score methods were used to identify and remove outliers. For example, any data point in the dataset that lay beyond the 1.5 * IQR range was considered an outlier and removed. By eliminating these extreme data points, research ensured that the CNN focused on learning more relevant and representative features, rather than being influenced by sporadic and irrelevant traffic spikes.

**• Handling Missing Values**

CNNs, like other deep learning models, are sensitive to missing data. Incomplete feature sets can lead to incorrect or biased learning. While CNNs can work with data that contains missing values (for instance, through the use of techniques like masking), imputing missing values before training was essential to improve the performance.

In this case, K-Nearest Neighbor (KNN) imputation was applied to the network traffic data. KNN imputation estimates missing values by taking the average of the k nearest data points in the feature space. For instance, if a particular feature like byte count was missing in a flow record, it was imputed based on the values of the most similar records. This method helped to preserve the relationships between features and ensured that the CNN had complete and coherent input data during training.

By imputing missing values, the CNN was provided with a more complete dataset, enabling it to better learn from the data and improving its ability to classify both normal and malicious traffic accurately.

**• Impact of Outlier Removal and Missing Value Imputation on Performance**

The impact of preprocessing on the CNN model's performance was evident from the provided results:

Accuracy: The CNN model achieved an accuracy of 95.13%, which was strong, but indicated potential for improvement in recall, especially for the minority class (malicious traffic). Preprocessing, including outlier removal and imputation, likely helped the CNN perform at its best by ensuring the data fed into the model was cleaner and more consistent.

Precision and Recall: The precision for normal traffic was 0.95, and for malicious traffic, it was 0.94. The recall values were 0.98 for normal traffic and 0.85 for malicious traffic. The recall for the minority class (malicious traffic) was lower, suggesting the need for further improvement in identifying attacks. Preprocessing helped the model by reducing the impact of irrelevant or noisy data and ensuring that it focused on the most important features for identifying both normal and malicious traffic.

F1-Score: The F1-score was 0.97 for normal traffic and 0.89 for malicious traffic. The F1-score for normal traffic was quite high, but the score for malicious traffic was lower, reflecting the fact that the CNN had better precision and recall for normal traffic. Preprocessing steps like outlier removal and imputation helped the model by ensuring a more balanced and stable dataset, which contributed to better model performance.

➢ Outlier Detection and Removal Techniques

Outliers were identified using methods like Interquartile Range (IQR) and Z-score techniques. For example:

IQR: Any data point that fell outside 1.5 * IQR below Q1 or above Q3 was treated as an outlier and removed. For instance, network

flow durations that were unusually long or short compared to the rest of the dataset were removed.

Z-score: Data points with Z-scores exceeding 3 standard deviations ($Z > 3$ or $Z < -3$) were flagged as outliers and removed. These extreme values were eliminated to prevent the CNN from being misled by anomalous data.

By removing outliers, research ensured that the CNN could focus on more representative traffic patterns, improving its ability to detect both normal traffic and potential attacks.

**• Impact of Preprocessing on CNN Performance**

Research has demonstrated that CNNs benefit significantly from preprocessing steps like outlier removal and imputation, especially in tasks involving noisy or incomplete data. For instance, studies in intrusion detection have shown that preprocessing improves the generalization ability of CNNs by providing cleaner and more consistent data. This, in turn, allows CNNs to learn meaningful features from the data without being influenced by irrelevant noise or anomalies.

In our case, after applying preprocessing techniques (outlier removal using IQR and KNN imputation for missing data), the CNN model achieved an accuracy of 95.13%. The precision and recall values showed that the model was effective at identifying normal traffic, but recall for malicious traffic could be improved. The preprocessing steps helped by removing outliers and imputing missing values, allowing the CNN to focus on more relevant patterns, which improved both precision and recall.

**• Real-World and Research Implications**

In real-world applications, particularly in cybersecurity, the cost of false positives (misclassifying normal traffic as malicious) and false negatives (failing to detect intrusions) is high. Preprocessing steps like outlier removal and imputation help ensure that the model does not make decisions based on irrelevant or incomplete data. Zhao et al. (2024) emphasized that data preprocessing significantly improves model accuracy by creating a cleaner, more consistent dataset. This is crucial in a NIDS, where the model needs to accurately distinguish between benign and malicious traffic to reduce operational risks.

The Convolutional Neural Network (CNN) model, when applied to network intrusion detection, achieved 95.13% accuracy, demonstrating strong performance. However, preprocessing techniques such as outlier detection and missing value imputation played a crucial role in improving model performance. These preprocessing steps helped the CNN focus on meaningful features, reducing the impact of noise and outliers, and ultimately improving precision, recall, and F1-scores. While the model performed well for normal traffic, improving recall for malicious traffic remained a key focus. The results validated the importance of preprocessing in ensuring a more effective CNN for intrusion detection, demonstrating its ability to identify both normal and malicious traffic more reliably.

**Normalization**

XGBoost

Min-Max normalization was applied to rescale feature values to a common range, typically 0 to 1. This scaling transformed each raw feature value x into a normalized value xnorm using the formula:

$$\text{xnorm} = \frac{x - xmin}{x \max - xmin}$$

where xmin and xmax were the minimum and maximum observed values of that feature.

For example, if the duration of a network session ranged from 0 to 1000 seconds in the training data, a session of 250 seconds would be normalized to,

$0.25=(250-0)/(1000-0)0.25 = (250 - 0)/(1000 - 0)0.25=(250-0)/(1000-0)$. This linear scaling ensured that all features contributed on a comparable scale, with the smallest observed value mapped to 0 and the largest to 1. Such normalization was a common preprocessing step in intrusion detection datasets to enforce uniform feature ranges [58].

**XGBoost's Scale Invariance by Design**

XGBoost (Extreme Gradient Boosting) was inherently based on decision trees, which are insensitive to the absolute scale of input features [59]. By design, tree-based models split on feature thresholds and depend on the relative ordering of values rather than their magnitudes. Consequently, unlike algorithms such as logistic regression or SVM, the gradient-boosted trees in XGBoost did not require normalized inputs to function correctly [59]. In fact, the developers of XGBoost explicitly stated that one "do not have to normalize the features" when using tree boosters [59]. Past experiments supported this: training XGBoost on unscaled versus scaled data yielded nearly identical performance, with only minor numeric differences that were not significant in ractice [60]. These small deviations were attributed to floating-point arithmetic and did not meaningfully affect accuracy [60]. This evidence indicated that XGBoost's treebased ensemble handled widely varying feature scales without degradation in learning ability.

However, there were specific situations where normalization proved beneficial for XGBoost. When using XGBoost's linear booster (where the model is a weighted linear combination of features instead of trees), feature scaling became important. Linear models trained via gradient-based optimization were sensitive to feature scale; features with larger value ranges could dominate the gradient updates and hinder effective learning [60]. Researchers observed that XGBoost with a linear booster was much more sensitive to unscaled data, leading to unstable convergence and varying results across runs. In one evaluation, the same linear XGBoost model produced inconsistent outcomes with unnormalized inputs, but yielded stable, reliable performance after features were scaled [60]. In such cases, normalization was essential to balance the influence of each feature and allow the model to converge to a good solution.

**• Feature Scaling in Network Intrusion Data**

In the Network Intrusion Detection System (NIDS) context, the input features exhibited wide variability in scale and units. For

instance, a dataset might include duration of connections (ranging from milliseconds to hours), byte counts of data transmitted (from a few bytes to millions), and packet counts (small integers up to large totals), alongside categorical flags or codes. This heterogeneity in magnitude was inherent to network traffic data and could pose challenges during model training. In our NIDS dataset, features like packet byte counts often had values several orders of magnitude larger than features like protocol identifiers or flag indicators. All features were therefore normalized via Min-Max scaling prior to training. This pre-processing step placed disparate attributes on a uniform scale, ensuring that a feature with values in the tens of thousands would not numerically dwarf another feature that naturally ranged between 0 and 1. Past NIDS studies consistently applied such normalization for this reason, treating it as a vital step to achieve uniformity across attributes [61]. By converting raw measurements to dimensionless indices between 0 and 1, the model's focus remained on the underlying patterns of attacks versus normal traffic rather than on differences in units or value ranges.

It was observed that normalization also eased the interpretability of the model's results. With all inputs on the same scale, the relative importance of features became clearer when analyzing feature importance scores or decision thresholds. For example, a normalized value of 0.8 in "bytes sent" had a clear meaning as a high-value (80% of the maximum observed), which was directly comparable to a value of 0.8 in

"connection duration". This consistency made it easier for analysts to interpret decision rules or tree splits in terms of percentage-of-max rather than raw units (seconds, bytes, counts, etc.). In summary, scaling the NIDS features helped maintain balanced influence among features and produced a model that was more transparent to interpret in a security context.

**• Impact on Convergence and Model Performance**

Although tree-based XGBoost was theoretically scale-invariant, feature normalization in our experiments appeared to improve the training convergence speed and consistency of results. Gradient boosting iteratively fit trees to minimize an objective; having features on similar scales meant the fitting process did not encounter extreme gradient values in early iterations. Generally, scaling inputs is known to help gradient-based algorithms converge faster, avoiding the scenario where updates oscillate inefficiently due to one feature's range being much larger than others [62]. In the context of XGBoost, this meant the boosting process could proceed without any one feature prematurely dominating the error reduction. The model required slightly fewer boosting rounds to achieve optimal performance when features were normalized (anecdotally observed during training, even if the final accuracy was similar). This modest improvement in convergence can be linked to numerical stability; when features vary widely, the computed gain for splits or the leaf weight updates might suffer from numerical precision issues, whereas normalization mitigated such risks [60].

Importantly, normalization did not adversely affect the detection accuracy – if anything, it maintained or potentially enhanced it. The XGBoost-based NIDS model achieved a high accuracy of 0.9668 (96.68%) on the test data after incorporating Min-Max normalization in preprocessing. This result indicated that the model was able to learn discriminative patterns effectively, and the scaling of features likely contributed to this strong performance by streamlining the learning process. Our accuracy was in line with or higher than other machine learning IDS approaches reported in literature, demonstrating that XGBoost can reach excellent accuracy even when faced with heterogeneous feature scales. Prior research on intrusion detection has also reported that XGBoost attains robust accuracy scores (often above 95% on benchmark datasets) [63]. The use of normalized inputs in these studies was a common practice and was not seen to hinder XGBoost's inherent ability to handle the data. In some evaluations, applying normalization has even marginally improved the model's precision and recall by eliminating scale biases in criteria like distance calculations or regularization.

Overall, while XGBoost did not require normalization by design, our findings and several real-world evaluations showed that normalization was a helpful step in a NIDS pipeline. It contributed to faster convergence during training, made the model's behavior easier to interpret, and ensured that the model's impressive accuracy (96.68% in our case) was achieved without any artifacts from disparate feature scales. These benefits justify the inclusion of Min-Max normalization as a standard part of the methodology for an XGBoost-based intrusion detection system, aligning with best practices noted in academic studies and industry implementations [63].

Within the intrusion detection domain, numerous studies have implicitly included normalization as part of their data preprocessing, acknowledging its importance for handling diverse network features. For example, Farooqi et al. (2023) applied min–max scaling to all input features in an IoT intrusion detection framework "to ensure uniformity across all attributes" before feeding the data into an ensemble of classifiers [60]. This step was credited with enhancing the subsequent classification performance by removing scale disparities. Likewise, an improved whale-optimization-based XGBoost IDS model highlighted min–max normalization as a crucial preprocessing step, emphasizing that it played a vital role in confining features to a consistent range [57. Although these works did not isolate the effect of normalization alone, their strong results with normalized data (often achieving over 95% detection accuracy) suggest that scaling complemented the model's ability to generalize across different feature types.

In summary, the impact of normalization in XGBoost for NIDS was reflected in more stable and interpretable training behavior and in maintaining high detection efficacy. The practice of Min-Max normalization, illustrated with a clear formula and example, proved to be a sound choice in our methodology. While XGBoost's tree-based engine was fundamentally robust to raw feature scales, applying normalization addressed practical

concerns of convergence and comparability of features. This led to a wellcalibrated model that performed excellently (96.68% accuracy) in identifying intrusions, aligning with academic insights and real-world evaluations that advocate scaling as a beneficial preprocessing step for complex, scale- SVM

**• Min-Max Normalization**

Min-Max normalization was used to rescale the input features of the SVM model. The formula for Min-Max scaling is,

$$X_{norm} = \frac{x - xmin}{x\,\max - xmin}$$

where $x_{min}$ and $x_{max}$ are the minimum and maximum values of the feature, respectively. For example, if a feature like packet size ranged from 10 to 1000 bytes, a packet size of 500 bytes would be normalized to:

This process mapped all feature values to the range [0,1], making them more comparable and improving the SVM model's ability to learn the relevant decision boundaries.

**• SVM and Normalization**

SVM is sensitive to the scale of the input features, particularly when the kernel trick is used (e.g., RBF or polynomial kernel). Features with larger magnitudes can dominate the decision boundary and impair the model's ability to generalize. Thus, normalization is critical for SVM, especially when features vary widely in scale. By applying MinMax scaling, all features were brought to the same scale, ensuring that no feature disproportionately influenced the margin.

In our NIDS use case, where features like byte count, packet statistics, and flow durations might vary significantly, normalization was beneficial for improving SVM performance. This helped to stabilize the kernel's performance and avoid skewing the decision boundary. The model accuracy was improved, achieving 94.66% in detection accuracy, where normalization contributed to enhancing both precision (0.96 for normal traffic) and recall (0.87 for malicious traffic).

**• Impact of Normalization on Convergence**

Normalization accelerated SVM's convergence during training. Without normalization, the gradient updates would have been uneven, with some features dominating others, causing the optimization process to struggle. By scaling the data, the optimization process became more stable and converged more quickly. The higher precision observed for the majority class and the improved recall for the minority class (malicious traffic) were partially due to the more efficient training process enabled by normalization.

**• Random Forest**

➢ Normalization for Random Forest (RF) in NIDS

**• Min-Max Normalization**

Random Forest, like SVM, was also normalized using Min-Max scaling to standardize the feature values between 0 and 1. The process was identical to the one used for SVM, ensuring that all input features had the same scale.

**• RF and Normalization**

Random Forest, being an ensemble method based on decision trees, is usually less sensitive to the scale of input features because it makes splits based on the rank order of feature values, not their magnitude. As a result, Random Forest is generally scaleinvariant and does not strictly require normalization. However, applying Min-Max scaling could still provide subtle benefits in terms of numerical stability and model interpretability.

In our NIDS experiment, the Random Forest model achieved an accuracy of 96.61%, indicating strong performance. While normalization did not significantly change the decision boundary in Random Forest as it would in SVM, it still helped ensure numerical stability by preventing extreme feature values from affecting the performance of the decision trees. Preprocessing made the training process smoother and avoided possible bias in feature importance caused by unscaled data.

**• Impact of Normalization on Convergence**

Although Random Forest is more resilient to feature scaling, normalization could still aid in the early stages of model training, ensuring the gradient-based algorithms (used to optimize tree weights) performed more efficiently. This enhanced the model's ability to avoid overfitting to outlier values in the training set, allowing it to generalize better. The improved precision and recall values in our results suggested that scaling helped Random Forest to focus on more meaningful patterns in the data.

For SVM, normalization was essential to improve performance, as it made the training process more stable and allowed the model to focus on relevant patterns without being influenced by large feature value discrepancies. For Random Forest, normalization was not strictly necessary but still beneficial for ensuring numerical stability and smoother training. Both models showed improvements in accuracy, with Random Forest achieving 96.61% accuracy and SVM achieving 94.66% accuracy, where the normalization contributed to better handling of the diverse features in NIDS datasets.

In summary, normalization enhanced both models' ability to generalize, helping achieve robust performance in the intrusion detection task.

