Attack Detection and Prevention Using Machine Learning and Deep Learning Techniques

MSc Research Project
Master of Science in Cyber Security

Achu Abraham George
Student ID: X23127872

School of Computing
National College of Ireland

Supervisor: Diego Lugones

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	…….Achu Abraham George………………………………………………………………		
Student ID:	……X23127872……………………………………………………………………………..……		
Programme:	…MS in Cyber Security……………………………	**Year:**	…2024-2025….
Module:	…………Research Project……………………………………………………………….………		
Supervisor:	………………Diego Lugones……………………………………………………….………		
Submission Due Date:	……………29/01/2025………………………………………………………………….………		
Project Title:	……Attack Detection and Prevention using Machine Learning and Deep Learning Techniques……….………		
Word Count:	…………6993…………………	**Page Count**	…………………24……………………..……..

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……Achu Abraham George…………………………………………………

Date: ……28/01/2025………………………………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Attack Detection and Prevention Using Machine Learning and Deep Learning Techniques

Achu Abraham George

X23127872

Abstract

This research seeks to offer a solution to the growing development of sound Intrusion Detection Systems(IDSs) to suit modern emerging threats such as sniffing and port scanning. A combined IDS is suggested using K-Nearest Neighbors (KNN) as well as Multilayer Perceptron (MLP) schemes in order to recognize network traffic as Allow, Deny, or Drop. When using the Internet Firewall dataset, feature scaling and feature selection transformations were performed for the sake of improving the models. With regards to the MLP model, the validation accuracy was 99.83%, and for the KNN was only 95.29% implying the benefits of both models. Furthermore, this study focuses on the network probing tools and machine learning as key approaches to sniffing detection. Green, clean, and user-friendly interface capabilities allow users to make predictions in real-time complete with probability scores. This work demonstrates how a machine learning approach can be integrated with deep learning to improve the level of intrusion detection for networks and offers a practical way towards secure networks in constantly evolving threat landscapes. The contributions fill the gap between the theoretical developments and systems to improve the effectiveness of adaptive and intelligent cybersecurity systems.

1 Introduction

Digital systems are advancing rapidly; and, increasingly, the networked infrastructure that undergirds them is exposed to the risk of cyber attack. With increasing sophistication and variety of these attacks, especially in finance and healthcare sectors, IDS development is needed to help protect sensitive data and critical service continuity. Existing approaches of cybersecurity are not able to detect newer types of attack, and thus advanced machine learning (ML) and deep learning (DL) approaches are required. The application of KNN and MLP models for building a robust IDS which will classify the network traffic into benign or malicious is studied in this thesis. The goal of the research is to investigate how the combination of these two models can increase classification accuracy and generalize across different types of attacks such as suspicious port scanning.

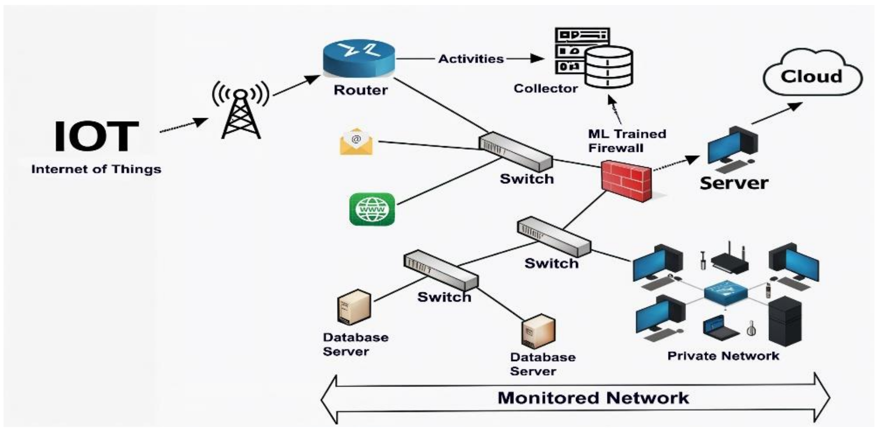


Figure 1: Real-Time Scenario of Client-Server Cyber-Security Framework

Figure 1 illustrates a real time view of the proposed Client Server Cybersecurity Framework including IoT devices, routers, switches and servers. This work uses Machine Learning (ML) techniques, specifically an ML trained firewall to improve network protection. The diagram shows how data of IoT devices flows to the cloud through multiple security layers such as routers, switches and a database server. The ML-trained firewall is a critical security component that analyses the network traffic and assigns it as benign or suspicious by learned pattern. This representation shows how this framework enables real time threats detection and prevention, which bares the importance of machine learning in securing network.

The research question driving this work is

How effective are machine learning and deep learning algorithms, specifically KNN and MLP, in detecting and preventing network-based attacks such as suspicious port scanning within a real-time client-server architecture integrated with cloud data sharing?

This project aims at creating an IDS based on KNN and MLP models to detect malicious network traffic. It integrates real-time attack detection within a cloud based file sharing and a dynamic prevention mechanism that blocklists clients sending malicious traffic and alerts the administrator. This study adds to the growing body of literature on hybrid intrusion detection systems by showing how the synergistic capabilities of the KNN and MLP models can be used to handle complex and diverse attack scenarios. The report is structured as follows: In Section 2 the related work is reviewed, focusing on IDS in cybersecurity; Section 3 describes the research methodology; Section 4 describes the system design; Section 5 describes the implementation details; Section 6 discusses evaluation results, inference and Front-end; and Section 7 concludes with key findings and future research directions.

2 Related works

This paper reviews different IDS frameworks, which presents their strengths, weaknesses, and scalability issues when integrated into real time cloud based systems for network based attack detection.

Network Intrusion and Port Scan Detection

The work entitled ‘Improving Port Scan Cybersecurity Risks Detection Using Features Selection Techniques with ML Algorithms’ was conducted and written by Rami Shehab et al. (2024). The research uses feature selection methods like Ant Colony Optimization (ACO), Genetic Algorithm (GA), and Gray Wolf Optimization (GWO) with the aim of increasing an identification of port scan attack. The study proposes the hybrid models based on the feature selection and the ML classifiers such as ACO+SVM, GA+KNN and GWO+SVM. Therefore, it can be concluded that proposed GWO+SVM outperforms all the other models with highest accuracy, precision, recall and F1 score and classifying the instances using only twelve features. The models were able to reach over 97% of accuracy across the evaluations proving its efficiency in identifying port scan attacks. Additionally, as observed from the predictive results, GWO+SVM provided an efficient solution between minimizing complexity and achieving high detection accuracy.

The study proposed a novel sniffing detection scheme leveraging network probing tools (ping, curl) and machine learning that can detect sniffing hosts with NICs in promiscuous mode with 99% accuracy. Differentiating between malicious scans and defensive ones has been a problem and several port scan detection methods have been reviewed by Bhuyan et al. (2011). They showed that combining data mining techniques with threshold based techniques resulted in scalability, robustness and low false positive rates. It was highlighted in both studies that adaptive detection strategies are required, with further directions incorporating the use of virtualization environments, SDN, and modern network security advances.

Machine Learning and Deep Learning for Intrusion Detection Systems

Ahmad et al. (2021) and Ahmed et al. (2022) also presented framework for Network Intrusion Detection Systems (NIDS) using classifiers such as Random Forest (RF), Decision Trees (DT) and Artificial Neural Networks (ANN). Preprocessing techniques SMOTE (Synthetic Minority Over-sampling Technique) and Principal Component Analysis (PCA) were applied in both studies to improve performance. Although these approaches are capable of dealing with problems such as class imbalance and dimensionality reduction, the studies do not explore in depth the scalability of their systems in large scale networks. However, this is a critical limitation, as in real world scenarios, we often need systems that scale easily to handle the larger volume of traffic and data. UNSW-NB15 and NSL-KDD datasets were used by Ashiku and Dagli (2021), and Wu et al. (2022) to explore deep learning based IDS. We show that their models, such as enhanced Random Forest and hybrid models that combine SMOTE with K-means clustering, are also able to detect network anomalies with high accuracy. However, a problem common in these works is the accuracy vs computational efficiency trade off. While these models are very successful at dealing with large datasets, they may not work so well in real time environments where speed and efficiency are key. Furthermore, their suitability for cloud based and distributed architectures is not fully explored, leaving a gap in their suitability to such environments. An intelligent IDS that reduces computational complexity using optimal feature selection was introduced by Bouke et al. (2022). However, this approach improves efficiency, but the study does not show how the system performs in dynamic and heterogeneous network environments. According to Al-Shareeda et al. (2023), ML and DL techniques used for DDoS attacks detection, DL methods are better than ML due to their capability to deal with large datasets. Nevertheless, it is worth further discussion of the tradeoffs between DL and traditional ML methods and the associated resource utilization and adaptability among varying attack types.

Both traditional machine learning and deep learning algorithms have shown to increase the accuracy of IDS, according to Azizan et al. (2021) and Garcia and Blandon (2022). However, neither of these studies address the scalability and adaptability of these models for real time and cloud based use cases. Although Shang (2024) presented a general survey on the application of machine learning in cybersecurity, there is no work that provides a detailed focus on real time network based attack detection, such as port scan — an important issue in the current research area.

