Syrian Arab Republic
Higher Institute for Applied Sciences and Technology
Department of Communications
Academic Year 2021/2022

Graduation Project

Prepared for the Bachelor's Degree in Communications Engineering

Utilizing Machine Learning for the Detection of Botnet Attacks Targeting IoT Devices

Prepared by

Hind Adham Makarem

Supervision

Dr. Sameeh Jammoul

Eng. Sleiman Moualla

2022/8/16

Abstract

The tremendous number of Internet of Things (IoT) applications, with their ubiquity, has provided us with unprecedented productivity and simplified our daily life. At the same time, the insecurity of these technologies ensures that our daily lives are surrounded by vulnerable computers, allowing for the launch of multiple attacks via large-scale botnets through the IoT. These attacks have been successful in achieving their heinous objectives. A strong identification strategy is essential to keep devices secured. This project aims to design a real-time multi-class classification model using machine learning and deep learning algorithms to classify botnets using IoT-23 dataset, and have this model to be integrated with an intrusion detection system to detect attacks in network traffic and alert the administrator. After data-Preprocessing four algorithms were applied, namely Random Forest, XGBoost, KNN and LSTM. In the end, the XGBoost algorithm achieved the best scenario compared to the rest of the algorithms, which achieves F1-Score equal to 98.68%. The model achieved this accuracy by taking 3 packets only from each flow. To reduce the complexity of the model, the features was shortened to 11 features after calculating Feature Importance in the model.

Keywords: *botnets, multi-class classification, Machine learning algorithm, Deep learning algorithm, Intrusion detection system.*

Introduction

The continuous evolution of Internet of Things (IoT) devices has led to increased reliance on their services by companies, organizations, and individuals. However, these devices have become vulnerable to attacks by malicious actors seeking to achieve unlawful gains through exploiting security vulnerabilities. This has prompted a growing interest in the security and protection of IoT.

In October 2016 [1], a major Internet outage occurred in the United States, where several key websites in various cities experienced Distributed Denial of Service (DDoS) attacks. This led to the compromise of the DNS service provider Dyn and rendered many websites, including Netflix, Twitter, and Reddit, inaccessible for several hours. It was later discovered that a malicious program called "Mirai" scanned smart cameras and attempted various login passwords. Mirai exploited the relative lack of security in IoT devices, coordinating large groups of these distributed devices into coordinated botnets to execute massive-scale DDoS attacks.

The threat of Mirai persisted in IoT devices, and others started using the original Mirai source code on GitHub to develop new variants like "Okiru" to expand their botnets to a variety of IoT devices. These risks have necessitated innovative approaches in intrusion detection systems to intelligently avoid new attacks, adapting to changes in their behavior using artificial intelligence techniques.

In this project, we aim to develop a real-time detection model capable of classifying various types of IoT botnet attacks using machine learning and deep learning algorithms. The model is trained on the IoT-23 dataset, achieving real-time detection by training on a small number of packets in each network flow. Once trained, the model can be integrated with an intrusion detection system to monitor packet traffic on networks and detect anomalies. The goal is to enhance the security of IoT devices and move them to a new level of vigilance and safety.

Chapter 1

Project Introduction

In this chapter, we get acquainted with the project's objective, elucidating the steps of the work through a detailed flowchart.

1.1 Project Objective

The project aims to design a system capable of real-time detection and classification of IoT botnet attacks, such as Distributed Denial of Service (DDoS), on IoT networks using machine learning and deep learning algorithms.

2.1 Project Description

The project provides insights into the types and structures of IoT botnets, their construction, communication, and the attacks they perform using IoT devices. It also reviews the latest research in intrusion detection systems that rely on artificial intelligence algorithms for detecting attacks on networks in general and IoT networks specifically. The outcome of this study is a system capable of detecting IoT botnet attacks in real-time, along with studying and developing methods to protect against these attacks.