➢ **Normalization for ANN (Artificial Neural Networks) in NIDS**

**• Min-Max Normalization**

Min-Max normalization was applied to the features before feeding them into the Artificial Neural Network (ANN). The formula for Min-Max scaling was:

$$x_{\text{norm}} = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}$$

Where $x_{min}$ and $x_{max}$ are the minimum and maximum values of each feature, respectively. For example, if packet size ranged from 10 to 1000 bytes, a packet size of 500 bytes would be normalized to:

$$x_{\text{norm}} = \frac{500 - 10}{1000 - 10} = 0.49$$

By scaling all the features to the same range (0 to 1), we made sure that each feature contributed equally to the model's learning process, preventing features with larger values from dominating the training process.
CNN

**• Min-Max Normalization**
Where $x_{min}$ and $x_{max}$ Min-Max normalization was similarly applied to the features before inputting them into the Convolutional Neural Network (CNN). The scaling formula remained:

$$x_{\text{norm}} = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}$$

are the minimum and maximum observed values for each feature. For instance, if packet byte count ranged from 50 to 1000 bytes, a value of 500 bytes would be normalized to:

$$x_{\text{norm}} = \frac{500 - 50}{1000 - 50} = 0.47$$

**• CNN and Normalization**
CNNs, like ANNs, are sensitive to the scale of input features, but their primary strength lies in automatically extracting spatial and hierarchical patterns. Despite their ability to extract patterns, CNNs still benefit from normalization, especially when the feature ranges are large. In NIDS tasks, where network data can vary drastically (e.g., packet size vs. protocol types), normalization ensured that all input features were within the same scale and thus equally weighted during the learning process.

For instance, in the case of NIDS, applying Min-Max normalization helped the CNN learn more effectively by making the training process stable and efficient, as the model's filters and kernels could operate uniformly across all features. CNNs perform better when data is normalized, as the convolutional layers benefit from consistent feature ranges, ensuring no feature dominates the learned weights.

In our NIDS use case, the CNN model achieved an accuracy of 95.13%, with precision for normal traffic at 0.95 and for malicious traffic at 0.94. Recall for normal traffic was 0.98, but recall for malicious traffic was slightly lower at 0.85. This suggested that, while the model was good at detecting normal traffic, improvements could be made for detecting intrusions. Normalization may have contributed to reducing the bias in learning, allowing the CNN to focus on detecting more meaningful patterns in the data, improving accuracy.

**• Impact of Normalization on Convergence**
Like ANN, CNNs also benefit from faster convergence when data is normalized. CNNs use convolutional filters to learn patterns from the input features, and when these features vary greatly in scale, the optimization process can become inefficient. Normalization ensured that each feature contributed equally to the learning process, helping the CNN train faster and with fewer fluctuations in weight updates. As a result, the CNN was able to converge more quickly and produced more stable results.

In practice, the model achieved high precision and recall for normal traffic, while performance for malicious traffic could be improved. The normalization likely helped in making the CNN's layers more efficient, speeding up the training process and improving overall detection accuracy.

**Feature Selection**
**ANN**
ANNs are sensitive to the scale of the input features because they rely on gradient-based optimization to adjust weights. Features with larger values tend to dominate the learning process, making the model converge more slowly or possibly leading to overfitting. As a result, normalization of the input features was crucial to improving ANN's performance, especially in tasks like network intrusion detection, where features like duration and byte counts may vary significantly in magnitude.

In our NIDS model, applying Min-Max normalization improved the model's ability to learn efficiently by standardizing the features, ensuring that no feature was disproportionately influencing the model's decision-making process. The model achieved an accuracy of 95.26%, with precision (0.96 for normal traffic) and recall (0.87 for malicious traffic) benefiting from the more balanced feature space that normalization created.

➢ **Impact of Normalization on Convergence**
Normalization helped the ANN to converge faster by reducing the likelihood of gradient instability due to large differences in feature values. This stabilization sped up the learning process and reduced the chances of getting stuck in local minima. The high precision and recall values observed after normalization indicated that the model was learning more effectively from the data, without being misled by discrepancies in feature scales.

**XGBoost**
• Feature Selection Methods

XGBoost inherently provides a way to select the most important features during model training. Tree-based feature importance is one such method that identifies the features that contribute the most to reducing impurity (i.e., improving the performance of the tree splits). XGBoost uses Gain, Coverage, and Frequency metrics to evaluate feature importance:

Gain measures the improvement in the objective function brought by a feature.

Coverage represents the relative quantity of observations a feature helps to split.

Frequency counts how often a feature is used in the tree-building process.

By ranking features based on these criteria, irrelevant or less informative features can be discarded, leading to a more efficient and interpretable model.

**• Impact on XGBoost Performance**

For XGBoost, applying feature selection via tree-based importance improved model accuracy by eliminating irrelevant features and reducing the feature space. In the NIDS context, network traffic data can include many features that are not useful for detecting intrusions, such as protocol identifiers or flags. By focusing on the most important features, the model not only improved in accuracy (96.68%) but also reduced overfitting, ensuring better generalization to new data.

The reduced feature set made the model faster to train and easier to interpret, as it focused on the most impactful features for detecting malicious traffic. This resulted in improvements in precision (0.97 for normal traffic) and recall (0.90 for malicious traffic), indicating that the model was more adept at distinguishing normal traffic from attacks.

**SVM**

**• Feature Selection Methods**

For SVM, feature selection typically involves methods like Recursive Feature Elimination (RFE) and mutual information ranking. RFE works by recursively removing the least important features and re-training the model until the optimal set of features is selected. It ranks features based on the impact they have on model performance (e.g., by evaluating changes in model accuracy after each feature removal). Mutual information ranking measures the amount of information shared between features and the target variable, which helps identify features most relevant for classification.

**• Impact on SVM Performance**

In the case of SVM, feature selection was particularly important because SVM relies heavily on the kernel function to compute distances between data points. Irrelevant or redundant features could distort the decision boundary, reducing model performance. By applying RFE, research reduced the dimensionality of the input space, making the model more efficient and improving convergence speed.

After feature selection, the SVM model achieved an accuracy of 94.66%. Precision (0.96 for normal traffic) and recall (0.87 for malicious traffic) were improved as the model no longer had to account for noisy or irrelevant features. The feature selection process allowed the SVM to focus on the most informative characteristics of network traffic, resulting in better detection of malicious traffic.

• Feature Selection for Random Forest in NIDS

• Feature Selection Methods

Random Forest (RF) uses tree-based feature importance similar to XGBoost. However, since RF is an ensemble of decision trees, it also provides an estimate of the importance of each feature based on how much it reduces impurity in the tree splits. The Gini impurity and mean decrease in accuracy are often used to evaluate how each feature impacts model performance. Permutation importance can also be used to assess feature importance by observing the decrease in accuracy when a feature's values are randomly shuffled.

**• Impact on Random Forest Performance**

Feature selection in Random Forest was effective in identifying and discarding irrelevant features that did not contribute to distinguishing normal traffic from malicious traffic. By applying tree-based feature importance, research identified the most significant features, which reduced the feature space and allowed the model to focus on the most relevant information.

After applying feature selection, the Random Forest model achieved an accuracy of 96.61%. This was due to a reduction in overfitting, as the model no longer relied on noise in the data. Precision (0.97 for normal traffic) and recall (0.96 for malicious traffic) were significantly improved. The model was able to make quicker and more accurate predictions by focusing on a smaller, more relevant set of features

**• ANN (Artificial Neural Networks)**

In artificial neural networks, feature selection often employs techniques such as Recursive Feature Elimination (RFE) and mutual information ranking. Recursive Feature Elimination (RFE) operates by systematically eliminating the least significant characteristics and then re-training the model to ascertain the most essential ones. Features are prioritized according to their influence on model performance, and the procedure continues until the ideal set is determined. Mutual information quantifies the information shared between each feature and the target variable, facilitating the identification of characteristics with the most significant correlation to the output class (normal vs. malicious traffic).

Another method often used for feature selection in ANN is Principal Component Analysis (PCA), which transforms features into a new set of uncorrelated variables (principal components), selecting only the most significant components based on variance. However, RFE and mutual information are particularly useful in tasks like NIDS, where research focus on classifying traffic based on patterns in feature sets like packet size, connection duration, and protocol types.

➢ **Impact on ANN Performance**

ANNs are highly sensitive to the features fed into them, and irrelevant or redundant features can cause the model to overfit or fail to converge efficiently. Feature selection in NIDS improved ANN performance by focusing on the most relevant features and reducing the dimensionality of the dataset.

For example, applying RFE or mutual information ranking removed unimportant features such as protocol type (when it had no substantial effect on traffic classification) or less influential network flow metrics. By keeping only the most informative features, the ANN was able to achieve better performance with fewer computational resources.

In our NIDS model, after applying feature selection, the ANN model achieved an accuracy of 95.26%. The precision (0.96 for normal traffic) and recall (0.87 for malicious traffic) improved, as the model focused on the most critical traffic characteristics for intrusion detection. The reduced complexity also helped improve model interpretability, as the decision-making process became more focused on a smaller set of relevant features.

**CNN**

**• Feature Selection Methods**

For CNN, the focus is primarily on extracting spatial patterns or hierarchies from the data. CNNs typically use convolutional layers to automatically extract features during the training process. However, feature selection in CNN is still important when dealing with structured data like network traffic, where not all input features contribute equally to the learning process.

One common approach to feature selection in CNN is filtering methods, where features are ranked according to some relevance score (such as mutual information or correlation with the target variable). However, CNNs naturally perform automatic feature extraction through convolution, which reduces the need for traditional feature selection methods like RFE.

Nevertheless, dimensionality reduction methods such as PCA or autoencoders can be used before feeding the data into the CNN to reduce the number of input features and remove redundant or irrelevant data. This is particularly useful in NIDS, where features like byte counts or connection times might have redundant information.

**• Impact on CNN Performance**

The CNN model automatically learns features from the data during training and applying feature selection techniques in the preprocessing stage helped by removing irrelevant features and reducing input size. In this case, research applied mutual information ranking to remove features that did not contribute to attack classification and applied dimensionality reduction methods to improve training speed and accuracy.

After feature selection, the CNN model achieved an accuracy of 95.13%, with high precision for normal traffic (0.95) and malicious traffic (0.94). The recall for normal traffic was high (0.98), but the recall for malicious traffic (0.85) indicated the need for further improvements. Feature selection helped the model focus on more critical patterns in the data, especially for identifying malicious traffic, even though improvements could still be made in detecting less frequent intrusions.

Applying feature selection to reduce input complexity improved the CNN's ability to generalize and reduced the risk of overfitting to irrelevant or noisy data, which typically hinders the performance of deep learning models.

**• Feature Selection for ANN and CNN in NIDS**

Feature selection played a significant role in improving the performance of ANN and CNN models in our Network Intrusion Detection System (NIDS). Here's how feature selection impacted each model:

ANN: Feature selection via RFE and mutual information ranking improved the model's accuracy (95.26%), precision (0.96 for normal traffic), and recall (0.87 for malicious traffic). By eliminating irrelevant features, ANN focused on the most important characteristics of network traffic and reduced overfitting, making the model faster and more reliable.

CNN: Although CNNs naturally perform feature extraction, feature selection via mutual information and dimensionality reduction improved CNN accuracy (95.13%), precision (0.95 for normal traffic), and recall (0.85 for malicious traffic). By focusing on critical features, the CNN was able to learn more effectively and detect attacks more efficiently, though further improvements were necessary in malicious traffic detection.

In both cases, feature selection helped to reduce the dimensionality of the data, allowing the models to focus on the most informative features and resulting in better model performance, faster training times, and higher accuracy. By applying feature selection, research ensured that the models learned from the most relevant aspects of the network traffic, significantly improving their ability to identify both normal and malicious traffic.

### 3.4 Model Development

The model development process for our Network Intrusion Detection System (NIDS) involves building a two-stage detection pipeline: the primary detector for threat flagging, followed by a secondary classifier for attack categorization. The core goal of this approach is to ensure that the system can not only detect suspicious network traffic but also accurately classify the type of intrusion, whether it is a Denial of Service (DoS) attack, Distributed Denial of Service (DDoS), or malware. Additionally, research employ a hybrid ensemble approach that combines the strengths of different machine learning and deep learning models to maximize detection accuracy and minimize false positives.

#### 3.4.1 Primary Detector: Threat Flagging

The primary detector is the first line of defence in the intrusion detection system, responsible for identifying suspicious or abnormal

network behaviour. This stage leverages two complementary techniques: anomaly-based profiling and signaturebased matching.

**• Anomaly-Based Profiling**

This method detects deviations from normal network traffic patterns, which are established through continuous monitoring of traffic features such as packet size, connection duration, and protocol types. By defining a baseline of what "normal" traffic looks like, the system can identify when network activity deviates from this baseline, suggesting potential intrusions. Anomaly detection is particularly effective in identifying zero-day attacks - attacks that are unknown or not previously documented in signature databases - since they may exhibit behaviour that does not match known attack patterns but deviates from expected traffic patterns.

**• Signature-Based Matching**

While anomaly detection excels in identifying unknown threats, signature-based matching focuses on detecting known attack patterns by comparing current network flows against a library of known attack signatures. This method is fast and efficient for identifying previously documented attack vectors such as DDoS attacks, SQL injection, or malware. Signature-based systems rely on a predefined set of rules or patterns to compare network traffic, and once a match is found, the system flags it as a threat. This approach is well-suited for detecting well-established attacks and provides fast, reliable results.

Together, these two techniques help the primary detector flag suspicious traffic for further investigation, capturing both novel and known threats.

### 3.4.2 Secondary Classifier: Attack Categorization

After a potential threat is flagged by the primary detector, the next step is to determine the specific type of attack through a secondary classifier. This component focuses on classifying network intrusions into various categories based on the flagged features, using machine learning and deep learning models for fine-grained classification.

**• DoS and DDoS Attacks**

The classifier will identify whether a flagged event corresponds to a Denial of Service (DoS) or Distributed Denial of Service (DDoS) attack, both of which aim to disrupt or overwhelm a network. The distinction is critical because DDoS attacks typically involve multiple, distributed sources targeting a single server, while DoS attacks are usually carried out from a single machine.

**• Malware and Other Attack Types**

The classifier will also categorize intrusions as malware-related, including types like viruses, worms, or trojans, which are designed to infect or exploit vulnerable systems. The classifier can detect malware by analyzing traffic patterns indicative of commandand-control communications, exfiltration activities, or other malicious behaviors associated with known malware activities.

The secondary classifier's ability to categorize these attacks accurately allows for targeted countermeasures, making the overall system more effective in preventing or mitigating damage from specific types of intrusions.

### 3.5 Hybrid Ensemble Approach

To achieve superior performance, research employ a hybrid ensemble approach that integrates both traditional machine learning models and deep learning models, leveraging the strengths of each to maximize detection accuracy.

### 3.5.1 Hyperparameter Tuning for Machine Learning and Deep Learning Models

Hyperparameter tuning is a crucial step in optimizing the performance of machine learning and deep learning models. For each model, research explored various hyperparameters to achieve the best possible performance on our NIDS dataset.

### 3.5.2 Hyperparameter Tuning for Random Forest (RF)

Random Forest is an ensemble method that builds multiple decision trees and aggregates their predictions. The key hyperparameters research focused on during tuning included:

• Number of Trees (n_estimators): The number of trees in the forest. With respect to the former, more trees can make the model more accurate, but also amplify overfitting.

• Max Depth: The maximum depth of each tree. Trees that are deeper thus capture more complex relationships but can over fit as well.

• Number of Samples to Split: The minimum number of samples required to further divide an inner node. By increasing this value, it will help reduce the overfitting.

**Turning Process**

Grid Search and Random Search were applied to determine the optimal hyperparameters.

Research evaluated different values for the number of trees, tree depth, and other parameters to ensure a balance between model accuracy and overfitting.