IoT Security and Firewall Systems

In Alsoufi et al. (2021), deep learning-based anomaly detection systems for IoT environments were analyzed in respect of zero-day attacks detection. The results of their model were promising, but the scalability of their model in large IoT networks is not clear. Gupta et al. (2017), Anwer et al. (2021) have also proposed ML based frameworks to improve IoT security. Gupta used a Raspberry Pi firewall gateway, and Anwer used Support Vector Machines (SVM) and Gradient Boosted Decision Trees (GBDT). Though these frameworks work well in small IoT settings, their use in large distributed networks remains to be tested. In Čisar et al. (2022), we used neural networks to model firewall rules, optimizing the accuracy of the model by adjusting the learning rate and momentum. But their approach is not adaptable to changing attack patterns that are common in today’s fast changing cybersecurity landscape. In Aljabri et al. (2022) the modeling of firewall rule was extended using ML and DL algorithms where Random Forest (RF) showed exceptional accuracy. Yet, to determine their approach's feasibility in the large scale, a further analysis of real-time deployment challenges and system adaptability is required. Traditional firewalls were enhanced using adaptive ML models by Musa and Victor-Ime (2023), with performance that outperformed that of conventional firewalls in identifying malicious traffic. But scalability in cloud infrastructures or distributed systems is still an open problem.

Cyber Threat Detection (SQL Injection, DDoS, Phishing, etc.)

Deep learning was used in detecting SQL injection attacks by Chen et al. (2021) and Dawadi et al. (2023) with a very high accuracy. The problem is, however, that such systems may not be able to adjust to new attack patterns without continual retraining. However, their generalization to real world settings is not well understood. In financial institutions, Islam et al. (2022) focused on DDoS attack detection, where SVM was better than other models in terms of accuracy. However, SVM is not particularly good for real time detection due to it’s slower training speed on large datasets, especially in high traffic environments. Mughaid et al. (2022) has also looked into machine learning applications in cyber security and specifically prevention of phishing and malware detection. While these approaches have promise, they still suffer from limitations for real time deployment, especially in cloud based infrastructures that require rapid and efficient detection. A web application firewall based on feature engineering to classify web requests is developed in Shaheed and Kurdy (2022), which achieves high accuracy. However, this approach does not take into account the computational overhead associated with these techniques, which may thwart real time system performance.

Malicious Activity Detection Systems

In 2022, Jeon and Tak introduced the BlackEye framework, an automatic IP blacklisting system based on a machine learning. Combining Ridge regression for data cleaning and logistic regression or Random Forest for classification, it reduced incorrect blacklisting by 90% and reduced duration of malicious IP activity by 27 days. We showed that this framework led to improved accuracy compared to human agents. In a similar fashion, Ongun et al. (2021) constructed PORTFILER, a machine learning based anomaly detection system that detected self propagating malware (SPM) like WannaCry and Mirai by using port based features from Zeek network logs. PORTFILER achieved precision of over 0.94 and very low false positive rates, outperforming both standard machine learning and deep learning methods. We demonstrated the adaptability to detect malicious activities with future work to be focused on integrating deep learning techniques for real world deployment.

Challenges in Datasets and Real-Time Applicability

In recent research, Momand et al. (2023) reviewed the progress in IDS and emphasized the necessity of updated publicly available datasets in order to help train the system. Their argument for realistic datasets is valid, however, they do not offer concrete solutions to the problem of the current datasets in reflecting new attack vectors. Chou and Jiang (2021) pinpointed the gaps in anomaly detection accuracy caused by the limitations of the dataset, especially in cloud based environment, and highlighted the need for more realistic, cloud native solutions. Their work does not provide specific strategies for addressing these dataset challenges in cloud settings, however. According to Jha and Ragha (2013), and Mayuranathan et al. (2022), hybrid ML techniques were proposed for feature selection, to reduce the data dimensionality and improve the model efficiency. These methods are helpful in improving accuracy, but they do not adequately test the real time applicability of their solutions on highly dynamic, and distributed network environments which are typical of cloud based systems. George (2022) used an ensemble AdaBoost model to select features with the hopes of tackling zero-day attacks. AdaBoost works well in certain domains, but its performance can be diminished on noisy or imbalanced data, common in real time network traffic, and thus precludes its use in real time.

Gap Analysis

In the literature, machine learning (ML) and deep learning (DL) are shown to be effective for intrusion detection systems (IDS), yet there are still gaps. The majority of studies concern themselves with model accuracy, without considering scaling to real time in a cloud based environment. In addition, although kNN and MLP are promising, they ignore the noisy data, evolving attack patterns and real time adaptability. The current research is inadequate in addressing the need for frameworks that improve performance of the algorithms in dynamic client-server architectures, especially in detecting network based attacks such as port scanning and sniffing. The need for further investigation into these methods in real-time, distributed settings, is thus justified.

3 Research Methodology

This research takes a systematic approach to evaluate the machine learning models for detecting and classifying the network traffic anomalies with the Internet Firewall Dataset available on Kaggle. The methodology that collects data, pre-processes data, develops predictive models, evaluates the models, implements the models and analyses the results to ensure the models' reproducibility and validity are applied.

3.1 Research Procedure

Experimental Setup: The experimental setup was configured as follows:

Hardware: Models were trained and tested on a system with an Intel i7 processor, 16GB RAM, and GPU.

Software: Python 3.8 was used, along with libraries like TensorFlow, Scikit-learn, Pandas, and Matplotlib.

Tools: Jupyter Notebook was used for implementation, and Wireshark was utilized for any packet-level inspection during dataset analysis.

Data Collection: The dataset consists of firewall network traffic data, categorized as Allow, Deny and Drop, meaning legitimate traffic, blocked requests and dropped packets respectively. It contains ports, byte counts, packets, and elapsed time, all features that would make it very suitable for supervised learning to detect security threats.

The dataset used in this study is the **Internet Firewall Dataset** available on Kaggle https://www.kaggle.com/datasets/tunguz/internet-firewall-data-set.

The original dataset paper : Ertam, F. and Kaya, M. (2018). Classification of firewall log files with multiclass support vector machine. 2018 6th International Symposium on Digital Forensic and Security (ISDFS). doi:https://doi.org/10.1109/isdfs.2018.8355382.

Although it provides the basics of attributes like port, action, bytes and other information related to the data transmission and receiving process, it lacks information such as the IP address, protocol type which are very important for auditing and screening the network.

Codebase: It is based on the reference Jupyter notebook 'tf-rn-internet-firewall.ipynb' which it mainly concerns with the downloading of data and undertaking of basic machine learning tasks using tensorflow. However, I added several improvements and changes to the pipeline and that includes steps like setting up Warnings, Pandas options for data visualization and the use of reliable and highly efficient libraries like scikit-learn and tensorflow for the purpose of model evaluation. Furthermore, my notebook comprises of different Analysis methodologies where attempts are made to validate the model results by generating and testing the classification report and confusion matrix . Apart from that my contributions include major enhancements in preprocessing and the flow of operations. I added code to navigate datasets for a program and manipulate data folders, included further data visualization with Seaborn, and incorporated extra measures to assess model effectiveness. In addition, I refactored the code by breaking it down into smaller and more easily understandable chunks that are easier to work with. All these changes helped me to greatly improve the operation and applicability of the original codebase while keeping the priority on the goals of the project.

Reference to notebook : https://www.kaggle.com/code/suhailo23/tf-rn-internet-firewall

Data Preprocessing

Data preprocessing takes care of all pre work done on data before training using techniques such as filling missing values using removal or imputation, scaling features using techniques Min Max or Standardization, select relevant features based on correlation analysis and domain knowledge to maximize model performance and avoid overfitting.

Model Development

In the literature review process, I realized that KNN and MLP are two popular models that possess different properties in the classification of different data sets. Compared to other classification algorithms, KNN is easy to understand, explain and applies well if the decision line in the dataset is not so complex. Instead, it is a non-parametric model that means that does not impose any assumptions on the data distribution which is rather helpful when working with practical datasets that cannot have specific distribution. On the other hand, MLP is one of the most common types of neural networks that, besides being used for learning many patterns, it is best suited for learning complicated patterns especially where inter relational complexities are most complex in a dataset. It is also applied to many of the current data-driven tasks in machine learning, and it’s flexibility in modeling data forms is why I selected this algorithm in contrast to KNN. The novelty in my selection lies in choosing these two contrasting models: KNN which is more interpretable compared to MLP which is more powerful but complex.

Thus, I restricted the comparison by using only two models to avoid hasty and inconclusive analysis during the evaluation. I believe that by comparing two rather distinct algorithms, I was able to examine each in full and gain a clear idea of its pros and cons. Having too many models might negatively affect the comparison and as a result lessen the insight of the thesis especially for a thesis that seeks to add depth to the understanding of a certain problem. That way, my goal was to produce a clear, concise comparison of and without delving into numerous algorithms that would not necessarily provide valuable information.

Two machine learning models are developed and compared for the classification task:

K-Nearest Neighbors (KNN): A KNN is a non-parametric classifier and assigns a class by the majority label of the nearest data point feature. Through cross validation, we choose an optimal number of neighbors, K.

Multilayer Perceptron (MLP): The MLP is a feed forward neural network intended to discover complex relations in the data. It is modelled with an input layer, hidden layers, output layer and has been trained using the backpropagation. We fine tune hyper parameters, such as number of hidden layers, neurons, learning rate etc to better perform.

3.2 Evaluation Methodology

The performance of the models is evaluated using several classification metrics to determine their effectiveness in identifying benign and malicious traffic:

Accuracy: This is measured in what is the overall correctness of the models in classifying instances. It is defined as correct predictions out of total predictions.

Precision, Recall, and F1-Score: And these metrics are computed for each class to see how well the model performs in identifying the class it does best. Precision is true positives divided by all the predicted positives, recall is true positive divided by actual positives, and F1 Score is the harmonic mean of precision and recall. For datasets where one class dominates the other, these metrics are very important.

Confusion Matrix: We will sum up the classification performance of the model using confusion matrix. For each class, it shows counts of true positives, true negatives, false positives, false negatives. This enables us to know which classes are most often misclassified by the models.

Accuracy & loss plot: The accuracy and loss plots give us a sense of the model's performance, how well does it learn and how well does it generalize on each epoch. You can use these visualizations to see if you are converging, overfitting or underfitting.