3.1 Work Steps

1. Familiarize with programming tools (Python language, Jupyter notebook, Wireshark tools).
2. Study references to select the dataset.
3. Study references on machine learning and deep learning algorithms applied to detect attacks on IoT networks.
4. Propose a detection model based on the completed literature review.
5. Pre-process the data in dataset to extract features.
6. Prepare the data for input to the model (Data cleaning).
7. Select the most impactful features (Feature Importance).
8. Choose machine learning algorithms based on the completed literature review.
9. Test the algorithms and adopt evaluation metrics, then compare them to choose the best scenario.
10. Compare the final results with previous findings referenced in the literature.

4.1 Project Flowchart

Figure (1-1) shows the proposed model flowchart. After conducting the literature review, we proposed a model for detection based on a dataset containing features at the packet level (Packet-based dataset) derived from the selected IoT-23 dataset. The proposed model achieves real-time attack detection and performs multiclass classification of attacks within the dataset. During the model development, we used three machine learning algorithms: Random Forest, K-Nearest Neighbor, and XGBoost, along with the deep learning algorithm Long Short-Term Memory (LSTM). We compared them based on several criteria and selected the algorithm that provided the best detection accuracy. The flowchart consists of three main sections:

- **Section 1 (Data Pre-processing):** Extracting features from IoT-23 dataset at the packet level and preparing the data for input to the classifier model.
- **Section 2 (Feature Selection):** Choosing the most impactful features.
- **Section 3 (Classifiers Modeling):** Implementing machine learning algorithms discussed earlier to select the best algorithm and scenario for optimal real-time attack detection.

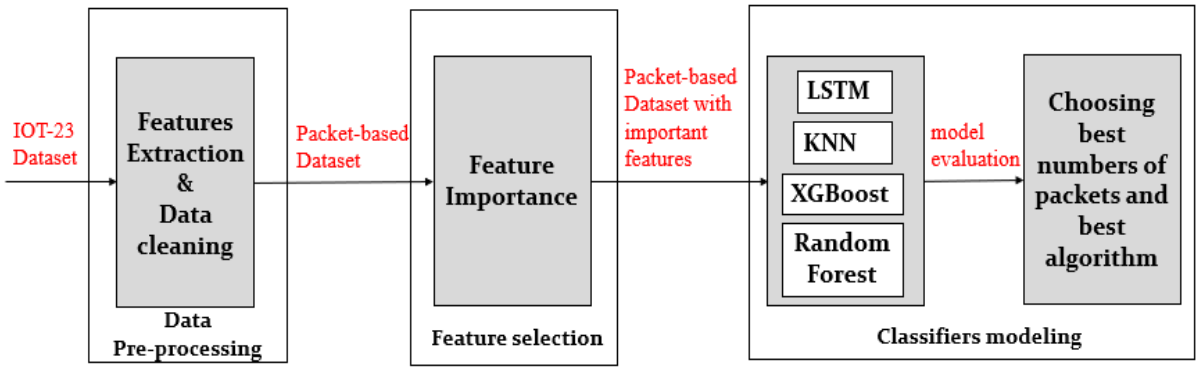

Figure (1-1): Project Flowchart.

5.1 Conclusion

In this chapter, we introduced the project's purpose and its primary objective. We presented the flowchart for the proposed model execution mechanism and discussed the contents of its sections. In the following chapters, we will detail each part of the project flowchart, leading to the application of algorithms and the presentation of results.

Chapter Four

The Proposed Model

In this chapter, we will elaborate in detail on the proposed model based on the selected dataset and the previous works conducted on it.

4.1 Introduction

Intrusion detection models are usually designed using computer programs or systems to prevent suspicious traffic movements. Recently, the integration of machine learning algorithms into these systems has become common due to their ability to process large amounts of information and predict new types of attacks. Proposing a machine learning model capable of achieving optimal detection requires conducting a comprehensive study to review previous works in this field and benefit from the diversity of proposed models. This integration should consider complementing the existing strengths of each model, resulting in a new model that achieves the desired objectives. In this chapter, we will provide detailed justifications for the proposed model based on the previous works conducted on the selected dataset.