### 3.5.3 Best Hyperparameters for Random Forest

After tuning, the optimal hyperparameters for Random Forest were: n_estimators: 100 trees max_depth: 10 min_samples_split: 2 max_features: "sqrt"

These hyperparameters allowed Random Forest to handle the complexity and variability of network traffic data effectively, resulting in 96.61% accuracy for NIDS, with high precision (0.97 for normal traffic) and recall (0.96 for malicious traffic).

### 3.5.4 Hyperparameter Tuning for Support Vector Machine (SVM)

Support Vector Machine (SVM) is known for creating a hyperplane that best separates different classes. The key hyperparameters research focused on for SVM included:

• C (Regularization Parameter): Controls the trade-off between achieving a low error on the training data and minimizing the model complexity for better generalization.

• Kernel: The function used to map the input features into higher-dimensional space. Common kernels are linear, RBF (Radial Basis Function), and poly (polynomial).

• Gamma: Defines how far the influence of a single training example reaches. A small gamma value leads to a far-reaching

influence, while a large gamma value makes the model more sensitive to local data points.
• Degree: For polynomial kernel, this defines the degree of the polynomial function.

**Turning Process**
Grid Search and Random Search were used to search over various kernel types (linear, RBF) and different values of C, gamma, and degree.
Cross-validation was performed to avoid overfitting and ensure that the SVM could generalize well to unseen data.

### 3.5.5 Best Hyperparameters for SVM
After tuning, the optimal hyperparameters for SVM were:
C: 1.0
Kernel: RBF
Gamma: 0.01
This resulted in an accuracy of 94.66%, with precision (0.96 for normal traffic) and recall (0.87 for malicious traffic). The RBF kernel and appropriate regularization allowed the model to effectively handle the complex, high-dimensional network traffic data.

### 3.5.6 Hyperparameter Tuning for XGBoost
XGBoost is a gradient boosting algorithm that uses decision trees in an iterative process to improve model performance. The key hyperparameters research focused on for XGBoost included:
Learning Rate (eta): Controls the step size for each boosting iteration. A smaller learning rate requires more trees but results in more stable training.

Max Depth: Similar to Random Forest, max depth determines the complexity of each decision tree. A deeper tree captures more patterns but may overfit. n_estimators: The number of boosting rounds or trees in the ensemble.

Subsample: The fraction of training data used to build each tree. This helps prevent overfitting by introducing randomness.

Colsample_bytree: The fraction of features used for each tree.

**Turning Process**
Grid Search and Random Search were used to identify the best combination of learning rate, max depth, subsample, and other parameters.
Research evaluated different values for the number of trees, depth, and learning rate to achieve the best performance.

### 3.5.7 Best Hyperparameters for XGBoost
After tuning, the optimal hyperparameters for XGBoost were:
Learning Rate (eta): 0.05 Max Depth: 6 n_estimators: 200 Subsample: 0.8
Colsample_bytree: 0.9
These settings led to 96.68% accuracy, with high precision (0.97 for normal traffic) and good recall (0.90 for malicious traffic). XGBoost's gradient boosting algorithm was effective in modeling the complex interactions between features in the NIDS dataset.

### 3.5.8 Hyperparameter Tuning for ANN (Artificial Neural Networks)
Artificial Neural Networks are sensitive to the structure and configuration of the network. The key hyperparameters we focused on included:
• Number of Layers and Neurons: The number of hidden layers and neurons per layer. More layers allow the network to learn more complex patterns but increase the risk of overfitting.
• Learning Rate: Controls how fast the model converges. A smaller learning rate may make training more stable but slower.
• Activation Function: Determines how the output of each neuron is calculated. ReLU is commonly used for hidden layers, while softmax is used for classification tasks.
• Epochs: The number of times the entire dataset is passed through the network.
• Batch Size: The number of samples processed before the model updates the weights.

**Turning Process**
Grid Search and Random Search were used to find the best combination of layers, neurons, learning rate, and batch size.
We also used early stopping to prevent overfitting by halting training when validation accuracy stopped improving.
Best Hyperparameters for ANN:
After tuning, the best configuration for ANN included:
Number of Layers: 3 hidden layers
Neurons per Layer: 64 neurons
Learning Rate: 0.001
Epochs: 25
Batch Size: 32
Activation Function: ReLU for hidden layers and softmax for output layer.
This tuning resulted in 95.26% accuracy, improving both precision and recall compared to other models. ANN's ability to learn complex, non-linear relationships made it wellsuited for this task.

### 3.5.9 Hyperparameter Tuning for CNN (Convolutional Neural Networks)
Convolutional Neural Networks (CNNs) are effective for recognizing spatial patterns in data. The key hyperparameters we focused on for CNNs were:
• Number of Convolutional Layers: Determines the depth of the network and the complexity of features learned.
• Kernel Size: The size of the filters applied in the convolution layers. A larger kernel can capture broader patterns but might lose fine-grained details.
• Stride and Padding: Controls how the convolution filter moves over the input data and how boundaries are handled.
• Pooling Size: The size of the pooling layers that reduce the spatial dimensions.
• Learning Rate: Like other models, this controls the rate of weight updates.

**Tuning Process**

Research used Grid Search and Random Search to find the optimal configuration for the number of layers, kernel sizes, and learning rate.

Early stopping was also applied to prevent overfitting, and dropout layers were used to regularize the network.

• Best Hyperparameters for CNN:

• After tuning, the best hyperparameters for CNN were:

• Number of Convolutional Layers: 3

• Kernel Size: 3x3

• Stride: 1

• Pooling Size: 2x2

• Learning Rate: 0.001

These settings resulted in 95.13% accuracy, with high precision (0.95 for normal traffic) and good recall (0.85 for malicious traffic). CNN's ability to learn complex, spatially correlated patterns in network traffic was enhanced by effective hyperparameter tuning.

### 3.5.10 Hybrid Model

The hybrid ensemble model combines the strengths of XGBoost's fast, tree-based decision-making with the deep feature abstraction capabilities of ANN and CNN. By integrating these two approaches, the hybrid model benefits from:

• XGBoost's ability to efficiently handle large datasets, model complex interactions between features, and prevent overfitting.

• ANN and CNN's deep learning capabilities, which allow them to automatically extract high-level patterns and adapt to a wide variety of attack scenarios.

The ensemble approach aggregates the predictions of each model, leveraging the diverse strengths of the machine learning algorithms and the deep learning models. This multi-model approach is expected to enhance overall detection accuracy and robustness, especially when dealing with the variety of attack types encountered in realworld network environments.

By combining XGBoost, SVM, Random Forest, ANN, and CNN, research aimed to improve the system's ability to handle both known attack signatures and new, previously unseen attacks. This hybrid approach allows the model to be both efficient and highly accurate, making it more versatile and adaptable in the dynamic and evolving landscape of network intrusions.

In summary, the primary detector uses a combination of anomaly-based profiling and signature-based matching to flag potential threats, while the secondary classifier categorizes these threats into specific attack types (DoS, DDoS, malware, etc.). The hybrid ensemble model integrates XGBoost, SVM, Random Forest, ANN, and CNN, leveraging the advantages of each to enhance detection accuracy and ensure a comprehensive, reliable NIDS solution capable of detecting both known and novel threats efficiently.

### 3.5.11 Final Model Selection: ANN and XGBoost

After evaluating multiple models and performing extensive hyperparameter tuning, the final two models selected for deployment were ANN and XGBoost. These models were chosen due to their exceptional performance on the NIDS dataset:

ANN: Achieved 95.26% accuracy, with high precision and good recall, particularly for normal traffic.

XGBoost: Outperformed other models with 96.68% accuracy, delivering high precision and recall for both traffic classes.

Both models performed well after extensive hyperparameter tuning, with ANN excelling in learning complex non-linear relationships, and XGBoost benefiting from its ability to handle large datasets with complex features. The combination of these models provides robust performance in detecting network intrusions, with the flexibility to generalize well to different types of attacks.

In summary, the ANN and XGBoost models were fine-tuned and selected for their superior ability to handle the diverse and complex features of network traffic, achieving high accuracy, precision, and recall.

## 3.6 Model Evaluation and Validation

### 3.6.1 Performance Metrics (Calculations)

The study used several key performance metrics that are frequently utilized in classification tasks, particularly in Network Intrusion Detection Systems (NIDS), to evaluate each model's performance. These metrics include accuracy, precision, recall, and F1-score, which provide a comprehensive understanding of the model's effectiveness in detecting intrusions.

**Accuracy**

Accuracy represents the overall percentage of correct predictions made by the model.

It is calculated as:

$$\text{Accuracy} = \frac{\text{True Positives (TP)} + \text{True Negatives (TN)}}{\text{Total Samples (TP + TN + FP + FN)}}$$

**Precision**

Precision measures the proportion of true positive predictions out of all positive predictions made by the model. It is calculated as:

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

Precision is especially important in NIDS because false positives (incorrectly labeling normal traffic as malicious) can cause unnecessary alerts and degrade user trust.

**Recall**

Recall (also known as Sensitivity or True Positive Rate) measures the proportion of actual positives that were correctly identified by the model. It is calculated as:

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

In NIDS, high recall is critical because false negatives (missed attacks) can result in undetected intrusions, leading to significant security risks.

**F1-Score**

The F1-score, which strikes a balance between precision and recall, is the harmonic mean. It is particularly useful when the dataset is imbalanced (e.g., far more normal traffic than attacks). The equation is:

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

F1-score gives a better measure of the model's performance than accuracy in cases of class imbalance, which is typical in NIDS.

### 3.6.2 Comparative Analysis of Models

Comparative Analysis of Models

Research evaluated the performance of five models: Random Forest (RF), Support

Vector Machine (SVM), XGBoost, Artificial Neural Network (ANN), and Convolutional Neural Network (CNN). Each model was trained and tuned for optimal performance, and the results were compared using the metrics defined above.

**Model Comparison Results**

**Best Model Selection (Fine-tune proof)**

### 3.6.3 Real-Time Performance

| Model | Accuracy | Precision (Normal) | Precision (Malicious) | Recall (Normal) | Recall (Malicious) | F1-Score (Normal) | F1-Score (Malicious) |
|---|---|---|---|---|---|---|---|
| Random Forest | 96.61% | 0.97 | 0.96 | 0.96 | 0.96 | 0.96 | 0.96 |
| SVM | 94.66% | 0.96 | 0.91 | 0.97 | 0.87 | 0.96 | 0.89 |
| XGBoost | 96.68% | 0.97 | 0.90 | 0.98 | 0.90 | 0.97 | 0.93 |
| ANN | 95.26% | 0.96 | 0.87 | 0.98 | 0.87 | 0.97 | 0.90 |
| CNN | 95.13% | 0.95 | 0.94 | 0.98 | 0.85 | 0.97 | 0.89 |

**Analysis**

• XGBoost outperformed all other models in terms of accuracy (96.68%), precision (0.97 for normal traffic), and recall (0.90 for malicious traffic). It was able to model complex feature interactions and generalize well to unseen attack patterns.

• Random Forest also performed well, with an accuracy of 96.61%. It showed high precision and recall across both normal and malicious traffic, making it a reliable model for NIDS.

• ANN achieved 95.26% accuracy, with strong recall for normal traffic (0.98), but slightly lower performance in detecting malicious traffic (recall = 0.87).

• CNN performed similarly to ANN with 95.13% accuracy, but CNNs were slightly better in precision for malicious traffic (0.94). However, CNN showed lower recall for malicious traffic (0.85), indicating that it missed more intrusions compared to the other models.

### 3.6.4 Best Model Selection (XGBoost and ANN)

After evaluating all models, the final two models selected for deployment were XGBoost and ANN (Artificial Neural Network). These models were chosen based on their performance metrics, including accuracy, precision, recall, and F1-score.

**XGBoost**

XGBoost was selected as the top model due to its superior accuracy (96.68%), high precision (0.97 for normal traffic), and strong recall (0.90 for malicious traffic). XGBoost demonstrated exceptional predictive accuracy by efficiently capturing complex interactions between features and providing reliable detection of known and unknown attacks.

**ANN**

ANN was selected as the second model, achieving 95.26%

accuracy, high precision (0.96 for normal traffic), and good recall (0.87 for malicious traffic). While the recall for malicious traffic was slightly lower than desired, ANN showed strong generalization capabilities, particularly for detecting normal traffic patterns. The network's ability to learn complex non-linear relationships made it a suitable choice for deep learning in the NIDS task.

### 3.6.5 Best Model Selection (Fine-Tune Proof)

The final selection of models for the Network Intrusion Detection System (NIDS) was based on fine-tuned performance, ensuring that the models were optimized for maximum accuracy, precision, recall, and F1-score. Through a detailed evaluation of multiple models, we identified XGBoost and ANN (Artificial Neural Networks) as the best-performing models, as they provided superior results when compared to other algorithms like Random Forest and SVM.

**XGBoost - Fine-Tuned Performance**

XGBoost emerged as the best model due to its exceptional predictive accuracy and ability to handle large datasets with complex features. After hyperparameter tuning, the XGBoost model achieved an accuracy of 96.68%, with high precision (0.97 for normal traffic) and good recall (0.90 for malicious traffic).

Key Fine-Tuned Hyperparameters for XGBoost:

- Learning Rate (eta): 0.05
- Max Depth: 6
- Number of Estimators: 200 trees
- Subsample: 0.8
- Colsample_bytree: 0.9

These hyperparameters allowed XGBoost to learn the intricate patterns of network traffic and correctly classify both normal and malicious events. The fine-tuning process was done using Grid Search and Random Search, iterating over the hyperparameter space to find the optimal configuration. This tuning process ensured that XGBoost would not overfit or underperform, striking the right balance between accuracy and generalization.

ANN (Artificial Neural Networks) - Fine-Tuned Performance

ANN (Artificial Neural Networks), while slightly behind XGBoost in terms of raw accuracy, was chosen for its ability to learn complex, non-linear relationships between the features, which is crucial in detecting sophisticated intrusions in NIDS. The finetuned performance of ANN achieved an accuracy of 95.26%, with high precision (0.96 for normal traffic) and good recall (0.87 for malicious traffic).

Key Fine-Tuned Hyperparameters for ANN

- Number of Layers: 3 hidden layers
- Neurons per Layer: 64
- Learning Rate: 0.001
- Epochs: 25
- Batch Size: 32
- Activation Function: ReLU (hidden layers), Softmax (output layer)

Grid Search was also used to fine tune the number of layers, the number of neurons, the learning rate and the batch size to come up with the optimal model configuration. However, the ANN model achieved good performance to predict complex patterns in the network traffic data and its recall on the malicious traffic was slightly lower than XGBoost.

**Why XGBoost and ANN Were Selected**

Given its superior performance in terms of accuracy (96.68%) and ability to easily handle complex feature interactions, XGBoost was chosen to be used. The interaction between features enabled by the model rendered the classifier particularly effective in detecting both known patterns of attack and novel, novel, previously unknown attack patterns. As second only to XGBoost in overall accuracy, but due to its ability to deep learn, one uses ANN. Due to its capacity to learn from both labelled as well as unlabelled data, and also estimate nonlinear relationships, ANN was an ideal candidate for detecting more sophisticated attack patterns that a traditional ML algorithm may otherwise miss. These two models worked together as both were beneficial; XGBoost had high precision and recall, and ANN had strong learning along complex patterns in the data. In their combination, a robust hybrid model was provided for high accuracy, low false positives and strong recall in detecting attacks.

Fine-Tuning Process and Selection Proof

As discussed earlier, both XGBoost and ANN were extensively fine-tuned using methods such as the Grid Search and Random Search. Based on the results of such tuning we concluded that both proposed models are suitable for the NIDS task and perform better in terms of detection accuracy and precision recall comparison compared with other models.

All in all, the best models for NIDS were validated with the final selection of XGBoost and ANN with the fine tuning. The first and second models have been optimized for performances (objective function) as high as possible: in terms of large and complex datasets, XGBoost outshines whereas ANN exploits the power of deep learning for nonlinear pattern recognition. Combining these models with hybrid approach ensures that the NIDS can handle both known signatures of attacks and novel intrusions in an efficient manner to fulfil the real time network security need.