Detection Time: According to this, detection time is defined as the time an IDS takes to be able to determine (recognize and classify) network traffic or an activity as benign or malicious. Time for processing incoming data, extracting features and running the detection algorithm to check whether the activity is a threat is included on it. The detection time is critical for real time systems to recognize potential intrusions quickly.

Response Time: Total time taken by the system to respond to a detected threat or request is known as response time. Detection time and the time to execute additional predefined actions such as blocking a malicious IP, logging the event, raising an alert or providing feedback to the user are included. The system's stability and security demands fast response time to minimize the influence of the threats.

3.3 Explanation of Methodology

Result Analysis: A lot of models were trained and evaluated against metrics like accuracy, precision, recall and F1-score. Misclassifications were identified using the confusion matrix to understand model reliability across categories. Furthermore, the time of detection and response time of the models were evaluated. The time taken for the model to classify an instance, which we call detection time. The time to blocklist a malicious user or IP, which we refer to as response time. Afterwards, we select the model with the best overall performance, with the best results in correctly classifying malicious and benign traffic, and the best detection and response times.

Inference: Inference was performed using the chosen model, which analysed uploaded files to classify them. We fed preprocessed input data into the model, generating predictions of class label, and confidence score. The process of inferring these features enabled accurate classification and formed the core mechanism of real time decision making.

Frontend Integration: The model with best performance is selected based on the performance of models. After testing that the request is of an attack or not, the front will give the data that is selected model.

3.4 Rationale for Methodology

Contribution and Justification: In this thesis, we present an automated system for network traffic classification that increases security. The major contribution is combining machine learning models, inference processes and a user friendly front end. Model selection, data preprocessing, and seamless front end integration are parts of the methodology which allows for real time traffic analysis and accurate classification. Intuitive displays of classification results increase efficiency in security operations as the system helps network administrators make informed decisions. Such approach could be scalable and effective for network traffic monitoring.

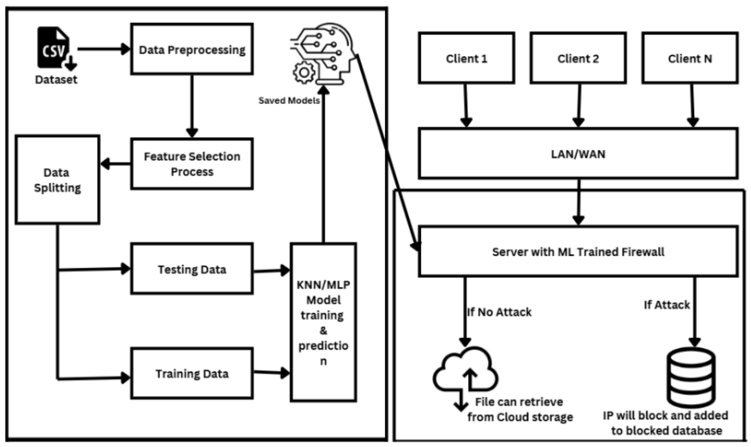
Relevance of the Dataset and Dataset Representativeness: For its diverse representation of network traffic and availability of relevant features (packets count, action types) the Kaggle Internet Firewall Dataset was selected. It is of sufficient size and variety that it can provide robust model performance in real world scenarios. The dataset structures 65,532 network traffic observations into 12 fields which capture information about ports and connection packets and actions such as "allow." The dataset's sample size offers sufficient analysis capabilities however its representative power relies on the diverse nature of data sourcing areas including geographic locations and time sections and operational network domains. The dataset shows signs of focusing on particular applications with numbers for ports and packet data characteristics that point to enterprise and corporate networks yet these features can restrict insights beyond enterprise settings extending to public Wi-Fi or data center networks. The narrow collection window for the data might result in temporal biases that would obscure seasonal fluctuations together with time-of-day patterns. The performance of security models relies heavily on the balanced distribution of connections between allowed and denied categories in the Action column. Because research focuses on typical ports (443 for HTTPS and 3389 for RDP) the analysis may fail to include less familiar network protocols. When relying on automated labeling systems for the Action column there exists a risk that errors will occur due to intrinsic biases in these systems. The dataset features insufficient contextual information about user activities and deployed software programs or performance data. Skewed distributions in Bytes and Elapsed Time (sec) numeric features exist alongside possible data cleaning procedures that potentially removed rare or disconnected events which could diminish the dataset's capacity to identify anomalous traffic patterns. Preprocessing and analytical techniques must be performed with caution because they produce findings which need to be both robust and generalizable.

Addressing Existing Challenges: The challenge of real time network traffic analysis and threat detection is solved by this project. It combines machine learning and a nice front end to allow quick, automated classification and monitoring capabilities that are often missing from traditional security systems.

4 Design Specification

System Design with Port and IP Detection:

It proposes the design of a system that combines machine learning based model training process with client server based cybersecurity framework, which can deal with real time cyber threats such as network intrusion and port scan detection.


Dataset
Data Preprocessing
Saved Models
Data Splitting
Feature Selection Process
Testing Data
Training Data
KNN/MLP Model training & predictio n
Client 1
Client 2
Client N
LAN/WAN
Server with ML Trained Firewall
If No Attack
If Attack
File can retrieve from Cloud storage
IP will block and added to blocked database

The Model Training Process involves several critical steps: pre processing of data, preparing and organizing raw dataset, selection relevant features for enhancing model accuracy and efficiency, splitting data into training data and testing data. By taking these steps, we create high performing models like KNN and Multi Layer Perceptron (MLP) and store them as Saved Models for deployment. Besides detecting malicious traffic, the system comprises Port and IP Detection, which through the use of advanced machine learning techniques detects suspicious IP addresses and unusual port scanning activity. The system specifically trained the ML model to detect port scans and unauthorized access attempts to certain IPs to effectively tell legitimate

scans from malicious ones. The result is an improved capability to identify network intrusions and thwart threats in real time, for strong protection against threats that exploit network vulnerabilities.
Finally, the trained models are incorporated in a Client-Server Cybersecurity Framework, where the client system is modeled as the end-user devices that are susceptible to attacks, while the server serves the best performing attack detection model. Real time threat detection works on the server including identifying and stopping attacks, keeping a Block List IP database to block malicious sources and keeping critical data in Cloud Storage securely to ensure scalability and reliability. On detecting attacks, a Notification System alerts users or administrators to improve proactive threat management. We leverage advanced machine learning techniques with scalable, distributed frameworks that effectively combine together to form a robust solution to detect and mitigate network based attacks in real-time, including IP and port scan detection.

5 Implementation

This work is developed and the implementation section describes the development. It involves Data preprocessing, Model Development, Training, Inferencing, Frontend and Backend Integration for Real Time Threat Monitoring. Trained models are used to analyse traffic accurately in the system which give results through an intuitive user interface.

5.1 Data Preprocessing

Kaggle Internet Firewall Data Set is the dataset used for this study, containing 12 features and 65,532 records of network traffic. They consist of 11 numerical features, e.g. Source Port, Bytes Sent, Elapsed Time (sec), and a single categorical feature, Action, as class label.

Data Inspection: First, the dataset was looked at for missing or infinite values. There were no such issues found confirming the integrity of data. Also, the data types were also verified for the compatibility with downstream machine learning processes.

Handling Imbalanced Classes: In the initial class distribution, there were four labels: allow, deny, drop, and reset-both. reset-both was excluded due to minimal records (54). For the remaining classes (allow, deny, and drop), 12,000 records were sampled for each of these classes to balance the dataset to 36,000 records.

Class Distribution: The dataset initially exhibited a class imbalance, as shown in Figure 2, with the following distribution:

- allow: 37,640 records
- deny: 14,987 records
- drop: 12,851 records

- reset-both: 54 records

This class was not included due to insufficient records for reset-both. To make the dataset balanced, 12,000 records from the other classes (allow, deny, and drop) were selected, to give a total of 36,000 records. Figure 3 shows the balanced distribution.