4.2 IoT-23 Dataset

4.2.1 Selection of the Dataset

In the process of searching for a training and testing dataset that fulfills the project's objectives, a thorough examination of several datasets was necessary, as the selection process significantly impacts the project. For example, the CIC-DDoS2019 dataset [26] is suitable for the project in terms of the types of network attacks it encompasses, such as PortMap, NetBIOS, LDAP, MSSQL, UDP, UDP-Lag, SYN, NTP, DNS, and SNMP, all of which belong to the DDoS attack category. However, this dataset is not specifically designed for Internet of Things (IoT) networks but rather for general internet networks. Furthermore, upon reviewing the dataset [27] generated by C. D. McDermott, F. Majdani, and A. V. Petrovski using laboratory software tools, although it is designed for IoT networks, it does not accurately represent real-world attacks in certain aspects, leading to unexpected errors in real-world experiments. There is also a dataset called "Bot-IoT" [28], generated by Koroniotis, Nickolaos, Nour Moustafa, Elena Sitnikova, and Benjamin

Turnbull. It is specifically designed for IoT networks and generated using real IoT devices. However, the DDoS attacks in this dataset are classified based on the protocol type rather than the type of network robots that launched the attacks. As for the remaining datasets that meet both requirements (being dedicated to IoT networks and generated using real IoT devices), they were outdated, and all related research concluded with optimal and non-improvable results. Therefore, the search for such a dataset was conducted under the following constraints:

- It should be dedicated to IoT networks.
- It should be generated using real IoT devices, not simulated.
- It should be recent.
- The research conducted on it should be subject to improvement and development.
- It should include variations of network robots (botnets).

Based on these required constraints, the IoT-23 dataset [29] was selected. Table (4-1) summarizes the comparison between the examined datasets and the selected dataset.

Table (4-1): Comparison between the Reviewed Datasets

Dataset	**IoT-specific**	**Real IoT Devices**	**Recent**	**Variation of Botnets**
CIC-DDoS2019	✗	✗	✗	✗
Mirai botnet	✓	✗	✗	✓
Bot-IoT	✓	✓	✗	✗
IoT-23	✓	✓	✓	✓

4.2.2- Introduction to the IoT-23 Dataset

The IoT-23 dataset was published in 2020 by Parmisano, Garcia, and Erquiaga, and it was developed at the Stratosphere Lab at the CTU University in the Czech Republic [29]. It consists of a traffic dataset comprising 20 sub-datasets of malicious software and three sub-datasets of benign traffic. Its purpose is to provide a large dataset of real-world IoT malware infections and benign IoT traffic for researchers to develop machine learning algorithms. The benign network traffic was collected using three separate IoT devices. These three devices are real devices, not emulated, and they are: Somfy door lock, Philips Hue, and Amazon Echo [29]. The attacks in the IoT-23 dataset are classified into nine types [30], as shown in the figure below.

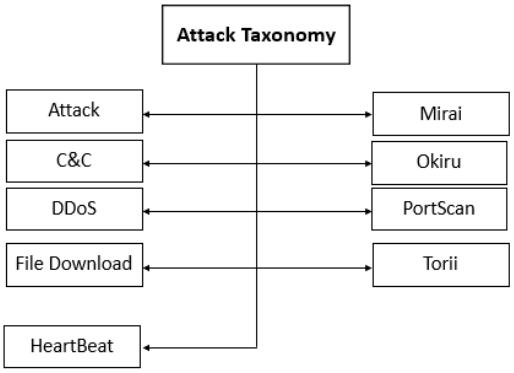

Figure (4-1): Available Categories in the IoT-23 Dataset [30].