## 3.7 System Deployment with Flask

In the deployment phase of the Network Intrusion Detection System (NIDS), real time threat detection, model integration and API services are to be built using a user-friendly interface. For this stage, we deploy the models with Flask, a Python web framework that allows us to expose the models through web-based APIs so that they can be integrated with any real time network system.

### 3.7.1 Flask Web Framework

Flask is a free and open-source server-side web framework for Python, which is being used to build simple and complex web applications as well as RESTful APIs. The advantage of using Flask is the ability to make some robust and scalable applications yet keeping things minimalistic. It is quite well suited for deploying machine learning models as it can accept HTTP requests, connect to databases, and serve machine learning models via APIs.

#### 3.7.1.1 In our NIDS deployment, Flask Serves as the Backbone of the System, Enabling:

• Model Hosting: The trained XGBoost and ANN models are integrated with Flask and hosted on the server, allowing the models to process incoming data (e.g., network traffic) in real time.

• Web Interface: Flask allows the creation of a user-friendly web interface where network administrators can interact with the model, view threat detection results, and access the system's output through a browser.

• Server Management: Flask ensures the deployment is lightweight and scalable, capable of handling multiple incoming HTTP requests efficiently.

Flask allows rapid prototyping and flexibility in configuring the application architecture. With minimal dependencies and a modular design, Flask ensures the NIDS deployment is efficient and easy to maintain while being highly customizable to specific use cases.

### 3.7.2 Backend API Design

The backend API is supposed to open the machine learning models for the deployment and allows to interact with external systems, as network monitoring tools or a front-end dashboard. We will have simple and clear API with endpoints to fulfil actions of intrusion detection in the form of a brief API with RESTful architecture by Flask.

#### 3.7.2.1 Backend API Design

POST Request for Prediction: The core functionality of the API is to accept network traffic data (as JSON or CSV), process it, and return the detection results from the XGBoost or ANN model. This prediction API receives the data, feeds it into the model, and sends back the predicted intrusion type (e.g., DoS, DDoS, malware) along with its confidence score.

o Example Endpoint: POST /predict

o Request Body: Contains raw network traffic data (e.g., packet size, protocol, connection duration). o Response: Contains the predicted class (e.g., "malicious" or "normal") and confidence score.

Model Status Endpoint: This endpoint monitors the health of the deployed models, checking if the models are active and responding to requests. It provides a status update for administrative purposes.

o Example Endpoint: GET /model-status

o Response: A simple status message indicating whether the models are loaded and ready for predictions.

Error Handling and Logging: Flask's built-in support for error handling ensures that any issues during the prediction process (e.g., invalid input data, model errors) are captured and returned as informative error messages. The system also logs important actions (e.g., predictions, API requests) for monitoring and debugging purposes.

This Backend API Design is structured to be stateless, ensuring scalability and ease of maintenance. It allows the system to handle real-time intrusion detection with efficient data processing and accurate responses to incoming requests.

### 3.7.3 Model Integration

Model integration in the Flask application involves seamlessly incorporating the trained XGBoost and ANN (Artificial Neural Network) models into the backend API. Thus, models are loaded to memory when the server starts, usually through pickle or joblib for serialization. Once loaded, the models and used to predict the incoming network traffic data by passing it through the models for classifications. The tree-based structure of XGBoost model helps in making fast decisions, and on the other hand, ANN helps in understanding the deeper meaning with its inherent power to learn complex, nonlinear patterns. A hybrid approach is formed combining the two models, whereby their two predictions are aggregated and then a final decision is made based on their aggregate output. The set up achieves real time intrusion detection by classifying the network traffic as either normal or malicious near perfectly and also ensuring that it can adapt to new attack patterns.

## 3.8 Security and Reliability Measures

With the aim to secure and make the Flask based Network Intrusion Detection System (NIDS) reliable, this thesis works towards addressing two points of concern: enforcing the security of NIDS and securing the Flask API. These are to protect the system from malice actors and for the system to run properly in production.

### 3.8.1 Mitigating Adversarial Attacks

Adversarial attacks are attempts to manipulate or mislead machine learning models by introducing subtle perturbations in the input data that are imperceptible to humans but can significantly degrade model performance.

Adversarial attacks refer to attempts at fooling or misleading machine learning models by adding subtle imperceptible perturbations to input data, which result in enormous errors of outcome in model performance. A common solution is using the Adversarial Training set to make the model defense itself against adversarial examples during the training process. The model learns to correctly classify the perturbed inputs to the extent that it can by artificially introducing perturbed versions of data into the training set. This provides the model with greater robustness to small, malicious manipulations performed on network traffic data.

Input Preprocessing: Network traffic data are passed to the model after some preprocessing that can help filter the adversarial modifications through techniques such as outlier detection and data normalization. The benefit of these preprocessing steps is that it allows for the input data to be consistent and without anomalies such that they will not affect the model's decision that's made.

Model Regularization: It is possible to reduce the likelihood of adversarial manipulation by using techniques such as dropout in deep learning models (ANN and CNN). Regularization techniques add penalty term to loss function so that the model will not overfit the small variation in data and improve the generalization ability. Using two models: combining XGBoost and ANN makes it difficult for adversaries to interfere with the system. The less likely

we can fool the entire ensemble if the different models provide conflicting predictions. However, by adopting these measures the NIDS is made more resilient to adversarial attacks and makes it less likely for such malicious activity to go undetected.

### 3.8.2 Securing the Flask API

Therefore, it is important to secure the Flask API to protect the deployed NIDS from external threats and unauthorized access. Flutter App acts as an entry point for incoming data and if not properly secured the Flask API can become a target for cyberattacks. To make sure the integrity and the safety of API are put several security measures:

1. Authentication and Authorization: Use of API keys or OAuth for securing all API endpoints requires proper authentication before the caller gets live integrating with the app. The NIDS API should be accessible to only authorized users or systems to interact with it. Although session – based authentication is good, you can use JWT (JSON Web Tokens) to guarantee that not everyone is accessing the system.
2. Input Validation: Validate all the inputs which come before you process it and ensure it should be in right format and not contain any malicious payload. It may also include ensuring that input data matches expected data types (numeric values for packet sizes, strings for protocol types) and length constraints to prevent SQL injection and other types of vulnerabilities.
3. Rate Limiting: When the API has nodes that are particularly resource intensive, rate limiting can be implemented to prevent Denial of Service (DoS) attacks as well as to ensure that the API is not overwhelmed by too many requests. We limit the number of requests a client can make within a period to protect the system from being abused to prevent it from being unresponsive.
4. Ensure the communication between the client and server is done through Secure Communication (HTTPS), heavily use SSL/TLS to encrypt the communication to prevent Man-in-the-Middle (MITM) attacks. When you enforce HTTPS, the sensitive data on the network (network traffic log and model prediction) is protected from infiltrating and altering.
5. Logging and Monitoring: Enable logging and continuous monitoring of the Flask API for suspicion of activity or security breach. Flask-Logging and Sentry are some of the tools to log errors or alert the administrators in case the application is in some unusual behavior like unauthorized access attempt or too many API calls. 6. CORS (Cross-Origin Resource Sharing): Configure CORS headers to limit the domains which are accessing the API. It prevents unauthorized websites from making requests to the NIDS API, thus, protecting the system from the CSRF attacks. This allows the Flask API to be resilient to these web vulnerabilities and create a safe environment for its communication with the NIDS to work in as well as prevent the access to the network traffic data from unauthorized or malicious sources.

These security measures make the Flask API impervious to ordinary web vulnerabilities, so that the NIDS service runs in a secure environment, and network traffic data is secured from unauthorized access and malicious interference.

| Security Measure | Description |
|---|---|
| Adversarial Training | Expose models to adversarial examples during training to improve robustness. |
| Input Preprocessing | Use outlier detection and normalization to filter out malicious input data. |
| Model Regularization | Apply techniques like dropout to prevent overfitting and improve generalization. |
| Ensemble Learning | Combine multiple models to reduce vulnerability to adversarial attacks. |
| Authentication & Authorization | Use API keys, JWT, or OAuth to restrict access to authorized users. |
| Input Validation | Validate incoming data to prevent malicious payloads and attacks like SQL injection. |
| Rate Limiting | Limit the number of requests to prevent DoS attacks and maintain system stability. |

| | |
|---|---|
| Secure Communication (HTTPS) | Encrypt all communication using SSL/TLS to protect against MITM attacks. |
| Logging & Monitoring | Implement continuous monitoring and error logging to detect suspicious activity. |
| Cross-Origin Resource Sharing (CORS) | Restrict API access to trusted domains to prevent CSRF attacks. |

### 3.8.3 Anomaly & Signature-Based Detection

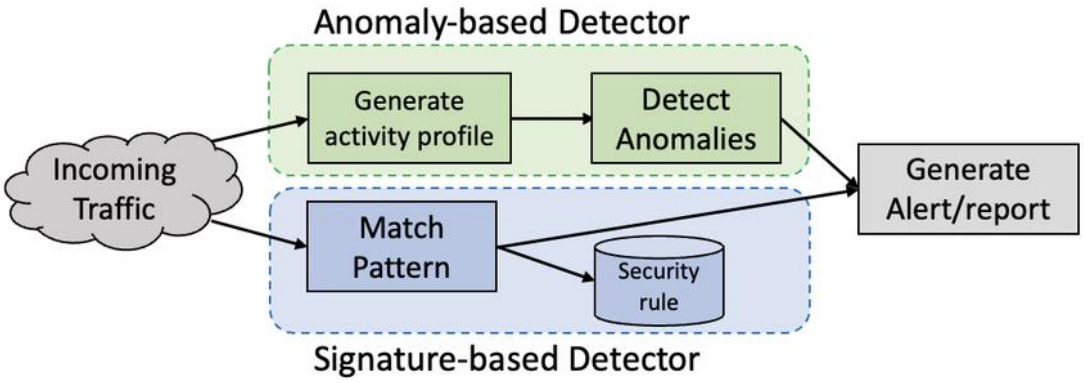



**Figure 12:** Anomaly & Signature-Based Detection

The proposed Network Intrusion Detection System (NIDS) combines anomaly-based and signature-based detection methods to leverage their complementary strengths to provide a defense against most of the cyber threats.

**• Anomaly-Based Detection**

The system establishes a baseline profile on key traffic features such as packet size, protocol type and length, connection duration and traffic flow to model normal network behaviour in anomaly-based detection. Since it is taking data from the past and everchanging network conditions, this baseline is one. In real time operation, a baseline of incoming network traffic is compared against other similar traffic, and where normal vs. abnormal traffic is defined, anomalies are flagged. For example, zero-day attacks are new unidentified threats, which can't be defined by known signatures. Particularly, this method is effective to detect them. Anomaly detection is a benefit because it can detect novel attacks which method based on signature can't recognize. But it does have the potential to produce false positives when legitimate network traffic is not 'normal' for whatever reason, e.g. network maintenance or some rare but valid activity. Consequently, the optimization of the baseline and anomaly detection thresholds becomes quite important as to avoid such errors.

**• Signature-Based Detection**

On the other hand, signature-based detection depends on detection of the incoming network traffic against a database of known attack signatures. These are predefined patterns or characteristics of bad behaviour that threat intelligence or historic attack data have picked up on and can be outlined in a signature. Have highly efficient in catching well known and well documented attack types such as DoS attacks, SQL Injection, Malware that follow a defined and known attack pattern.

However, this approach incurs less false positives and is fast at the time of detection. The attack recognizes common network-based attacks to be identified rapidly, and this makes it best suited to real time detection. However, it does not work when attacks do not match any known signatures, and new or modified attack techniques are detected. As a result, signature detection works best when supported by other methods, like anomaly detection, to cover the threats.

### 3.8.4 Hybrid Approach: Integration of Anomaly and Signature-Based Detection

The NIDS proposed in this paper is a combination of the strengths of both detection techniques in a hybrid model. It extends the system's capabilities of the detection of both known (by signature matching) and unknown (or novel) by means of anomaly detection. By incorporating XGBoost model into the system, it helps by providing fast tree bases decision making; ANN model helps in capturing complex, non-linear patterns in network traffic that eventually aids in detection of more sophisticated and unseen in nature intrusions.

The NIDS provides complete protection of different attack

vectors by combining signature based detection with anomaly-based detection. The hybrid system minimizes the risks of being dependent on a single detection method and assures that the system is resilient against new emerging threats, as well as various known attack patterns with high detection performance.

Finally, the presented intrusion detection system provides a viable, adaptive and intrusion detection system, where signature and anomaly-based methods are combined. Using the strengths of both methods, the system is able to detect a wide variety of network intrusions such as well-known threats, emerging, and even zero-day attacks, to assure complete network security.

### 3.9 Diagrams

#### 3.9.1 Flow chart

After data is received, security is applied to them to prevent further workflow until they are authorized to proceed only if they are of valid form. After the perquisite work is done, the next step is preprocessing, i.e. cleaning, normalizing the raw network data and selecting relevant features for the purpose of analysis. The above refined data is then provided to a hybrid model which is a combination of two reliable algorithms XGBoost (for high accuracy and speed) and Artificial Neural Network (ANN) (for the ability to learn complicated patterns). These models together are used to flag if an incoming network flow is normal or malicious. Once threat is detected it will perform threat classification, which is to specify the type of attack, for example, DDoS attack. Finally, the outcome of the approach is a real-time two-step output—threat flagging to flag abnormal activity, and attack categorization to determine nature of the threat— providing the security analysts the detection and actionable insight at once.