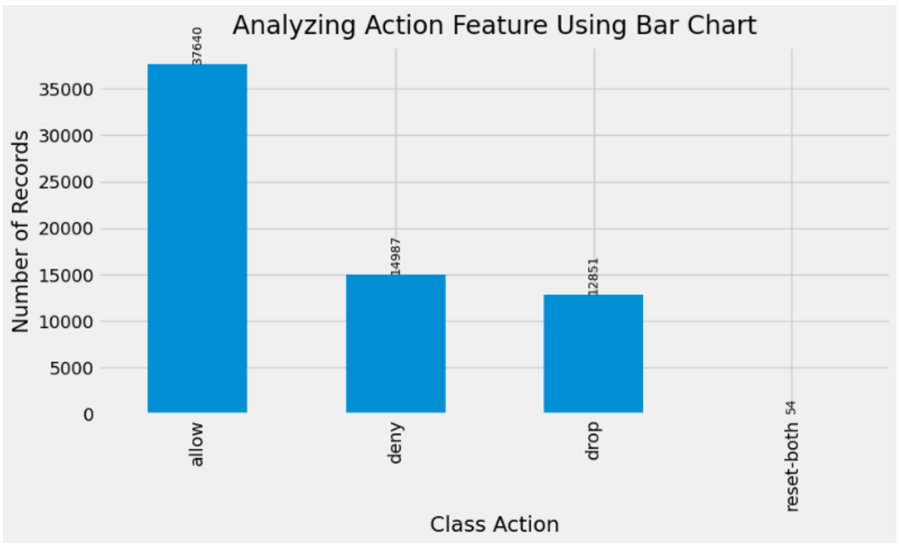


Figure 1 : Imbalance class Features

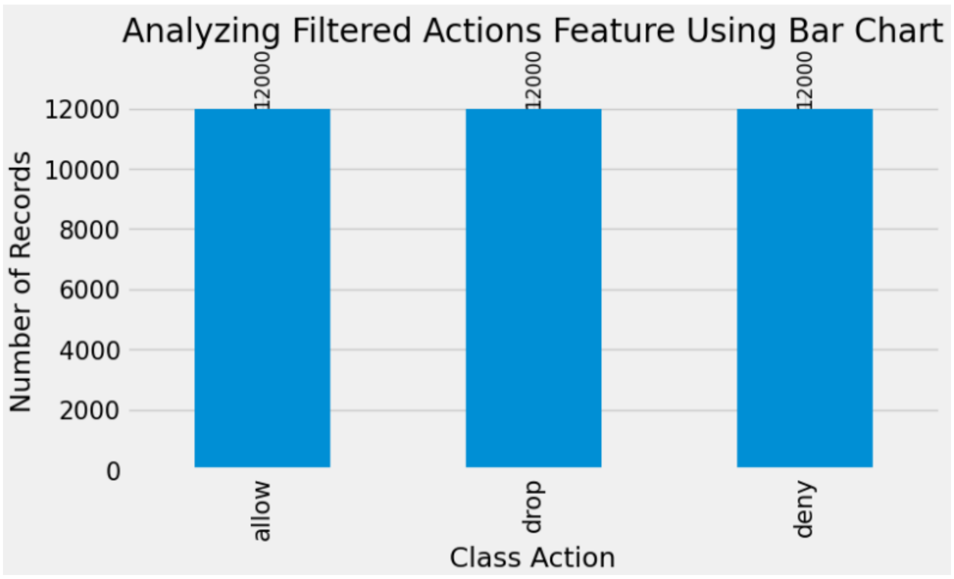


Figure 2 :Balanced Class Features

Encoding and Normalization: The **Action** column was encoded into numerical values:

- *allow*: 0
- *deny*: 1
- *drop*: 2

Min-Max Normalization was applied to scale numerical features to a range of [0, 1].

Data Splitting: The dataset was shuffled and split into training (80%) and testing (20%) sets while preserving class balance. The split resulted in:

- Training set: 28,800 samples
- Testing set: 7,200 samples

Saved Outputs

- Filtered data (filtered_data.csv)
- Scaled data (scaled_data.csv)
- Scaler model (scaler.pkl)
- Train/Test datasets (X_train.csv, X_test.csv, y_train.csv, y_test.csv)

Model Development: Scikit learn was used to develop the KNN model optimized for neighbor count; the MLP model developed using TensorFlow/Keras uses sequential architecture with dense layers and batch normalization to increase classification accuracy.

Model Training: The neighbor count for KNN model was varied and trained to achieve optimal classification while the MLP model used deep learning techniques, regularization and Adam optimizer to improve performance. Evaluation was done through validation accuracy, while training both models on split data.

Model Training for KNN:

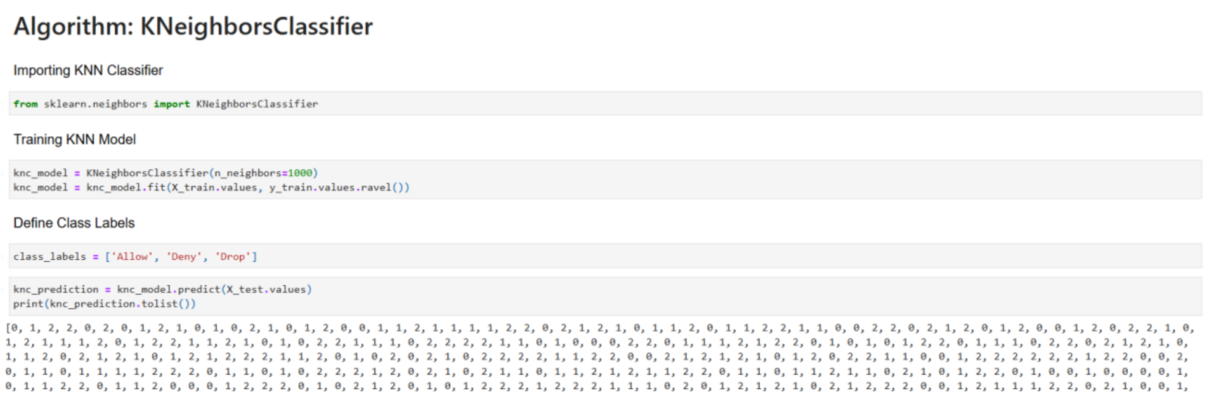
Data Preprocessing: Min-Max Normalization was used to scale the dataset so that all features were in the range [0, 1] as this is required to use distance based algorithms such as KNN.

Model Initialization: A KNN model was implemented as in Figure 3 using Scikit-learn with the number of neighbors (K) chosen via cross-validation balancing accuracy with generalization.

Training Phase: X_train and y_train is the training dataset on which KNN model is fitted. The training data was stored by the model during training to calculate distances to be used for classification.

Hyperparameter Tuning: We explored optimal neighbor counts on validation sets to enhance classification performance.

Evaluation: The model was then trained on the test dataset (X_test, y_test) with accuracy, precision and recall performance metrics.



Algorithm: KNeighborsClassifier

Importing KNN Classifier

```
from sklearn.neighbors import KNeighborsClassifier
```

Training KNN Model

```
knc_model = KNeighborsClassifier(n_neighbors=1000)
knc_model = knc_model.fit(X_train.values, y_train.values.ravel())
```

Define Class Labels

```
class_labels = ['Allow', 'Deny', 'Drop']
```

```
knc_prediction = knc_model.predict(X_test.values)
print(knc_prediction.tolist())
```

```
[0, 1, 2, 2, 0, 2, 0, 1, 2, 1, 0, 1, 0, 2, 1, 0, 1, 2, 0, 0, 1, 1, 2, 1, 1, 1, 1, 2, 2, 0, 2, 1, 2, 1, 0, 1, 1, 2, 0, 1, 1, 2, 2, 1, 1, 0, 0, 2, 2, 0, 2, 1, 2, 0, 1, 2, 0, 0, 1, 2, 0, 2, 2, 1, 0,
1, 2, 1, 1, 1, 2, 0, 1, 2, 2, 1, 1, 2, 1, 0, 1, 0, 2, 2, 1, 1, 1, 0, 2, 2, 2, 2, 1, 1, 0, 1, 0, 0, 0, 2, 2, 0, 1, 1, 1, 2, 1, 2, 2, 0, 1, 0, 1, 0, 1, 2, 2, 0, 1, 1, 1, 0, 2, 2, 0, 2, 1, 2, 1, 0,
1, 1, 2, 0, 2, 1, 2, 1, 0, 1, 2, 1, 2, 2, 2, 1, 1, 2, 0, 1, 0, 2, 0, 2, 1, 0, 2, 2, 2, 2, 1, 1, 2, 2, 0, 0, 2, 1, 2, 1, 2, 1, 0, 1, 2, 0, 2, 2, 1, 1, 0, 0, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 0, 0, 2,
0, 1, 1, 0, 1, 1, 1, 1, 2, 2, 2, 0, 1, 1, 0, 1, 0, 2, 2, 2, 1, 2, 0, 2, 1, 0, 2, 1, 1, 0, 1, 1, 2, 1, 2, 1, 1, 2, 2, 0, 1, 1, 0, 1, 1, 2, 1, 1, 0, 2, 1, 0, 1, 2, 2, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1,
0, 1, 1, 2, 2, 0, 1, 1, 2, 0, 0, 0, 1, 2, 2, 2, 0, 1, 0, 2, 1, 2, 0, 1, 0, 1, 2, 2, 2, 1, 2, 2, 2, 1, 1, 1, 0, 2, 0, 1, 2, 1, 2, 1, 0, 2, 1, 2, 2, 2, 0, 0, 1, 2, 1, 1, 1, 2, 2, 0, 2, 1, 0, 0, 1,
```

Figure 3 : Code Snippet for KNN Model Training

Model Training for MLP:

Data Preparation: The data was preprocessed and split into training and testing sets in the same way KNN was done, so that the input to the MLP model was consistent.

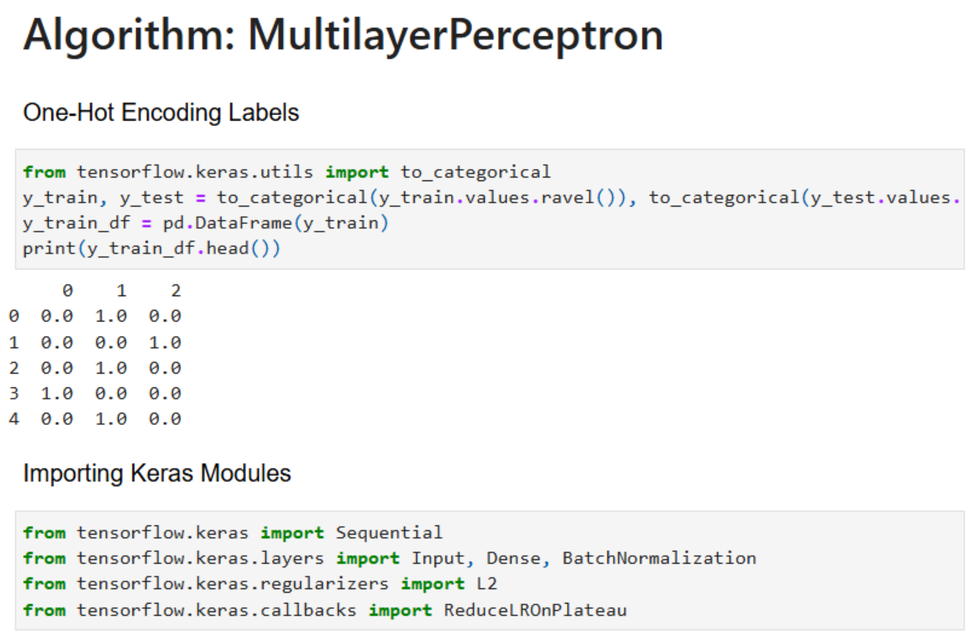
Model Architecture: A sequential architecture was implemented, using TensorFlow/Keras, to implement the MLP. Input, multiple dense layers with ReLU activation, batch normalization, output layer with softmax activation for multi class classification was included.

Compilation: We compiled the model with Adam optimizer, categorical cross entropy loss function and accuracy as the evaluation metric.

Training Process: When feeding model with proper batch size, we trained the model on X_train and y_train for 50 epochs. To prevent overfitting techniques such as regularization techniques like dropout was applied.

Validation: The model performance was monitored during training on a validation set, by monitoring convergence and checking for overfitting by means of metrics such as loss or accuracy.

Evaluation: After the training, MLP model was evaluated with the unseen X_test and y_test datasets. The classification reliability was validated through the use of metrics such as F1-score, confusion matrix and accuracy.


Algorithm: MultilayerPerceptron

One-Hot Encoding Labels

```
from tensorflow.keras.utils import to_categorical
y_train, y_test = to_categorical(y_train.values.ravel()), to_categorical(y_test.values.
y_train_df = pd.DataFrame(y_train)
print(y_train_df.head())
```

```
     0    1    2
0  0.0  1.0  0.0
1  0.0  0.0  1.0
2  0.0  1.0  0.0
3  1.0  0.0  0.0
4  0.0  1.0  0.0
```

Importing Keras Modules

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Input, Dense, BatchNormalization
from tensorflow.keras.regularizers import L2
from tensorflow.keras.callbacks import ReduceLROnPlateau
```

Figure 4 : Code Snippet for MLP Model Training

KNN Model Implementation: The KNN classifier was trained with network traffic data set to 1000 neighbors with an objective of categorizing the network traffic into either allow, deny, or drop. On test data, evaluation of the performance of the model was done.

MLP Model Implementation: The developed MLP model with TensorFlow/Keras was containing multiple dense layers with ReLU activation and regularization. It had three categories of the network traffic, and the outcomes of the softmax layer were class probabilities.

Inferenceing: As shown by model evaluation the best performing model was subjected to an inference system from the front end of the developed model.

Front-End Development: The front end design is socket program based, where the CLIENT and ADMIN interfaces can interface independently to request files, or identify and eliminate threats in real time. This website is developed using HTML, CSS and Java script languages.

Backend Integration: The backend programming was done using Python with flask, which does the work of controlling the user inputs received then scales the data to classify using the trained MLP model. The front-end obtains predictions from it, it deals with security of alerts and logs every transaction for tracking purposes.

Output and Results: Several outputs from the implementation are offered, including: Transformed data (post-preprocessing) as in Figure 2 & Figure 3; Pre-processed data readily consumable for model input; Threat classification results display interface including attack categories and their corresponding probabilities; Threat detections user interface including detailed logs on the security status of transactions; Threat as updates. These outputs enable efficient monitoring and security of the traffic in the network and the file transfer.

6 Evaluation

The baseline paper is centered round improving the identification of port scan risk in cybersecurity by employing feature selection algorithms known as ACO (Ant Colony Optimization), GA (Genetic Algorithm), and GWO (Gray Wolf Optimization) in combination with two machine learning algorithms, namely SVM AND KNN. The authors followed the feature selection method on the dataset – CICIDS2017 including 78 features encompassing different attacks' scenarios The CICIDS2017 has many attacks features, including 78 to determine an attack, thus, they reduce the computational burden by applying feature selection. For specific models, ACO chose 10 features, GA chose 13 and GWO selected 12. These selected features were used to train classifiers of Support Vector Machine (SVM) and k-Nearest Neighbor (kNN) where the traffics were categorized as either benign or malicious.

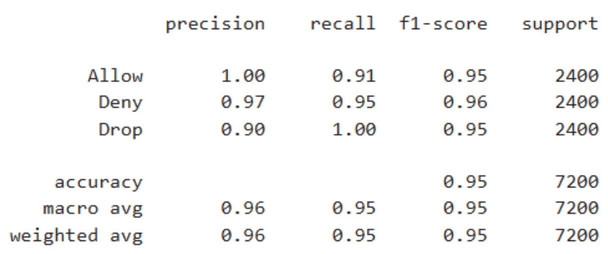
The performance of developed models was satisfactory, the accuracy of models was within 97%. Thus, accuracy, precision, recall, F1-score, sensitivity, and specificity were the key classifiers showing high classification potential. According to this study, it was confirmed that the best result was obtained with the GWO+SVM with the accuracy of approximately 98% and the best performances of all indices introduced in the study. ACO+SVM and GA+KNN also reached good results albeit slightly lower values compared to GWO+SVM. However, this poses a major drawback in large real-time applications because of the high computational complexity that is normally involved in performing the optimization. Although the feature selection successfully eliminates the dimensional problem and enhanced the classification, using this method offers extra time-consuming which makes it less feasible that can be applied in real-time intrusion detection.

A clear limitation with the study is that no evaluation or consideration of the detection time and a somewhat related factor is the response time. These metrics are as important for the analysis of the time factor in threat assessment and prevention by the system. Lack of such analysis raises questions regarding the practicality of the system in actual use, tactical, day to day cyber-security environments.

The experimental results of the proposed models and system implementation are evaluated and critically analyzed below. After revealing the findings, it discusses implications for academic Discussions. This paper consists of evaluation of K Neighbors Classifier in detail along with the results of the proposed models and the implementation of the system. It describes the findings, subsequently examine its Implication on academic Discourses.

6.1 Evaluation of K Neighbors Classifier

The validation accuracy of the K-Nearest Neighbors (KNN) Classifier achieved 95.29%. Across all classes, the model was high in precision, recall, and F1-scores. The confusion matrix shows the model has a large majority of correct predictions, and we see a few misclassifications, which shows that it is doing great at being able to distinguish between the classes.



```
              precision    recall  f1-score   support

       Allow       1.00      0.91      0.95      2400
        Deny       0.97      0.95      0.96      2400
        Drop       0.90      1.00      0.95      2400

    accuracy                           0.95      7200
   macro avg       0.96      0.95      0.95      7200
weighted avg       0.96      0.95      0.95      7200
```

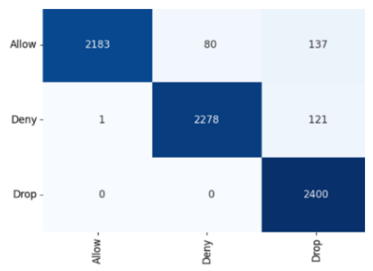


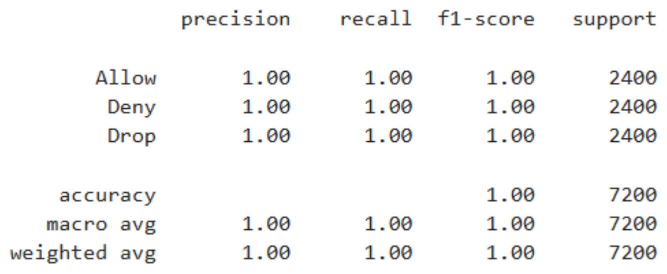
Figure 5: Classification report & Confusion matrix of K Neighbors Classifier

Classification report: As shown in Figure 6 right, overall accuracy is strong at 0.95. The support and precision, recall, and F1-score for the Allow class are 24,800, 1.00, 0.91, and 0.95 respectively. For the Deny class metrics, we have precision as 0.97, recall as 0.95, F1-score as 0.96 and support as 24,800. For the same support, the Drop class had precision 0.90, recall 1.00, F1-score 0.95. The macro average and weighted average F1-scores are 0.96, indicating balanced and effective classification for all categories.

Confusion Matrix: The following results are shown in Figure 6 left: 2183 correct predictions for "Allow", 80 incorrect "Deny" predictions for "Allow", 137 incorrect "Drop" predictions for "Allow", 1 incorrect "Allow" prediction for "Deny", 2278 correct predictions for "Deny", 121 incorrect "Drop" predictions for "Deny", and 2400 correct predictions for "Drop". The classifier worked well according to the matrix as most of the predictions were correct especially for the "Allow" and "Drop" classes. All off diagonal values represent misclassifications, with little misclassification between "Allow" and "Deny", and none between "Drop" and either of the other 2.

6.2 Evaluation of Multi-layer Perceptron

Validation accuracy with the Multilayer Perceptron (MLP) model shows strong performance with 99.83%. The proposed method effectively classifies instances into multiple categories and obtains high precision and recall for each category. Model is very good in recognizing "Allow" and "Deny" and has very little misclassification. It however confuses a bit the "Deny" and "Drop" categories. Therefore, the MLP model is very accurate and does a pretty good job with this task.