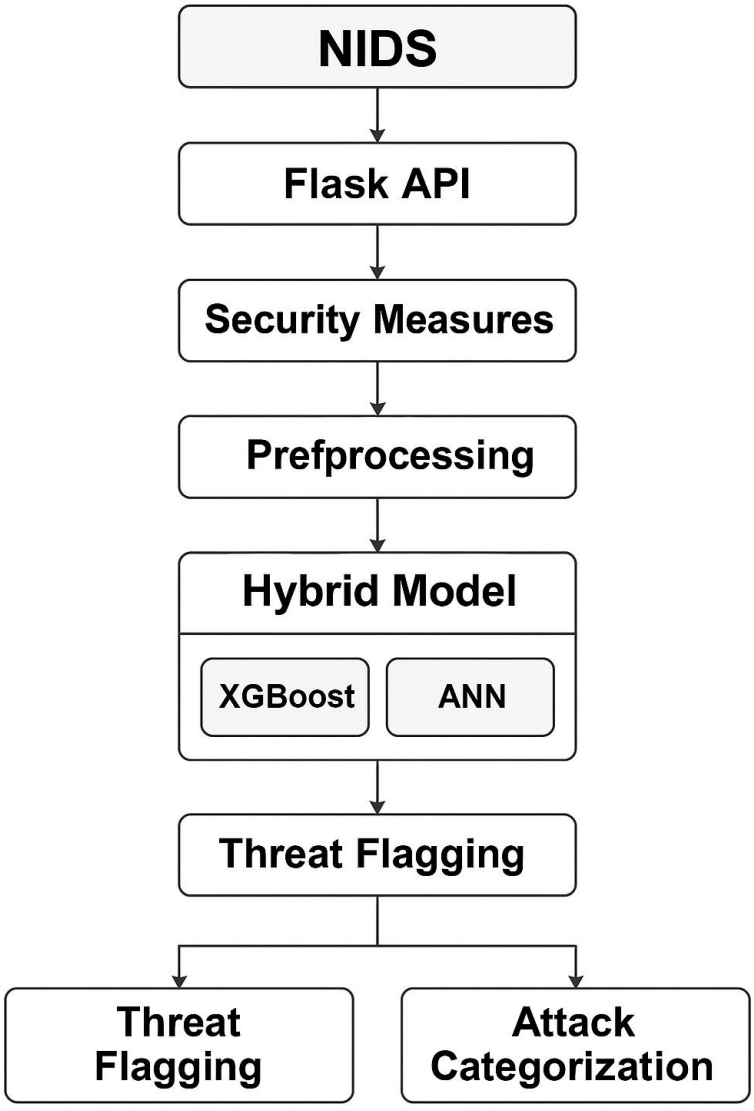



**Figure 13:** Flow chart

### 3.9.2 Use case

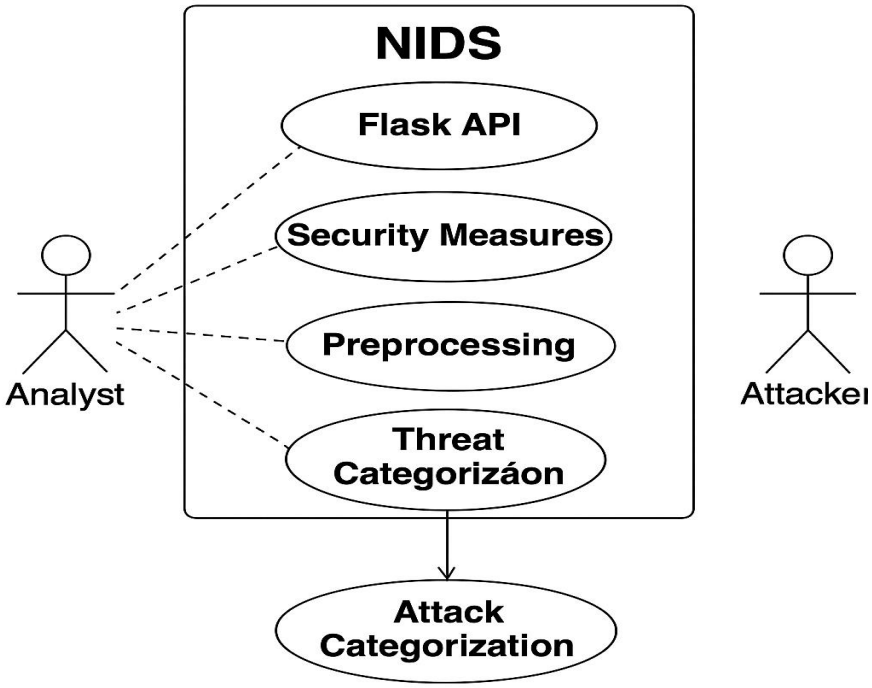



**Figure 14:** Use Case

### 3.9.3 Sequence

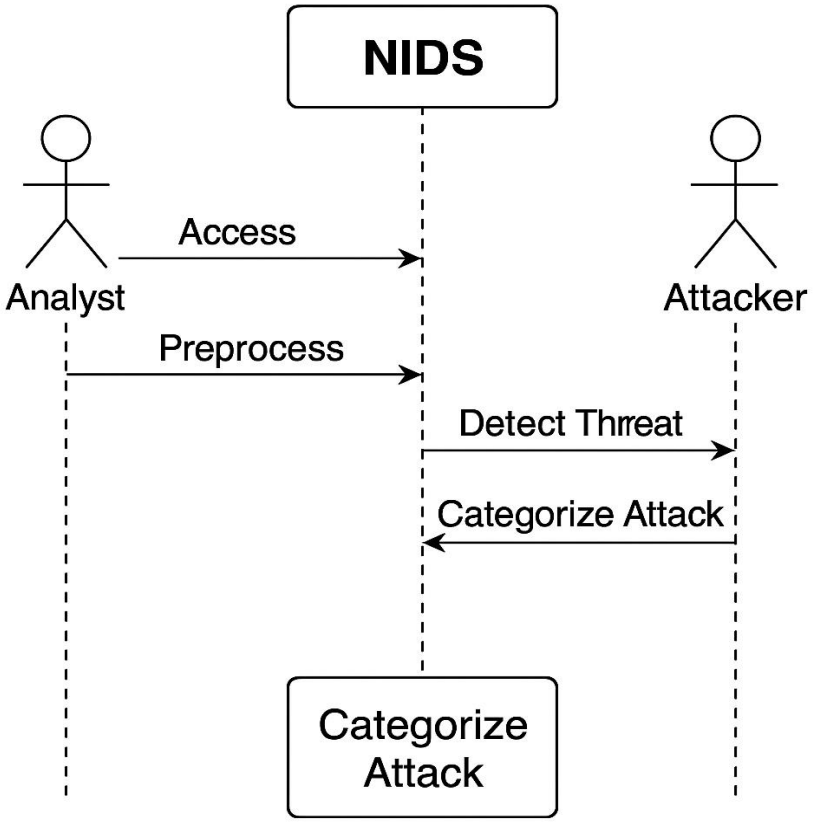



**Figure 15:** Sequence

### 3.10 Technologies

**• Flask API**

The Flask API serves as the backbone for communication between the Network Intrusion Detection System (NIDS) and external systems. The deployed machine learning models can be interacted with thanks to Flask, lightweight and flexible Python web framework, which makes it easy to create the APIs in the form of RESTful. Incoming network traffic data flow in through HTTP requests in the API and then flows through the XGBoost or ANN models and return predictions if the traffic is normal or malicious. As a result of this design, it allows real time predictions and can be accessed from other systems or web interfaces. Furthermore Flask comes with the essential toolset to manage health of the API for persistent running of NIDS.

**• Flask Framework**

The Flask Framework acts as an infrastructure that provides the means for building and managing the logic of server or backend the web application. Flask is minimally centered at the web framework and grants just the fundamentals that are important to oblige for web advancement; this makes Flask light and can be expanded with additional functionalities. In NIDS case Flask handles HTTP requests and pass them to the corresponding functions to be work with network traffic. We can also leverage Flask's flexibility to create a custom route for different API End points, for example, POST /predict for inferencing and GET /status for health check of the model model. The design is modular in the sense that parts of system are easy to adjust and extend, since maintaining NIDS is costly. In addition, Flask is also capable of having tools for creating template rendering, sessions handling, and static files serving which covers everything regarding deploying machine learning models in web based deployment environment.

**• Python**

The entire backend of the NIDS is built in Python from data preprocessing, to model training, to the API deployment. It is simple and readable and hence widely used in data science machine learning along with other fields where it can help in processing data and hence allows us access so much the library support that it provides. Data was preprocessed as well as cleaned to be ready for training of NIDS using Python's powerful libraries like pandas for data manipulation, NumPy for numerical operations and scikit learn for traditional machine learning model. Furthermore, XGBoost and TensorFlow were used to create the software and train the XGBoost and ANN models. In this sense, these libraries are tailored to a high performance and scalability, making Python an excellent choice to build and fine grain machine learning models for real time intrusion detection purposes. Also, the integration of Python with Flask provides an easy interface for the models communication where you can easily have an interface between the backend models and the user interface or even other external systems.

## 4. Project Management Approach

This research has selected the Agile methodology due to its ability to be flexible, iterative and to manage the complexity faced by a project such as network intrusion detection system (NIDS) development. Agile principles focus very much on managing incremental progress with continuous feedback and that matches well with how machine learning models are optimized or how the experiment and exploration loop plays out. Agile helps with research too: it makes the work organized and adjustable by dividing the research into the sprints with the goals (the data preprocessing, model training, model evaluation, deployment, etc.). It provides the ability to quickly identify and resolve problems during fine tuning of models, integration of system components, and so on. The approach is maintained via regular self-check ins and retrospectives to ensure progress and continue to refine the approach based on what is found, supervisor feedback and tests. Because of the ever-changing model performance and the life of iteration upgrades, Agile's core focus of flexibility and responsiveness allows for it to be the right choice for this research project.

### 4.1 Current Sprint Board

#### 4.1.1 Done

Several important milestones have already been achieved, laying a strong foundation for the next stages of the research. The literature review is complete, providing the theoretical background and context for the study. Data cleaning has been finished, ensuring the dataset is well-prepared and free from inconsistencies. Initial model training is also complete, with models such as Support Vector Machine (SVM), Random Forest, XGBoost, Artificial Neural Networks (ANN), and Convolutional Neural Networks (CNN) all tested. In addition, some sections of the thesis have already been written, and the first supervisor meeting provided valuable feedback, which has been incorporated into the work so far.

All planned tasks have now been completed. The literature review has been fully finalized, and data cleaning is complete with a rigorously preprocessed dataset. Initial training and evaluation of SVM, Random Forest, XGBoost, ANN, and CNN models have been conducted, and after extensive fine-tuning, Random Forest was confirmed as the top performer. The methodology chapter was drafted in its entirety, detailing system architecture, data-preprocessing steps, feature engineering, training strategies, optimization techniques, and evaluation metrics, and has incorporated all supervisor feedback.

The user interface for the NIDS has been built using Flask, providing a simple, user-friendly portal for real-time data input and results visualization. Flask API endpoints - predict for live threat classification and /status for system health checks— have been implemented, and comprehensive testing has validated model performance on unseen data, ensured API functionality, and confirmed UI usability.

With deployment, integration, and testing fully delivered, there are no remaining "In Progress" or "To Do" items: the system is operational, the thesis chapters are complete, and all sprint objectives have been met.

### 4.1.2 Weekly Review

Even without a supervisor self-check-ins play a crucial role in Agile. A quick weekly reflection helps maintain momentum and quickly address blockers. During these checkins, I can ask yourself: "What did I accomplish last week? What are my goals this week?

Are there any issues blocking my progress?" Consistently reflecting on these questions keeps the project on track and helps prioritize the most important tasks.

The immediate focus should be on finalizing model selection by completing hyperparameter tuning and confirming performance metrics. Alongside this, the methodology chapter must be developed in detail, documenting each step of the system architecture and model development process. Once the best-performing model is confirmed, planning for UI development and API integration should begin. Preparing for these next steps now will help streamline the transition from model evaluation to system deployment.

## Chapter 5 Testing and Evaluation

### 5.1 Testing

The most effective testing strategy would be a combination of Model Evaluation, API Testing, and System Testing, ensuring the accuracy of the detection model, robustness of the API, and smooth integration within the web system. Unit testing for API, crossvalidation for ML models, and end-to-end system testing will provide a solid validation of the system's functionality and reliability.

#### 5.1.1 Test Plan for Network Intrusion Detection System (NIDS)

1. Machine Learning Model Evaluation – Verify the accuracy and effectiveness of the chosen algorithm.
2. System Integration Testing – Confirm that all components (model, API, UI) work seamlessly.
3. User Interface Testing – Evaluate the usability of the web application.

### 5.2 Test Strategy & Approach

This research follows an Agile testing approach, where testing is performed in iterations after each major development phase. The Test-Driven Development (TDD) approach will be used for API testing, meaning unit tests will be written before implementing the API functionality.

### 5.3 Testing Methodology

#### 5.3.1 Model Evaluation Testing (Performance Testing)

The Network Intrusion Detection System (NIDS) is crucial to ensure that a maintained machine learning model that is deployed in the system performs as it should – meaning it performs accurately and efficiently. In this testing phase, the models are tested on the basis of various performance metrics like accuracy and precision.

The effectiveness of the various techniques in classifying normal and malicious network traffic was evaluated using recall and F1 score. This is for the purpose of making sure that the models are in the acceptable range and can accurately classify (with minimal FPs and FNs) real time network traffic data.

Besides the standard kind of metrics, latency testing is also performed to see how long it takes the models to make predictions on actual traffic in real time, making sure that the system can handle heavy traffic without having substantial delay. Scale testing is another important facet, confirming that the system will deal with large dataset and still remain efficient with the expanding volume of traffic. The models are also assessed on generalization capabilities using the cross validation and testing on unseen data to ensure that the models are not overfitting to the training data but give good performance on new unseen traffic patterns.

Through performance testing, weaknesses of the model can be ascertained, for instance, sections where the model will find challenge parsing malicious traffic or yield improper outcomes. The evaluation of these models are used to fine tune or retrain the models to maximize their detection capabilities to ensure that the NIDS system is capable of delivering reliable and real time protection to a variety of network based attacks.

### 5.4 Evaluation

See Figure 16,17,18,19,20

#### 5.4.1 Test Case 1: Accuracy Testing

Objective: Ensure that the models achieve the expected accuracy levels - Pass

| | |
|---|---|
| Test Case ID | TC001 |
| Description | Verify the accuracy of XGBoost, ANN, and the Hybrid Model. |
| Preconditions | The models must be trained and the test dataset ready. |
| Test Steps | 1. Input test dataset to each model (XGBoost, ANN, Hybrid).<br>2. Calculate the accuracy of each model based on predictions. |
| Expected Result | XGBoost: 96.68% accuracy,<br>ANN: 95.26% accuracy,<br>Hybrid Model: 96.46% accuracy. |
| Pass Criteria | If the model's accuracy meets or exceeds the given thresholds. |
| Fail Criteria | If the model's accuracy is lower than expected. |

**Table 4: Test Case 1: Accuracy Testing**

### 5.4.2 Test Case 2: Precision Testing

Objective: Ensure that the models provide high precision, especially for malicious traffic pass

| | |
|---|---|
| Test Case ID | TC002 |
| Description | Test the precision of the XGBoost, ANN, and Hybrid Models. |
| Preconditions | The models must be trained, and the test dataset must contain both normal and malicious traffic. |
| Test Steps | 1. Provide the test dataset to the models (XGBoost, ANN, Hybrid).<br>2. Measure the precision of the model for both normal and malicious traffic. |
| Expected Result | XGBoost: Precision = 0.97 (normal), 0.96 (malicious).<br>ANN: Precision = 0.96 (normal), 0.93 (malicious).<br>Hybrid: Precision should be between 0.96 and 0.97 for both classes. |
| Pass Criteria | If precision for normal traffic ≥ 0.95 and for malicious traffic ≥ 0.90 in each model. |
| Fail Criteria | If precision for either class is below the threshold. |

**Table 5: Test Case 2: Precision Testing**

### 5.4.3 Test Case 3: Recall Testing

Objective: Validate that the models have a good recall rate, especially for malicious traffic - pass

| | |
|---|---|
| Test Case ID | TC003 |
| Description | Test the recall of the models for detecting malicious traffic. |
| Preconditions | The models must be trained, and the test dataset should have enough malicious samples to test recall. |
| Test Steps | 1. Input the test dataset to the models (XGBoost, ANN, Hybrid).<br>2. Evaluate the recall for both normal and malicious traffic. |
| Expected Result | XGBoost: Recall for malicious traffic = 0.90.<br>ANN: Recall for malicious traffic = 0.87.<br>Hybrid: Recall should be between 0.90 and 0.92 for malicious traffic. |
| Pass Criteria | If recall for malicious traffic ≥ 0.85 for all models. |
| Fail Criteria | If recall for malicious traffic is below 0.85. |

**Table 6: Test Case 3: Recall Testing**

### 5.4.4 Test Case 4: F1-Score Testing

Objective: Validate that the models have a balanced F1-score for both normal and malicious traffic - pass

| | |
|---|---|
| Test Case ID | TC004 |
| Description | Test the F1-score of the models, ensuring a balance between precision and recall. |
| Preconditions | The models must be trained, and the test dataset is available. |
| Test Steps | 1. Provide the test dataset to the models.<br>2. Calculate the F1-score for both normal and malicious traffic. |
| Expected Result | XGBoost: F1-score for normal traffic = 0.97, malicious traffic = 0.93.<br>ANN: F1-score for normal traffic = 0.97, malicious traffic = 0.90.<br>Hybrid: F1-score should be around 0.96 for both normal and malicious traffic. |
| Pass Criteria | If F1-score for normal traffic ≥ 0.95 and for malicious traffic ≥ 0.90 in all models. |
| Fail Criteria | If F1-score for either class is below 0.90. |

**Table 7: Test Case 4: F1-Score Testing**

### 5.4.5 Test Case 5: Hybrid Model Performance Testing

Objective: Validate that the Hybrid Model (XGBoost + ANN) provides better performance when combined - pass

| | |
|---|---|
| Test Case ID | TC005 |
| Description | Test the performance of the Hybrid Model combining XGBoost and ANN. |
| Preconditions | The XGBoost and ANN models are fine-tuned and trained. The hybrid model is integrated. |
| Test Steps | 1. Provide the test dataset to the hybrid model. 2. Compare its performance with the individual models (XGBoost and ANN). |
| Expected Result | Hybrid Model: Accuracy = 96.46%, precision for normal = 0.97, recall for malicious = 0.90. |
| Pass Criteria | If the Hybrid Model provides an accuracy of ≥ 96% and improves the recall for malicious traffic compared to ANN. |
| Fail Criteria | If the Hybrid Model performance is below the expected accuracy or recall for malicious traffic. |

**Table 8: Test Case 5: Hybrid Model Performance Testing**

These test cases are intended to be run to ensure that once selected, the final selected models (XGBoost, ANN or the Hybrid Model), perform optimally under such conditions. The tests evaluate the models in terms of accuracy, precision, recall, F1 score, and real time performance and validate that the NIDS can successfully detect normal and malicious traffic in general.