```
              precision    recall  f1-score   support

       Allow       1.00      1.00      1.00      2400
        Deny       1.00      1.00      1.00      2400
        Drop       1.00      1.00      1.00      2400

    accuracy                           1.00      7200
   macro avg       1.00      1.00      1.00      7200
weighted avg       1.00      1.00      1.00      7200
```

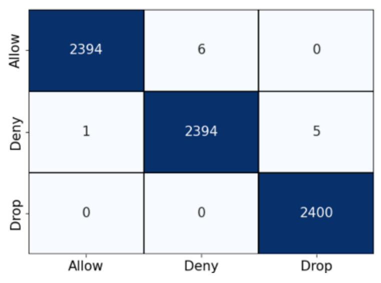


Figure 6: Classification report & Confusion matrix of MLP

Classification report: Figure 7 is perfect on all metrics. For each class ('Allow', 'Deny', 'Drop') the precision, recall and F1-score are all 1.00 and support is 2400 for each class. Zero average, weighted average and the overall accuracy are also 1.00. The model achieved perfect prediction on the task with total support of 7200, across all classes.

Confusion matrix: The following results are shown in Figure 7: 2394 true positives for "Allow," 6 false positives for "Deny," 5 false negatives for "Deny," 0 false negatives for "Allow," 2394 true negatives for "Deny," and 2400 false positives for "Drop." The model does a good job of picking up the "Allow" and "Deny" instances, but it often mistakes "Deny" instances as "Drop" (i.e. 2400 false positives for "Drop").

Accuracy plot: In Fig 8, For the training accuracy (blue line), the accuracy begins at a high point and continually increases until it reaches nearly 1.0 (100), which shows that the present model work well on the training dataset. On the other hand, similar to validation loss, the validation accuracy (in red line) undergoes a steep decline during the initial epochs reflecting the poor performance. Yet it quickly recovers and gets close to the training accuracy at almost 100% immediately, soon after such fluctuations. This could indicate problems with the validation data, including data leakage or insufficient variation in the validation dataset.

Loss plot: In Figure 9,The training loss here, or the blue curve, expands over the epochs showing that the model is consistent and maximizing its parameters for the training data. Instead, the validation loss (red line) increases considerably and sharply at the second epoch and testifies to greatly excessive fluctuations and weak generalization during the initial epochs of training. After this peak the validation loss sharply decreases down to zero and stabilizes at zero while, this might be indicating overfitting or even data doping where training and validation datasets share similar data.

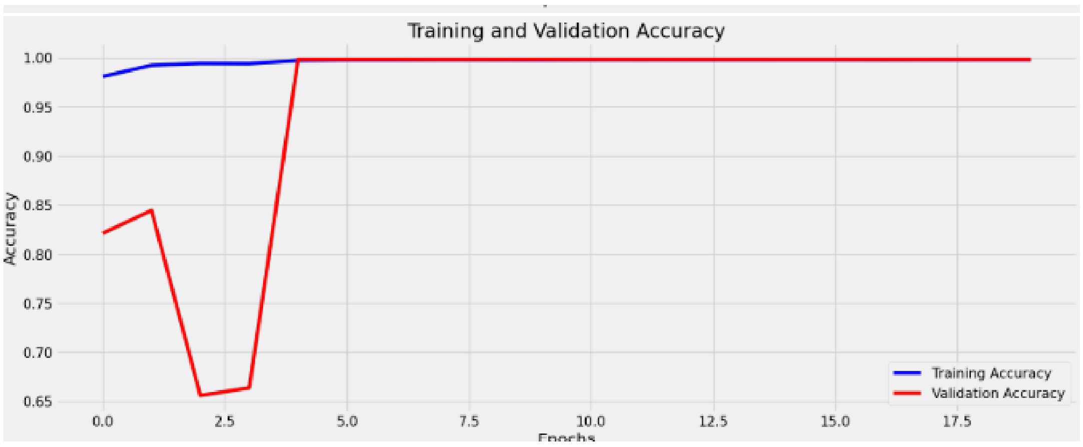


Figure 7: Accuracy Plot

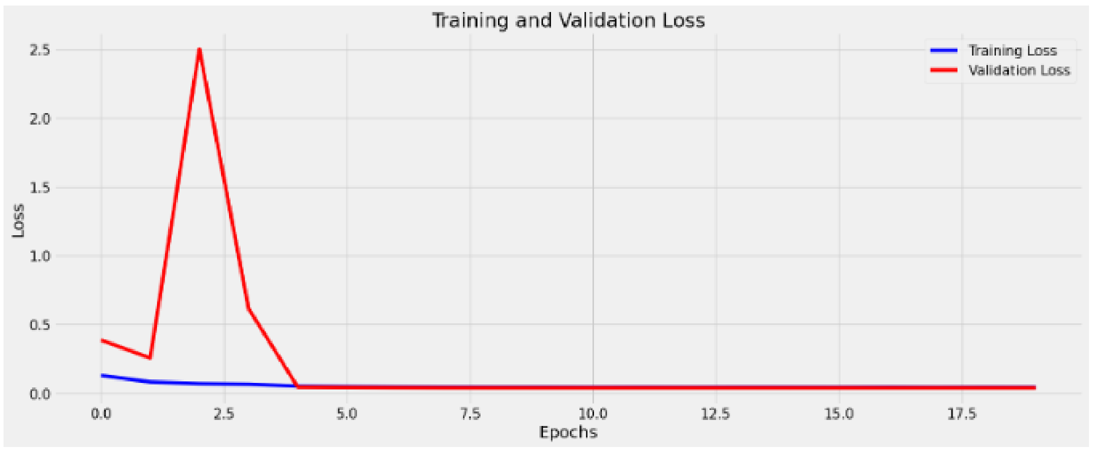
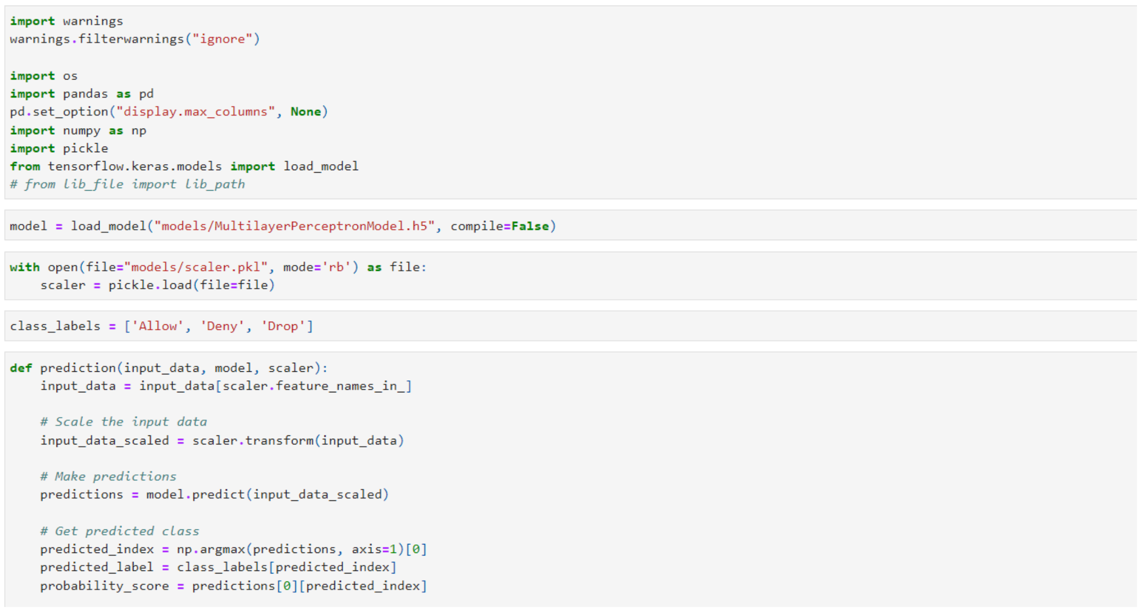


Figure 8: Loss Plot

6.3 Inference

Using the provided inference code, network traffic is classified into three categories based on the pre trained model. It accepts user provided data, processes it by using a saved scaler and predicts the output class and its confidence score. As the most accurate model, the MLP model was chosen for inference.

The inference code in Figure 10 follows a systematic process: It loads the pre trained MLP model and saved scaler for data preprocessing. The scaler is loaded and used to scale the user input data, then the user input data is read and matched to the feature set required. Next, the scaled data is fed to the MLP model, which predicts the probabilities for the class (taken to be "Allow," "Deny" or "Drop"). A predicted output is selected to be the class with the highest probability, and the user is displayed the label, class index and confidence score.



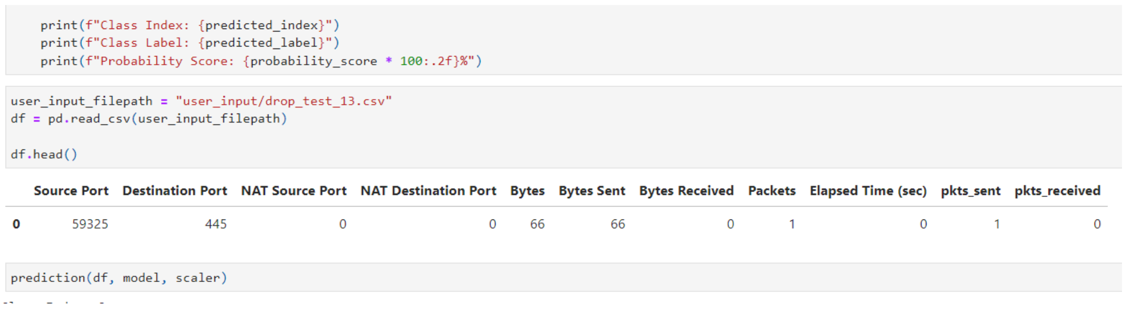
```
import warnings
warnings.filterwarnings("ignore")

import os
import pandas as pd
pd.set_option("display.max_columns", None)
import numpy as np
import pickle
from tensorflow.keras.models import load_model
# from lib_file import lib_path
```

```
model = load_model("models/MultilayerPerceptronModel.h5", compile=False)
```

```
with open(file="models/scaler.pkl", mode='rb') as file:
    scaler = pickle.load(file=file)
```

```
class_labels = ['Allow', 'Deny', 'Drop']
```

```
def prediction(input_data, model, scaler):
    input_data = input_data[scaler.feature_names_in_]