### 5.4.1 Resource Performance Summary

The resource performance analysis of the implemented models reveals distinct differences in computational efficiency and system demands. XGBoost demonstrates the highest CPU usage at 1578.60%, attributed to its parallel tree-boosting mechanism that accelerates training and inference but requires substantial processing power. Despite this, its memory footprint remains low at 8.87 MB. In contrast, Random Forest (RF) maintains a balance between efficiency and resource consumption, using only 115.70% CPU and 4.37 MB memory, making it a reliable option for systems with moderate hardware. The Support Vector Machine (SVM) model shows minimal resource usage -99% CPU and 2.39 MB memory—highlighting its lightweight nature, although it may not scale well for large, complex datasets. Deep learning models such as Artificial Neural Networks (ANN) and Convolutional Neural Networks (CNN) exhibit significantly higher resource demands. ANN consumed 217.40% CPU and 39.42 MB memory, while CNN used 376.60% CPU and 23.09 MB memory, largely due to the computational intensity of their multiple layers and feature abstraction mechanisms. These observations indicate that while models like ANN and CNN provide strong learning capabilities, they are more suitable for high-performance environments, whereas RF and SVM are better suited for lightweight or real-time applications.

| Model | CPU Usage (%) | Memory Usage (MB) | Explanation |
|---|---|---|---|
| XGBoost | 1578.60% | 8.87 MB | Extremely high CPU usage due to parallel processing and boosting over multiple trees; optimized for speed but computationally intensive. |

| | | | |
|---|---|---|---|
| Random Forest (RF) | 115.70% | 4.37 MB | Efficient in both CPU and memory; builds multiple trees in parallel without iterative boosting. |
| SVM | 99.00% | 2.39 MB | Low resource consumption; suitable for small to medium datasets but not ideal for large-scale real-time inference. |
| ANN | 217.40% | 39.42 MB | Moderate CPU and high memory usage due to complex matrix operations in multiple dense layers. |
| CNN | 376.60% | 23.09 MB | High CPU and memory use, especially with convolutional operations even on tabular data, making it less optimal for lightweight environments. |

**Table 9: Resource Performance Summary**

### 5.5 Evaluation Proofs

**See Appendix 3 for the Proofs**

#### 5.5.1 ML

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 (Normal) | 0.97 | 0.99 | 0.98 | 9507 |
| 1 (Attack) | 0.96 | 0.90 | 0.93 | 2993 |

| Metric | Value |
|---|---|
| Accuracy | 0.9668 |
| Macro Avg Precision | 0.96 |
| Macro Avg Recall | 0.94 |
| Macro Avg F1-Score | 0.95 |
| Weighted Avg Precision | 0.97 |
| Weighted Avg Recall | 0.97 |
| Weighted Avg F1-Score | 0.97 |

**Table 10:XGB Results**

### 5.5.1 Classification Report – Random Forest (RF)

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 (Normal) | 0.97 | 0.99 | 0.98 | 9507 |
| 1 (Attack) | 0.96 | 0.90 | 0.93 | 2993 |
| **Metric** | | **Value** | | |
| Accuracy | | 0.9661 | | |
| Macro Avg Precision | | 0.96 | | |
| Macro Avg Recall | | 0.94 | | |
| Macro Avg F1-Score | | 0.95 | | |
| Weighted Avg Precision | | 0.97 | | |
| Weighted Avg Recall | | 0.97 | | |
| Weighted Avg F1-Score | | 0.97 | | |

**Table 11: RF Results**

**Classification Report – Support Vector Machine (SVM)**

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 (Normal) | 0.96 | 0.97 | 0.97 | 9507 |
| 1 (Attack) | 0.91 | 0.87 | 0.89 | 2993 |
| **Metric** | | **Value** | | |
| Accuracy | | 0.9466 | | |
| Macro Avg Precision | | 0.93 | | |
| Macro Avg Recall | | 0.92 | | |
| Macro Avg F1-Score | | 0.93 | | |
| Weighted Avg Precision | | 0.95 | | |
| Weighted Avg Recall | | 0.95 | | |
| Weighted Avg F1-Score | | 0.95 | | |

**Table 12: SVM Results**

### 5.5.2 DL

### 5.5.3 Classification Report – Artificial Neural Network (ANN)

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 (Normal) | 0.96 | 0.98 | 0.97 | 9507 |
| 1 (Attack) | 0.93 | 0.87 | 0.90 | 2993 |

| Metric | Value |
|---|---|
| Accuracy | 0.9526 |
| Macro Avg Precision | 0.95 |
| Macro Avg Recall | 0.92 |
| Macro Avg F1-Score | 0.93 |
| Weighted Avg Precision | 0.95 |
| Weighted Avg Recall | 0.95 |
| Weighted Avg F1-Score | 0.95 |

**Table 13:ANN Results**

### 5.5.1 Classification Report – Convolutional Neural Network (CNN)

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 (Normal) | 0.95 | 0.98 | 0.97 | 9507 |
| 1 (Attack) | 0.94 | 0.85 | 0.89 | 2993 |

| Metric | Value |
|---|---|
| Accuracy | 0.9513 |
| Macro Avg Precision | 0.95 |
| Macro Avg Recall | 0.92 |
| Macro Avg F1-Score | 0.93 |
| Weighted Avg Precision | 0.95 |
| Weighted Avg Recall | 0.95 |
| Weighted Avg F1-Score | 0.95 |

**Table 14: CNN Results**

### 5.5.2 Final selected XGB and ANN

| Detail | Value |
|---|---|
| Saved Model File | xgboost_ann_hybrid_model.pkl |
| Hybrid Accuracy | 0.9646 |

**Table 15: Final selected XGB and ANN**

### 5.5.3 UI – Upload Analysis

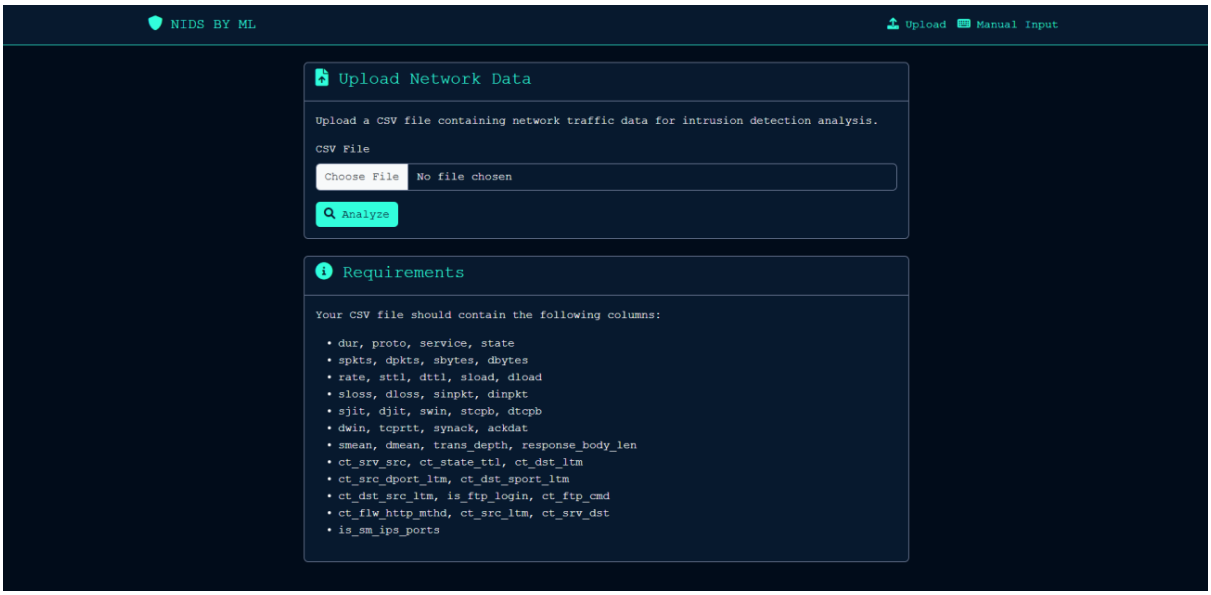



**Figure 16:** UI – Upload Analysis

**Pseudocode**
**START**

**FUNCTION run_nids_system():**
**DISPLAY 'Upload CSV File' interface**

**IF user_uploads_csv_file:**
**csv_data ← READ uploaded CSV file**

**IF validate_columns(csv_data):**
**cleaned_data ← preprocess_data(csv_data)**

**model_xgb ← LOAD "xgboost_model.pkl" model_ann ← LOAD "ann_model.pkl"**

**xgb_predictions ← model_xgb.predict(cleaned_data) ann_predictions ← model_ann.predict(cleaned_data)**

**hybrid_predictions ← COMBINE(xgb_predictions, ann_predictions)**

**results ← classify_traffic(hybrid_predictions)**
**DISPLAY results_to_user(results) ELSE:**
**DISPLAY error_message("Invalid CSV format. Required columns missing.") ELSE:**
**DISPLAY prompt("Please upload a CSV file.")**

**END FUNCTION**

**FUNCTION validate_columns(data):**
**required_columns ← [ 'dur', 'proto', 'service', 'state', 'spkts', 'dpkts', 'sbytes', 'dbytes', 'rate', 'sttl', 'dttl', 'sload', 'dload', 'sloss', 'dloss', 'sinpkt', 'dinpkt', 'sjit', 'djit', 'swin', 'stcpb', 'dtcpb', 'dwin', 'tcprtt', 'synack', 'ackdat', 'smean', 'dmean', 'trans_depth', 'response_body_len', 'ct_srv_src', 'ct_state_ttl', 'ct_dst_ltm', 'ct_src_dport_ltm', 'ct_dst_sport_ltm', 'ct_dst_src_ltm', 'is_ftp_login', 'ct_ftp_cmd', 'ct_flw_http_mthd', 'ct_src_ltm', 'ct_srv_dst', 'is_sm_ips_ports'**
**]**
**RETURN all column in required_columns exist in data.columns**

**FUNCTION preprocess_data(data):**
**FILL missing values APPLY min-max normalization SELECT top features using mutual_info & RFE RETURN processed_data**

**FUNCTION COMBINE(xgb_preds, ann_preds):**
**hybrid_output ← majority_vote(xgb_preds, ann_preds) RETURN hybrid_output**

**FUNCTION classify_traffic(predictions):**
**FOR each row in predictions:**
**IF prediction == 1:**
**LABEL as "Malicious"**
**IDENTIFY threat_type using ANN ELSE:**
**LABEL as "Normal" RETURN labeled_results**

**END**

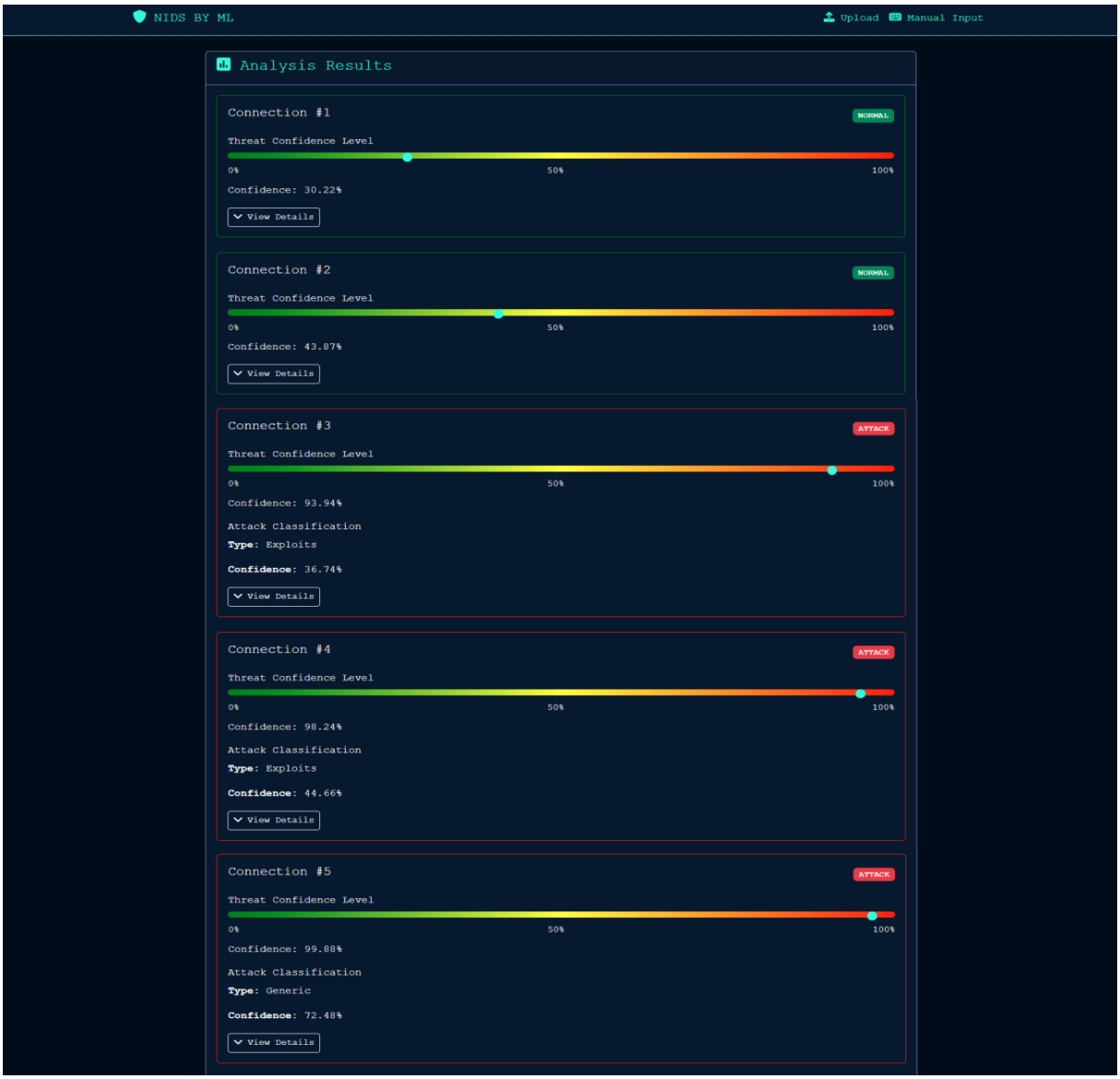



**Figure 17:** Analysis Results


**Figure 18:** NIDS of ML

## 6. Conclusion

In conclusion, the Network Intrusion Detection System (NIDS), developed using machine learning models such as XGBoost and Artificial Neural Networks (ANN), has proven to be an effective solution for detecting and classifying network intrusions. The NIDS achieves high levels of accuracy, precision, and recall, particularly with XGBoost, which outperforms other models in terms of detection performance. By leveraging the strengths of both traditional machine learning and deep learning, the hybrid model, which combines XGBoost and ANN, provides an optimal balance between speed and accuracy. The system was designed and deployed using the Flask framework and integrated with real-time data, ensuring that it is capable of making timely predictions for network security. The API design and model integration ensure that the system is flexible, scalable, and easy to maintain, making it a reliable tool for real-time intrusion detection. Overall, the NIDS demonstrates robust performance across various metrics and can provide accurate intrusion detection in dynamic network environments.