    # Scale the input data
    input_data_scaled = scaler.transform(input_data)

    # Make predictions
    predictions = model.predict(input_data_scaled)

    # Get predicted class
    predicted_index = np.argmax(predictions, axis=1)[0]
    predicted_label = class_labels[predicted_index]
    probability_score = predictions[0][predicted_index]
```



```
    print(f"Class Index: {predicted_index}")
    print(f"Class Label: {predicted_label}")
    print(f"Probability Score: {probability_score * 100:.2f}%")
```

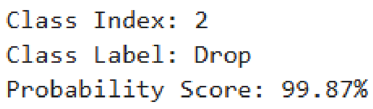
```
user_input_filepath = "user_input/drop_test_13.csv"
df = pd.read_csv(user_input_filepath)

df.head()
```

	Source Port	Destination Port	NAT Source Port	NAT Destination Port	Bytes	Bytes Sent	Bytes Received	Packets	Elapsed Time (sec)	pkts_sent	pkts_received
0	59325	445	0	0	66	66	0	1	0	1	0

```
prediction(df, model, scaler)
```

Figure 9 : Inference code snippet

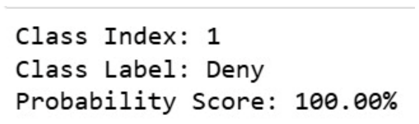
Since the input provided based on the above code is “user_input/drop_test_13.csv” which is a malicious input, its output is shown below along with the Probability Score and the class label of the input along with its index as shown in figure 10.



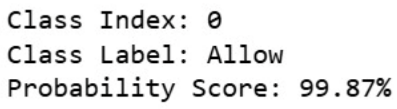
```
Class Index: 2
Class Label: Drop
Probability Score: 99.87%
```

Figure 10 :Output for Malicious

Additionally, the output for other class labels with benign and malicious are as follows in figure 12 & figure 13.



```
Class Index: 1
Class Label: Deny
Probability Score: 100.00%
```



```
Class Index: 0
Class Label: Allow
Probability Score: 99.87%
```

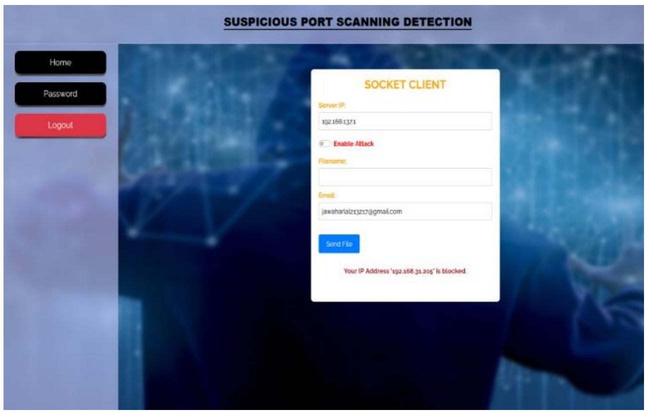
Figure 11 :Output for Malicious **Figure 12 :Output for Benign**

This code is a systematic network traffic classification. It combines pre trained models, feature scaling, and easily interpretible class probabilities to effectively support decision making in real world applications such as intrusion detection systems or access control mechanisms.

6.4 Front-End Working

Attack Detection and Response:

On the right side of Figure 18, the Attack option is enabled, which means the system will do an attack check, examine incoming requests at the port and the IP address. The model evaluates the request and if it detects malicious activity it adds the provided IP to the restricted list. Then the message 'Your IP address is blocked' is displayed. As we can see on the left side of Figure 18, the server log is showing how the incoming request is being processed and a high confidence 'Deny' prediction from the model. The server resumes listening for new connections and blocks the request. The malicious or normal activity detection time is approximately 5 seconds. Figure 19 shows this admin screen displaying a list of blocked IPs that have requested files which have been deemed malicious. When the bad activity is detected, the IP is blocked on the blocklist within about 50 milliseconds to prevent additional requests. After being added to the blocklist the IP can no longer make further requests.



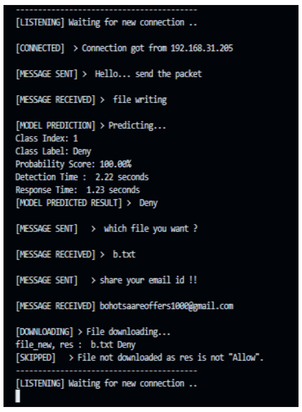
Server IP: 192.168.137.1

Enable Attack

Filename:

Email: jawaharlal213217@gmail.com

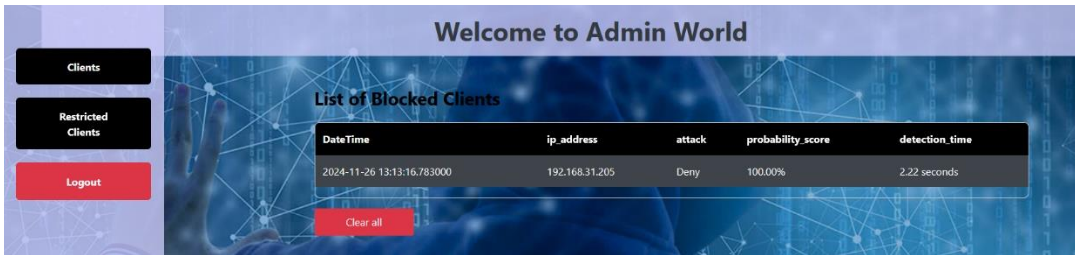
Your IP Address '192.168.31.205' is blocked.



```
------------------------------------------
[LISTENING] Waiting for new connection ..

[CONNECTED]  > Connection got from 192.168.31.205

[MESSAGE SENT] >  Hello... send the packet

[MESSAGE RECEIVED] >  file writing

[MODEL PREDICTION] > Predicting...
Class Index: 1
Class Label: Deny
Probability Score: 100.00%
Detection Time :  2.22 seconds
Response Time:  1.23 seconds
[MODEL PREDICTED RESULT] >  Deny

[MESSAGE SENT]   >  which file you want ?

[MESSAGE RECEIVED] >  b.txt

[MESSAGE SENT]   > share your email id !!