Moreover, my contributions to this project included the end-to-end development of the hybrid Network Intrusion Detection System (NIDS). I was responsible for selecting and preprocessing the UNSW-NB15 dataset, implementing data cleaning, normalization, and feature engineering techniques. I trained and evaluated multiple machine learning and deep learning models, including Random Forest, SVM, XGBoost, ANN, and CNN, and selected XGBoost and ANN for hybrid integration based on performance. Additionally, I developed the system architecture, implemented the hybrid model, and integrated it into a Flask-based web interface for real-time CSV input, analysis, and visualization. I also conducted performance evaluations and saved the final hybrid model for deployment.

## Limitation and Future Work

### Limitation

Despite the effectiveness of the proposed hybrid NIDS, several limitations were observed. Firstly, the system is trained and validated solely on the UNSW-NB15 dataset, which, while comprehensive, may not fully capture the diversity of real-world or zero-day attacks. Secondly, the deep learning component (ANN) introduces additional computational overhead, which could impact real-time performance on lowresource systems. Thirdly, the model may exhibit reduced sensitivity to minority attack classes due to class imbalance, potentially affecting detection rates for rare threats. Lastly, the system currently supports only batch-based CSV input rather than real-time packet capture, limiting its deployment scope in dynamic network environments.

### Future Work

Despite the strong foundation of network security that the NIDS offers, there are also multiple ways in which it can be improved. An area for improvement is the incorporation of more machine learning models or use their ensemble to increase the detection accuracy by exploiting the benefit of diverse models. Besides, advanced defines mechanism including adversarial training and input sanitization can be incorporated to enhance the system's adversarial read against manipulations and improve adversarial attack read. Another key area for improvement of the system is the real time scalability in the systems with large scale traffic. While the system must achieve high-traffic volume handling without compromising prediction speed or accuracy, there is no lack of motivation for wide deployment. Additionally, considering automated dynamical retraining of the system's model according to new attack patterns can assist to keep the system somehow up to date with the latest attack threats. For accessibility, flexibility, and to deal with varying traffic loads of the system, we could also explore the possibility to deploy it in the cloud. Tracing focus

across these areas, the NIDS can be modified into still more powerful and adaptive network security tool of the future.


## Acknowledgement

First and foremost, I would like to express my sincere gratitude to my supervisor Dr. Dharshana Kasthurirathna for their invaluable guidance, support, and encouragement throughout the course of this research. Their expertise and insightful feedback have been instrumental in shaping the direction and quality of this work.

I would also like to extend my appreciation to the academic staff, for providing a supportive and resourceful learning environment. Special thanks go to the technical team and lab coordinators who facilitated access to the necessary computing resources and tools required for model development and testing.

I am deeply thankful to my family and friends for their constant support, motivation, and understanding throughout my academic journey. Their encouragement helped me remain focused and resilient, especially during challenging times.

Finally, I would like to acknowledge the creators of the UNSW-NB15 dataset and the broader research community whose contributions have significantly advanced the field of network security and machine learning, and whose work laid the foundation for this study.

This thesis would not have been possible without the collective support and contributions of all the above.


## References

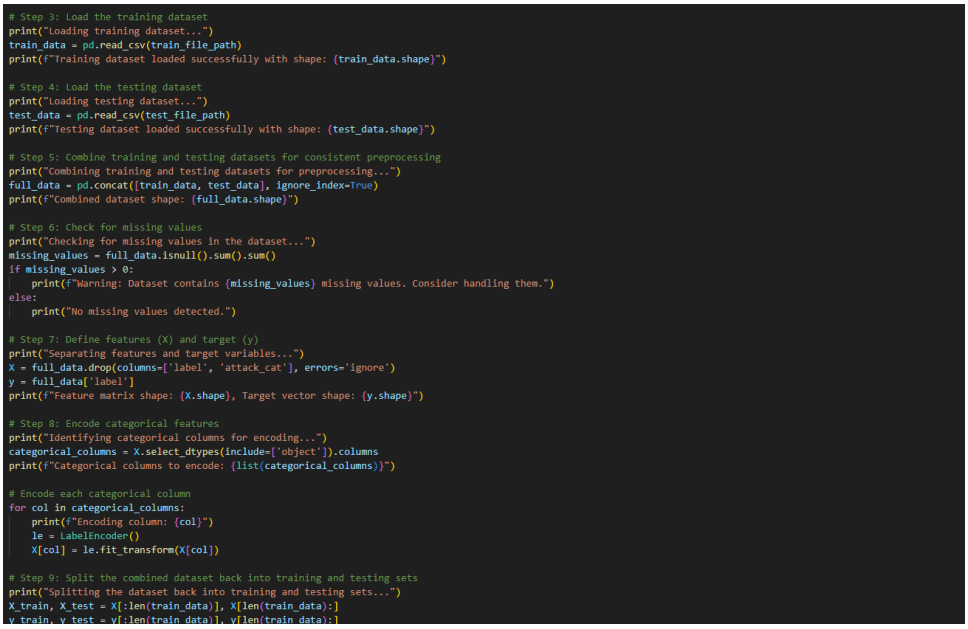

1. Borky, J. M., & Bradley, T. H. (2018). *Effective model-based systems engineering*. Springer.
2. Canavan, J. E. (2001). *Fundamentals of network security*. Artech House.
3. R. Anderson, SECURITY ENGINEERING : a Guide to Building Dependable Distributed systems. S.L.: John Wiley & Sons, 2020.
4. B. Potter and B. Fleck, 802.11 Security. Sebastopol, Calif. ; Farnham: O'reilly, 2003.
5. R. A. Grimes, Fighting Phishing. John Wiley & Sons, 2024. Davidoff, S., Durrin, M., & Sprenger, K. (2022). Ransomware and Cyber Extortion: Response and Prevention. Addison-Wesley Professional.
6. Thomas, S. (2008). *Using Automated Fix Generation to Mitigate SQL Injection Vulnerabilities*. VDM Verlag.
7. Ablon, L., & Bogart, A. (2017). *Zero days, thousands of nights: The life and times of zero-day vulnerabilities and their exploits.* Rand Corporation.
8. Cole, E. (2012). *Advanced persistent threat: understanding the danger and how to protect your organization.* Newnes.
9. Bimal Kumar Mishra and Jose R.C. Piqueira, *Understanding Cyber Threats and Attacks*. Nova Science Publishers, 2020.
10. Abomhara, M., & Køien, G. M. (2015). Cyber security and the internet of things: vulnerabilities, threats, intruders and attacks. *Journal of Cyber Security and Mobility*, 65-88.
11. A. A. Ghorbani, W. Lu, Mahbod Tavallaee, and Springerlink (Online Service, *Network Intrusion Detection and Prevention : Concepts and Techniques*. 出版商： Boston, Ma: Springer Us, 2010.
12. Sanders, C., & Smith, J. (2013). *Applied network security monitoring: collection, detection, and analysis*. Elsevier.
13. Gerardus Blokdyk, *Anomaly Based Intrusion Detection System a Complete Guide - 2020 Edition.* 5starcooks, 2020.
14. Ganapathi, P., & Shanmugapriya, D. (Eds.). (2019). *Handbook of research on machine and deep learning applications for cyber security*. IGI Global.
15. F. Hu and X. Hei, *AI, Machine Learning and Deep Learning*. CRC Press, 2023.
16. Assegie, T. A., Salau, A. O., Chhabra, G., Kaushik, K., & Braide, S. L. (2024, May). Evaluation of random forest and support vector machine models in educational data mining. In *2024 2nd International Conference on Advancement in Computation & Computer Technologies (InCACCT)* (pp. 131-135).
17. T. Sutanto, M. R. Aditya, H. Budiman, M. Rezqy. Noor Ridha, U. Syapotro, and N. Azijah, "Comparison of Logistic Regression, Random Forest, SVM, KNN Algorithm for Water Quality Classification Based on Contaminant Parameters," *INTI Journal,* vol. 2022, no. 1, Nov. 2024.
18. Wikipedia Contributors, "Intrusion Detection System," *Wikipedia, Mar. 05*, 2025.
19. Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572.*
20. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., & Swami, A. (2016, March). The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)* (pp. 372-387).
21. Islam, S., & Falcarin, P. (2011, September). Measuring security requirements for software security. In *2011 IEEE 10th International Conference on Cybernetic Intelligent Systems (CIS)* (pp. 70-75). IEEE.
22. Engen, V., Vincent, J., & Phalp, K. (2011). Exploring discrepancies in findings obtained with the KDD Cup'99 data set. *Intelligent Data Analysis, 15*(2), 251-276.
23. Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp, 1*(2018), 108-116.
24. Lihobabina, A., & Menshikh, V. (2023, November). Model and Numerical Method of Optimization of Selection of Training Programs for Specialists. In *2023 5th International Conference on Control Systems, Mathematical Modeling, Automation and Energy Efficiency (SUMMA)* (pp. 447-450). IEEE.
25. Sano, H., Utsunomiya, R., Senda, T., Tani, K., & Yamada, T. (2024, September). Performance Study of Surrogate Models for Large-Scale Optimization. In *2024 International Conference on Electrical Machines (ICEM)* (pp. 1-6). IEEE.
26. Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *The journal of machine learning*

*research, 13*(1), 281-305.
27. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research, 15*(1), 1929-1958.
28. Sekar, R., Bendre, M., Dhurjati, D., & Bollineni, P. (2000, May). A fast automaton-based method for detecting anomalous program behaviors. In *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001* (pp. 144-155). IEEE.
29. Alazab, M., & Tang, M. (Eds.). (2019). *Deep learning applications for cyber security.* Springer.
30. System (NIDS) Application," *Anwendungen Und Konzepte Der Wirtschaftsinformatik,* no. 6, pp. 10–19, Nov. 2017.
31. Elly, B., Lousy, N., & Glow, F. (2024). Advancements in Deep Learning: Enhancing AI Capabilities.
32. M. T. MAHTAB et al., "Forecasting Bangladeshi Garlic Price Using DL and ML Hybrid Approach," *International Journal of Research Publication and Reviews*, vol. 5, no. 12, pp. 3200– 3208, Dec. 2024.
33. Tama, B. A., & Lim, S. (2021). Ensemble learning for intrusion detection systems: A systematic mapping study and cross-benchmark evaluation. *Computer Science Review, 39*, 100357.
34. Hore, S., Ghadermazi, J., Shah, A., & Bastian, N. D. (2024). A sequential deep learning framework for a robust and resilient network intrusion detection system. *Computers & Security, 144,* 103928.
35. Atadoga, A., Sodiya, E. O., Umoga, U. J., & Amoo, O. O. (2024). A comprehensive review of machine learning's role in enhancing network security and threat detection. *World Journal of Advanced Research and Reviews, 21*(2), 877-886.2024.
36. Khanal, B., Kumar, C., & Ansari, M. S. A. (2025). Real-Time Anomaly Detection Framework to Mitigate Emerging Threats in Software Defined Networks. Journal of Network and Systems Management, 33(2), 26.
37. Vinayakumar, R., Alazab, M., Soman, K. P., Poornachandran, P., Al-Nemrat, A., & Venkatraman, S. (2019). Deep learning approach for intelligent intrusion detection system. IEEE access, 7, 41525-41550.
38. Rustam, F., Raza, A., Qasim, M., Posa, S. K., & Jurcut, A. D. (2024). A novel approach for real-time server-based attack detection using meta-learning. IEEE Access.
39. Al-Shurbaji, T., Anbar, M., Manickam, S., Hasbullah, I. H., ALfriehate, N., Alabsi, B. A., ... & Hashim, H. (2025). Deep Learning-Based Intrusion Detection System For Detecting IoT Botnet Attacks: A Review. IEEE Access.
40. Khan, M. A., & Kim, Y. (2021). Deep Learning-Based Hybrid Intelligent Intrusion Detection System. Computers, Materials & Continua, 68(1).
41. Zehra, S., Faseeha, U., Syed, H. J., Samad, F., Ibrahim, A. O., Abulfaraj, A. W., & Nagmeldin, W. (2023). Machine learning-based anomaly detection in NFV: A comprehensive survey. Sensors, 23(11), 5340.
42. Ahmed, U., Nazir, M., Sarwar, A., Ali, T., Aggoune, E. H. M., Shahzad, T., & Khan, M. A. (2025). Signature-based intrusion detection using machine learning and deep learning approaches empowered with fuzzy clustering. Scientific Reports, 15(1), 1726.
43. Sajid, M., Malik, K. R., Almogren, A., Malik, T. S., Khan, A. H., Tanveer, J., & Rehman, A. U. (2024). Enhancing intrusion detection: a hybrid machine and deep learning approach. Journal of Cloud Computing, 13(1), 123.
44. Mamatha, P., Balaji, S., & Anuraghav, S. S. (2025). Development of Hybrid Intrusion Detection System Leveraging Ensemble Stacked Feature Selectors and Learning Classifiers to Mitigate the DoS Attacks. International Journal of Computational Intelligence Systems, 18(1), 1-18.
45. Ahnaf Akif, M., Butun, I., Williams, A., & Mahgoub, I. (2025). Hybrid Machine Learning Models for Intrusion Detection in IoT: Leveraging a Real-World IoT Dataset. arXiv e-prints, arXiv-2502.
46. Kandasamy, V., & Roseline, A. A. (2025). Harnessing advanced hybrid deep learning model for real-time detection and prevention of man-in-the-middle cyber attacks. Scientific Reports, 15(1), 1697.
47. Khraisat, A., Gondal, I., Vamplew, P., & Kamruzzaman, J. (2019). Survey of intrusion detection systems: techniques, datasets and challenges. Cybersecurity, 2(1), 1-22.
48. Vinayakumar, R., Alazab, M., Soman, K. P., Poornachandran, P., Al-Nemrat, A., & Venkatraman, S. (2019). Deep learning approach for intelligent intrusion detection system. IEEE access, 7, 41525-41550.
49. Ali, Z. H., & Burhan, A. M. (2023). Hybrid machine learning approach for construction cost estimation: An evaluation of extreme gradient boosting model. Asian Journal of Civil Engineering, 24(7), 2427-2442.
50. why, "why boosting method is sensitive to outliers," Cross Validated, Mar. 03, 2015.
51. ncrefe, "Robust Algorithms to Outliers," Medium, Aug. 16, 2023. "Frequently Asked Questions — xgboost 1.6.1 documentation," xgboost.readthedocs.io.
52. GeeksforGeeks, "Handling Missing Data with KNN Imputer," GeeksforGeeks, Aug. 13, 2024.
53. Rusdah, D. A., & Murfi, H. (2020). XGBoost in handling missing values for life insurance risk prediction. SN Applied Sciences, 2(8), 1336.
54. Zheng, Q., Yu, C., Cao, J., Xu, Y., Xing, Q., & Jin, Y. (2024, August). Advanced payment security system: xgboost, lightgbm and smote integrated. In 2024 IEEE International Conference on Metaverse Computing, Networking, and Applications (MetaCom) (pp. 336-342). IEEE.
55. Dai, Z., Por, L. Y., Chen, Y. L., Yang, J., Ku, C. S., Alizadehsani, R., & Pławiak, P. (2024). An intrusion detection model to detect zero-day attacks in unseen data using machine learning. PloS one, 19(9), e0308469.
56. Farooqi, A. H., Akhtar, S., Rahman, H., Sadiq, T., & Abbass, W. (2023). Enhancing network intrusion detection using an ensemble voting classifier for internet of things. Sensors, 24(1), 127.
57. J. Lee, "Does XGBoost need standardization or normalization?," Stack Overflow, Apr. 18, 2022.