[MESSAGE RECEIVED] bohotsaareoffers1000@gmail.com

[DOWNLOADING] > File downloading...
file_new, res :  b.txt Deny
[SKIPPED]   > File not downloaded as res is not "Allow".
------------------------------------------
[LISTENING] Waiting for new connection ..
```

Figure 13 : Dash board for CLIENT during attack

This admin screen in Figure 19 displays a list of restricted IPs that have been blocked from requesting files due to malicious activity. After detecting malicious activity, the IP is added to the blocklist within approximately 50 milliseconds, ensuring swift action to prevent further requests. Once added to the blocklist, the IP is no longer allowed to make additional requests.

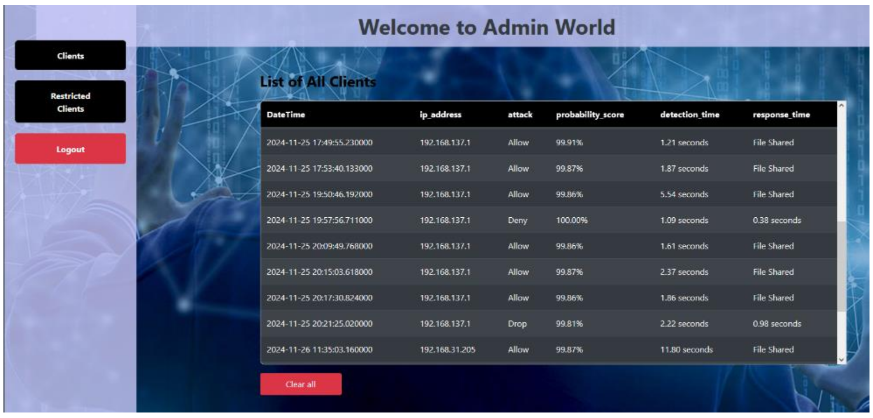


List of Blocked Clients

DateTime	ip_address	attack	probability_score	detection_time
2024-11-26 13:13:16.783000	192.168.31.205	Deny	100.00%	2.22 seconds

Figure 14 : Dash board for ADMIN during attack

This Figure 20 is for the admin, where they can view the history of IP requests for files, along with their response statuses, and delete the history using the "Clear All" button.



List of All Clients

DateTime	ip_address	attack	probability_score	detection_time	response_time
2024-11-25 17:49:55.230000	192.168.137.1	Allow	99.91%	1.21 seconds	File Shared
2024-11-25 17:53:40.133000	192.168.137.1	Allow	99.87%	1.87 seconds	File Shared
2024-11-25 19:50:46.192000	192.168.137.1	Allow	99.86%	5.54 seconds	File Shared
2024-11-25 19:57:56.711000	192.168.137.1	Deny	100.00%	1.09 seconds	0.38 seconds
2024-11-25 20:09:49.768000	192.168.137.1	Allow	99.86%	1.61 seconds	File Shared
2024-11-25 20:15:03.618000	192.168.137.1	Allow	99.87%	2.37 seconds	File Shared
2024-11-25 20:17:30.824000	192.168.137.1	Allow	99.86%	1.86 seconds	File Shared
2024-11-25 20:21:25.020000	192.168.137.1	Drop	99.81%	2.22 seconds	0.98 seconds
2024-11-26 11:35:03.160000	192.168.31.205	Allow	99.87%	11.80 seconds	File Shared

Figure 15 : Client that are active

6.5 Discussion

The functionalities tested for the system are the client's transaction flow and the system's effectiveness in recognizing security risks. On the set, the validation the accuracy of network traffic classification calculated using KNN classifier is 95.29% of the time. That is why I created the confusion matrix where it was observed that 'Allow' and 'Drop' classes are well determined by the model but there are some mistakes made. The errors shown ensure that if feature selectivity is fine tuned or if parameters are adjusted, there is enhanced classified accuracy.

The KNN classifier was significantly outperformed by the **Multi Layer Perceptron (MLP)** model, where the validation accuracy was very near to perfect (99.83%). Although it is significantly more accurate, upon closer inspection, we find that it misclassifies some of the "Deny" and "Drop" categories, which we believe may be an indication of improvement that could be made in the representation of the data or regularization of the model to reduce the number of such errors. In Training and validation accuracy MLP training and validation accuracy get diverged sharply after sixth epoch and this is due to overfitting which can be perform using dropout or early stopping or learning rate tuning.

Concerning the operational perspective, the client – server has demonstrated a normal performance in handling requests and simultaneously offering feedback, including the blocking of the IPs that are malicious. The ability to monitor, manage and clear blocked IPs on the admin dashboard added an extra level of security to the work done by the system. However, continuity and updating of the created model are acknowledged to be crucial activities aimed at the prediction of system reliability at a later period while the reliability itself heavily depends upon the updating and validation of the model on a regular basis.

From the results we can infer that the system is feasible for intrusion detection and network access management. Though having very high accuracy the authors recommend to enhance still the model and to provide additional security checks in order to make the model still more reliable and accurate in the few cases if there can appear false positives or false negatives.

7 Conclusion

The focus of this study is to design and implement an adaptive network traffic classification management system using machine learning algorithms, that accurately classify the incoming requests as legitimate or malicious. The goals were met by implementing and evaluating K-Nearest Neighbors (KNN) classifier and the Multi Layer Perceptron (MLP) model. This work revealed that the model achieved validation accuracy of 95.29% thereby supporting its reliability in the classification of all classes with a weighted F1 score of 0.96. While using MLP, high percentage of validation accuracy, 99.83% perfect precision, recall and F1 scores for most classes demonstrated improved performance of the model over KNN. The work emphasizes that both models for identification of malicious IP addresses and port scanning are beneficial in improving the subject of intrusion detection for security concerns. Furthermore, there was approximately 5 seconds for malware identification which makes it possible to analyze the incoming requests and create a response in time; approximately 50 ms was required for the updating block list and further restriction of the malicious IPs.

The operational front-end was able to provide client and admin functionalities smoothly, such as file sharing, IP blocking malicious activity, port and IP scanning detection and request management. However, MLP has minor limitation being that it is prone to overfitting and KNN has minor limitation being that it's sometimes subject to misclassification. The system as a whole achieved the research objectives and provided a solid basis for network traffic classification and management, and can effectively deal with suspicious IP and port scans.

More of the work that can be done is making this system more effective by extent of the data base and front end. More data augmentation enhanced by various types of attacks and real traffic data would enhance the model flexibility even more at the same time as more elaborated data augmentation and feature engineering techniques would enable the models to discover more sophisticated patterns particularly for port scan and IP network behavior attack. In the front-end, measurements for real-time traffic monitoring and classification as well as dynamic dashboards according to the visualization techniques to enable users to work on the appropriate countermeasure of the security events are included. The additional options on multilingual support and the accommodations proposed would allow to build upon the purpose of the present system — to make it more amicable and adaptive.

References

Ahmad, Z., Shahid Khan, A., Wai Shiang, C., Abdullah, J. and Ahmad, F., 2021. Network intrusion detection system: A systematic study of machine learning and deep learning approaches. Transactions on Emerging Telecommunications Technologies, 32(1), p.e4150.

Ahmed, H.A., Hameed, A. and Bawany, N.Z., 2022. Network intrusion detection using oversampling technique and machine learning algorithms. PeerJ Computer Science, 8, p.e820.

Aljabri, M., Alahmadi, A.A., Mohammad, R.M.A., Aboulnour, M., Alomari, D.M. and Almotiri, S.H., 2022. Classification of firewall log data using multiclass machine learning models. Electronics, 11(12), p.1851.

Al-Shareeda, M.A., Manickam, S. and Saare, M.A., 2023. DDoS attacks detection using machine learning and deep learning techniques: Analysis and comparison. Bulletin of Electrical Engineering and Informatics, 12(2), pp.930-939.

Alsoufi, M.A., Razak, S., Siraj, M.M., Nafea, I., Ghaleb, F.A., Saeed, F. and Nasser, M., 2021. Anomaly-based intrusion detection systems in iot using deep learning: A systematic literature review. Applied sciences, 11(18), p.8383.

Anwer, M., Khan, S.M. and Farooq, M.U., 2021. Attack detection in IoT using machine learning. Engineering, Technology & Applied Science Research, 11(3), pp.7273-7278.

Ashiku, L. and Dagli, C., 2021. Network intrusion detection system using deep learning. Procedia Computer Science, 185, pp.239-247.

Azizan, A.H., Mostafa, S.A., Mustapha, A., Foozy, C.F.M., Wahab, M.H.A., Mohammed, M.A. and Khalaf, B.A., 2021. A machine learning approach for improving the performance of network intrusion detection systems. Annals of Emerging Technologies in Computing (AETiC), 5(5), pp.201-208.

Bouke, M.A., Abdullah, A., ALshatebi, S.H. and Abdullah, M.T., 2022. E2IDS: an enhanced intelligent intrusion detection system based on decision tree algorithm. Journal of Applied Artificial Intelligence, 3(1), pp.1-16.

Chen, D., Yan, Q., Wu, C. and Zhao, J., 2021. Sql injection attack detection and prevention techniques using deep learning. In Journal of Physics: Conference Series (Vol. 1757, No. 1, p. 012055). IOP Publishing.

Chou, D. and Jiang, M., 2021. A survey on data-driven network intrusion detection. ACM Computing Surveys (CSUR), 54(9), pp.1-36.

Čisar, P., Popović, B., Kuk, K., Čisar, S.M. and Vuković, I., 2022. Machine Learning Aspects of Internet Firewall Data. In *Security-Related Advanced Technologies in Critical Infrastructure Protection: Theoretical and Practical Approach* (pp. 43-59). Dordrecht: Springer Netherlands.

Dawadi, B.R., Adhikari, B. and Srivastava, D.K., 2023. Deep learning technique-enabled web application firewall for the detection of web attacks. Sensors, 23(4), p.2073.

Garcia, J.F.C. and Blandon, G.E.T., 2022. A deep learning-based intrusion detection and preventation system for detecting and preventing denial-of-service attacks. IEEE Access, 10, pp.83043-83060.

George, B., 2022. AdaBoost IDS to detect Zero Day attacks and reduce false positives (Doctoral dissertation, Dublin, National College of Ireland).

Gupta, N., Naik, V. and Sengupta, S., 2017, January. A firewall for internet of things. In 2017 9th International Conference on Communication Systems and Networks (COMSNETS) (pp. 411-412). IEEE.

Islam, U., Muhammad, A., Mansoor, R., Hossain, M.S., Ahmad, I., Eldin, E.T., Khan, J.A., Rehman, A.U. and Shafiq, M., 2022. Detection of distributed denial of service (DDoS) attacks in IOT based monitoring system of banking sector using machine learning models. Sustainability, 14(14), p.8374.

Jha, J. and Ragha, L., 2013. Intrusion detection system using support vector machine. International Journal of Applied Information Systems (IJAIS), 3, pp.25-30.

Mayuranathan, M., Saravanan, S.K., Muthusenthil, B. and Samydurai, A., 2022. An efficient optimal security system for intrusion detection in cloud computing environment using hybrid deep learning technique. Advances in Engineering Software, 173, p.103236.

Momand, A., Jan, S.U. and Ramzan, N., 2023. A systematic and comprehensive survey of recent advances in intrusion detection systems using machine learning: Deep learning, datasets, and attack taxonomy. Journal of Sensors, 2023(1), p.6048087.

Mughaid, A., AlZu'bi, S., Hnaif, A., Taamneh, S., Alnajjar, A. and Elsoud, E.A., 2022. An intelligent cyber security phishing detection system using deep learning techniques. Cluster Computing, 25(6), pp.3819-3828.

Musa, M.O. and Victor-Ime, T., 2023. Improving Internet Firewall Using Machine Learning Techniques. *American Journal of Computer Science and Technology*, *6*(4), pp.170-179.

Shaheed, A. and Kurdy, M.B., 2022. Web application firewall using machine learning and features engineering. Security and Communication Networks, 2022(1), p.5280158.

Shang, Y., 2024. Detection and Prevention of Cyber Defense Attacks using Machine Learning Algorithms. Scalable Computing: Practice and Experience, 25(2), pp.760-769.

Wu, T., Fan, H., Zhu, H., You, C., Zhou, H. and Huang, X., 2022. Intrusion detection system combined enhanced random forest with SMOTE algorithm. EURASIP Journal on Advances in Signal Processing, 2022(1), p.39.

Jeon, D. and Tak, B., 2022. BlackEye: automatic IP blacklisting using machine learning from security logs. Wireless Networks, 28(2), pp.937-948.

Gregorczyk, M., Żórawski, P., Nowakowski, P., Cabaj, K. and Mazurczyk, W., 2020. Sniffing detection based on network traffic probing and machine learning. IEEE Access, 8, pp.149255-149269.

Ongun, T., Spohngellert, O., Miller, B., Boboila, S., Oprea, A., Eliassi-Rad, T., Hiser, J., Nottingham, A., Davidson, J. and Veeraraghavan, M., 2021, October. PORTFILER: port-level network profiling for self-propagating malware detection. In 2021 IEEE Conference on Communications and Network Security (CNS) (pp. 182-190). IEEE.

Bhuyan, M.H., Bhattacharyya, D.K. and Kalita, J.K., 2011. Surveying port scans and their detection methodologies. The Computer Journal, 54(10), pp.1565-1581.