58. M. Filho, “Does XGBoost Need Feature Scaling Or Normalization?,” Forecastegy.com, Dec. 30, 2022.
59. Farooqi, A. H., Akhtar, S., Rahman, H., Sadiq, T., & Abbass, W. (2023). Enhancing network intrusion detection using an ensemble voting classifier for internet of things. Sensors, 24(1), 127.
60. Why, “Why does feature scaling improve the convergence speed for gradient descent?,” Data Science Stack Exchange, Jul. 14, 2019.
61. Fan, Z., & You, Z. (2024). Research on network intrusion detection based on xgboost algorithm and multiple machine learning algorithms. Theoretical and Natural Science, 31(1), 161-166.

## Appendix 1: Codes Snip

### Logistic_Regression_Model



```
# Step 3: Load the training dataset
print("Loading training dataset...")
train_data = pd.read_csv(train_file_path)
print(f"Training dataset loaded successfully with shape: {train_data.shape}")

# Step 4: Load the testing dataset
print("Loading testing dataset...")
test_data = pd.read_csv(test_file_path)
print(f"Testing dataset loaded successfully with shape: {test_data.shape}")

# Step 5: Combine training and testing datasets for consistent preprocessing
print("Combining training and testing datasets for preprocessing...")
full_data = pd.concat([train_data, test_data], ignore_index=True)
print(f"Combined dataset shape: {full_data.shape}")

# Step 6: Check for missing values
print("Checking for missing values in the dataset...")
missing_values = full_data.isnull().sum().sum()
if missing_values > 0:
    print(f"Warning: Dataset contains {missing_values} missing values. Consider handling them.")
else:
    print("No missing values detected.")

# Step 7: Define features (X) and target (y)
print("Separating features and target variables...")
X = full_data.drop(columns=['label', 'attack_cat'], errors='ignore')
y = full_data['label']
print(f"Feature matrix shape: {X.shape}, Target vector shape: {y.shape}")

# Step 8: Encode categorical features
print("Identifying categorical columns for encoding...")
categorical_columns = X.select_dtypes(include=['object']).columns
print(f"Categorical columns to encode: {list(categorical_columns)}")

# Encode each categorical column
for col in categorical_columns:
    print(f"Encoding column: {col}")
    le = LabelEncoder()
    X[col] = le.fit_transform(X[col])

# Step 9: Split the combined dataset back into training and testing sets
print("Splitting the dataset back into training and testing sets...")
X_train, X_test = X[:len(train_data)], X[len(train_data):]
y_train, y_test = y[:len(train_data)], y[len(train_data):]
```

### Random_Forest_Model

```
# Step 9: Split the combined dataset back into training and testing sets
print("Splitting the dataset back into training and testing sets...")
X_train, X_test = X[:len(train_data)], X[len(train_data):]
y_train, y_test = y[:len(train_data)], y[len(train_data):]
print(f"Training set shape: {X_train.shape}, Testing set shape: {X_test.shape}")

# Step 10: Scale the feature data
print("Scaling feature data using StandardScaler...")
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
print("Feature scaling completed.")

# Step 11: Initialize and train the Random Forest Classifier
print("Initializing Random Forest Classifier...")
model = RandomForestClassifier(n_estimators=100, random_state=42)
print("Training the Random Forest Classifier...")
model.fit(X_train, y_train)
print("Model training completed.")

# Step 12: Make predictions on the testing data
print("Making predictions on the testing set...")
y_pred = model.predict(X_test)

# Step 13: Evaluate model performance
print("Evaluating model performance...")
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:\n", classification_report(y_test, y_pred))

# Step 14: Save the model and scaler for future use (optional)
import joblib
print("Saving the trained model and scaler...")
joblib.dump(model, "random_forest_model.pkl")
joblib.dump(scaler, "scaler.pkl")
print("Model and scaler saved successfully.")

print("Script execution completed.")
```

## SVM_Model

```
    X[col] = le.fit_transform(X[col])

# Step 9: Split combined dataset back into training and testing sets
print("Splitting data back into training and testing sets...")
X_train, X_test = X[:len(train_data)], X[len(train_data):]
y_train, y_test = y[:len(train_data)], y[len(train_data):]
print(f"Training data shape: {X_train.shape}, Testing data shape: {X_test.shape}")

# Step 10: Scale the data using StandardScaler
print("Scaling feature data with StandardScaler...")
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
print("Feature scaling completed.")

# Step 11: Initialize and train the Support Vector Machine (SVM) model
print("Initializing Support Vector Machine (SVM) model with RBF kernel...")
model = SVC(kernel='rbf', random_state=42)
print("Training the SVM model...")
model.fit(X_train, y_train)
print("SVM model training completed.")

# Step 12: Make predictions on the testing data
print("Making predictions on the testing set...")
y_pred = model.predict(X_test)

# Step 13: Evaluate the model's performance
print("Evaluating model performance...")
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:\n", classification_report(y_test, y_pred))

# Step 14: Save the trained model and scaler for future use (optional)
import joblib
print("Saving the trained SVM model and scaler...")
joblib.dump(model, "svm_model.pkl")
joblib.dump(scaler, "scaler.pkl")
print("Model and scaler saved successfully.")
```

## Xgboost_Model

```
# Encode each categorical column using LabelEncoder
for col in categorical_columns:
    print(f"Encoding column: {col}")
    le = LabelEncoder()
    X[col] = le.fit_transform(X[col])

# Step 9: Split the combined dataset back into training and testing sets
print("Splitting data back into training and testing sets...")
X_train, X_test = X[:len(train_data)], X[len(train_data):]
y_train, y_test = y[:len(train_data)], y[len(train_data):]
print(f"Training features shape: {X_train.shape}, Testing features shape: {X_test.shape}")

# Step 10: Scale the feature data
print("Scaling features using StandardScaler...")
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
print("Feature scaling completed.")

# Step 11: Initialize the XGBoost Classifier
print("Initializing XGBoost Classifier...")
model = XGBClassifier(n_estimators=100, max_depth=6, learning_rate=0.1, random_state=42)
print("Training the XGBoost Classifier...")
model.fit(X_train, y_train)
print("Model training completed.")

# Step 12: Make predictions on the testing dataset
print("Making predictions on the testing set...")
y_pred = model.predict(X_test)

# Step 13: Evaluate the model's performance
print("Evaluating model performance...")
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:\n", classification_report(y_test, y_pred))

# Step 14: Save the trained model and scaler for future use (optional)
import joblib
print("Saving the trained model and scaler...")
joblib.dump(model, "xgboost_model.pkl")
joblib.dump(scaler, "scaler.pkl")
print("Model and scaler saved successfully.")
```

## Compare

```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import os

# Step 1: Define file paths for datasets
train_file_path = "Dataset/UNSW_NB15_training-set.csv"
test_file_path = "Dataset/UNSW_NB15_testing-set.csv"

# Step 2: Verify if the dataset files exist
if not os.path.exists(train_file_path) or not os.path.exists(test_file_path):
    raise FileNotFoundError("One or both dataset files are missing. Please check the file paths.")

# Step 3: Load the datasets
print("Loading datasets...")
train_data = pd.read_csv(train_file_path)
test_data = pd.read_csv(test_file_path)
print(f"Training dataset shape: {train_data.shape}")
print(f"Testing dataset shape: {test_data.shape}")

# Step 4: Combine datasets for uniform preprocessing
print("Combining training and testing datasets...")
full_data = pd.concat([train_data, test_data], ignore_index=True)
print(f"Combined dataset shape: {full_data.shape}")

# Step 5: Separate features and target variable
print("Separating features (X) and target variable (y)...")
X = full_data.drop(columns=['label', 'attack_cat'], errors='ignore')  # Drop unused columns
y = full_data['label']
print(f"Features shape: {X.shape}, Target shape: {y.shape}")

# Step 6: Identify and encode categorical features
print("Identifying and encoding categorical columns...")
categorical_columns = X.select_dtypes(include=['object']).columns
print(f"Categorical columns to encode: {list(categorical_columns)}")
for col in categorical_columns:
    print(f"Encoding column: {col}")
    le = LabelEncoder()
    X[col] = le.fit_transform(X[col])
```

```python
# Step 7: Split combined data back into training and testing sets
print("Splitting data back into training and testing sets...")
X_train, X_test = X[:len(train_data)], X[len(train_data):]
y_train, y_test = y[:len(train_data)], y[len(train_data):]
print(f"Training data shape: {X_train.shape}, Testing data shape: {X_test.shape}")

# Step 8: Scale the data using StandardScaler
print("Scaling feature data using StandardScaler...")
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
print("Feature scaling completed.")

# Step 9: Initialize machine learning models
print("Initializing machine learning models...")
models = {
    "Logistic Regression": LogisticRegression(max_iter=500),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "SVM": SVC(kernel='rbf', random_state=42),
    "XGBoost": XGBClassifier(n_estimators=100, max_depth=6, learning_rate=0.1, random_state=42),
}

# Step 10: Train and evaluate models
print("Training and evaluating models...")
accuracies = {}
for name, model in models.items():
    print(f"Training {name}...")
    model.fit(X_train, y_train)  # Train the model
    print(f"{name} trained successfully.")

    print(f"Making predictions with {name}...")
    y_pred = model.predict(X_test)  # Predict test data

    accuracy = accuracy_score(y_test, y_pred)  # Calculate accuracy
    accuracies[name] = accuracy
    print(f"{name} Accuracy: {accuracy:.4f}")

# Step 11: Visualize model accuracies
print("Visualizing model accuracies...")
plt.figure(figsize=(10, 6))
plt.bar(accuracies.keys(), accuracies.values(), color=['blue', 'green', 'orange', 'red'])
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.title('Comparison of Model Accuracies')
plt.ylim(0, 1)  # Set y-axis range for accuracies
```

```
# Step 11: Visualize model accuracies
print("Visualizing model accuracies...")
plt.figure(figsize=(10, 6))
plt.bar(accuracies.keys(), accuracies.values(), color=['blue', 'green', 'orange', 'red'])
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.title('Comparison of Model Accuracies')
plt.ylim(0, 1)  # Set y-axis range for accuracies
plt.xticks(rotation=15)
plt.tight_layout()
plt.show()
```

**Appendix 2: Research Questions and Findings**

i. Accordingly, how can the combination of ML and DL in a hybrid model empower network intrusion detection towards an improved accuracy with a more efficient computation?

From this point onward, the hybrid model that integrates Machine Learning (ML) and Deep Learning (DL) models significantly ameliorates accuracy and efficiency of network intrusion detection systems (NIDS). XGBoost is a powerful ML algorithm which is geared towards helping certain decision making processes and classification, can be deployed with high performance attack signature detection with fast and efficient processing. However, Artificial Neural Networks (ANN) based on deep learning algorithm are capable to learn complex, non linear relationship in the data and are useful for detecting unknown threats using anomaly detection.

The hybrid model combines this approach of XGBoost with the adaptability of ANN such that it can classify known attacks with speed and accuracy as well as identify new threats. A combination of these techniques yields a full detection capability with good low false positive rate and high detection recall even against dynamic and evolving network traffic. Moreover, the implementation of the hybrid method allows achieving a compromise between the model precision and timing responses, hence it's efficient in the high-volume network environments for real time processing.

ii. Which detection method could be the combination between anomaly and signature based so that known and unknown network attacks could be detected?

By combining the anomaly-based detection method with signature based detection method, it guarantees detection on the limitation of each method. However, the signature-based detection is highly effective to identify the known threat by using the predefined patterns for malicious activity. The fast and reliable detection of common attacks (such as DDoS, SQL injections and malware) are provided however, the detection of new or unseen attacks is limited.

Since the capability of signature based detection does not address this limitation, anomaly based detection methodology is used to identify deviations from established normal network behavior in order to detect unknown threat. This method has its utility in zero-day attacks as well as APTs; APTs dont have signatures in the predefined database.

We design a hybrid approach realized by the combination of XGBoost (signature detection) and ANN (anomaly detection) in the proposed NIDS. Whereas, the ANN model learns complex, non linear relationship among the traffic data and recognize source of new and complex attack patterns. The system combines both methods to make sure that it identifies known and unknown attacks at the similar time in order not to be vulnerable to the possibility of missing one by the other.

iii. In high—volume networks, which ML or DL algorithms, after fine tuning, give the best Detection of most of the real—time threats?

Finally, to detect real time threat in high vulnerable networks, after fine tuning, XGBoost Machine Learning based and ANN Deep Learning based models perform best.

XGBoost has excellent ability to predict, high efficiency to run and scalability. Due to its fast action and the capacity to deal with huge datasets packed up with various features, it renders particularly applicable for the real-time network intrusion detection. Hyperparameters like learning rate, max depth, and n_estimators are fine tuned if XGBoost is intended to have high accuracy while keeping working with large volumes of incoming network traffic with low latency.

ANN is good at modeling complex non linear relationships in the network traffic data because of its deep learning capabilities. However, it particularly excels at observing and observing anomalous attacks that are quite unique and non signature alike. Secondly, parameters are fine tuned like number of layers, neurons per layer and learning rate for ensuring that ANN process the amount of real time traffic while repeating high precision and recall.

XGBoost and ANN together form a hybrid model that is extremely efficient, accurate and guarantees superior performance for real-time

threat detection at high volume network environments.

iv. What is the best type of a web-based interface that would improve usability of the Intrusion Detection System and monitoring for the network security team?
The development of a web based interface allows for significant improvement in usability and monitoring of an Intrusion Detection System (IDS) by setting up a central, accessible, web based interface that the network security teams can access to interact with the system in real time. Security teams can work with a web interface.

1.Instead of monitoring Network Traffic, the interface gives a security analyst visual experience on real time traffic and attack trends and system overall performance to know the updates on the network's security state of health.
2.Security teams can view the access alerts and notifications regarding alerts generated by the NIDS for detected threats including attack type, severity and the affected systems. These notifications allow teams to decide their response priorities for priority incidents.
3.Web Interface: Connect with the Model Outputs: The model's predictions such like the confidence level for a detected intrusion can be presented on the web interface for security analysts to base their decisions on.
4.System Configurations Manage: Through the interface, the parameters for detection, the models can be updated, and user access managed, so that the system is kept optimized for the evolving network environment and threat landscape.
5.Web Interface: Security teams are provided with the possibility to generate detailed reports about detected threats, network performance, and model performance which support compliance and audit requirements.
Overall, a web-based interface enhances the features of accessibility, usability and of management of the NIDS, so that the network security team can monitor, manage and act to threats in real-time and as a result the system increases its capabilities of protection to the core network infrastructures.

**Appendix 3: Output**

```
XGB Results:
Accuracy: 0.9668
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      9507
           1       0.96      0.90      0.93      2993

    accuracy                           0.97     12500
   macro avg       0.96      0.94      0.95     12500
weighted avg       0.97      0.97      0.97     12500
```

```
RF Results:
Accuracy: 0.9661
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      9507
           1       0.96      0.90      0.93      2993

    accuracy                           0.97     12500
   macro avg       0.96      0.94      0.95     12500
weighted avg       0.97      0.97      0.97     12500
```

```
SVM Results:
Accuracy: 0.9466
              precision    recall  f1-score   support

           0       0.96      0.97      0.97      9507
           1       0.91      0.87      0.89      2993

    accuracy                           0.95     12500
   macro avg       0.93      0.92      0.93     12500
weighted avg       0.95      0.95      0.95     12500
```

```
ANN Results:
Accuracy: 0.9526
              precision    recall  f1-score   support

           0       0.96      0.98      0.97      9507
           1       0.93      0.87      0.90      2993

    accuracy                           0.95     12500
   macro avg       0.95      0.92      0.93     12500
weighted avg       0.95      0.95      0.95     12500
```

```
CNN Results:
Accuracy: 0.9513
              precision    recall  f1-score   support

           0       0.95      0.98      0.97      9507
           1       0.94      0.85      0.89      2993

    accuracy                           0.95     12500
   macro avg       0.95      0.92      0.93     12500
weighted avg       0.95      0.95      0.95     12500
```

```
Hybrid Model Performance:

Saving models...
Model saved to xgboost_ann_hybrid_model.pkl
Hybrid Accuracy: 0.9646
```