The executed attacks include various bots such as Mirai and Torii (as shown in Table (2-4)). In addition to the original packet captures, network flows generated by Zeek/Bro IDS are also available along with their labels.

Table (4-2): Explanation of the Categories in the IoT-23 Dataset [29].

Label	**Semantic Meaning**
Attack	Represents an unknown type of attack from an infected host to a clean host.
Benign	Indicates no suspicious or harmful activity.
C&C	The infected device is connected to a C&C server.
DDoS	A distributed denial of service attack is being executed by the infected device, and this is detected based on the volume of traffic directed to the same IP address.
FileDownload	Indicates that a malicious file has been downloaded to the infected host.
HeartBeat	The sent packets on this connection are used to track the infected host by the command and control (C&C) server.
Mirai	Refers to the connection having characteristics of the Mirai botnet family.
Okiru	\Indicates that the connection has characteristics of the Okiru botnet family.
PartOfA HorizontalPortScan	These connections are used to perform a horizontal port scan to gather information for further attacks..
Torii	Indicates that the connection has characteristics of the Torii botnet family.

The IoT-23 dataset provides two downloadable versions: the first version, called the Light version, has a size of 43GB and only contains Log.Labeled files, without the PCAPs files. The second version includes PCAPs files, allowing users to extract new features, and its size is 100GB. In the proposed model, the larger version was used. However, due to limited computational capacity to process and train a dataset of 100GB, only a portion of it was utilized. The files used are listed in Table (4-3).

Table (4-3): The datasets used in the IoT-23 test and training set [29].

Scenario	**Malware**	**Packets**	**Zeek Flows**
CTU-IoT-Malware-Capture-1-1	Hide N' Seek	1,686,000	1,008,749
CTU-IoT-Malware-Capture-3-1	Muhstik	496,000	156,104
CTU-IoT-Malware-Capture-4-1	Benign-Philips HUE	21,000	461
CTU-IoT-Malware-Capture-8-1	Hakai	23,000	10.404
CTU-IoT-Malware-Capture-21-1	Torii	50,000	3,287
CTU-IoT-Malware-Capture-34-1	Mirai	233,000	23,146
CTU-IoT-Malware-Capture-35-1	Mirai	46,000,000	10,447,796
CTU-IoT-Malware-Capture-36-1	Okiru	13,000,000	13,615,107
CTU-IoT-Malware-Capture-42-1	Trojan	24,000	4,427
CTU-IoT-Malware-Capture-44-1	Mirai	1,309,000	238
CTU-IoT-Malware-Capture-48-1	Mirai	13,000,000	3,394,347
CTU-IoT-Malware-Capture-49-1	Mirai	18,000,000	5,410,562
CTU-IoT-Malware-Capture-60-1	Gagfyt	271,000,000	3,581,029

4.3 Previous Studies on the IoT-23 Dataset

In this paragraph, we will discuss previous works conducted on the IoT-23 dataset, and examine the research papers published thereafter. In all the following works, machine learning algorithms were employed to build various models for detecting network anomalies or classifying attacks within the IoT-23 dataset.

4.3.1 Model Number 1

Vibekananda Dutta, Michał Choras, Marek Pawlicki, and Rafał Kozik published a research paper in 2020 [31]. This paper presents an aggregate method that leverages different models of deep learning algorithms such as Deep Neural Networks (DNN), Long Short-Term Memory neural networks (LSTM), and meta-classifier classifiers following the stacked generalization principle, as illustrated in Figure (4-2). In the proposed approach, the anomaly detection in the network was performed in two stages. In the first stage, data preprocessing was carried out, and the Deep Sparse AutoEncoder (DSAE) tool was utilized for feature engineering. In the second stage, LSTM and DNN algorithms were combined for classification. The effectiveness of the proposed approach was tested on heterogeneous test and training sets such as LITNET-2020 and NetML-2020, in addition to the testing and training dataset collected within the Internet of Things environment, namely IoT-23.

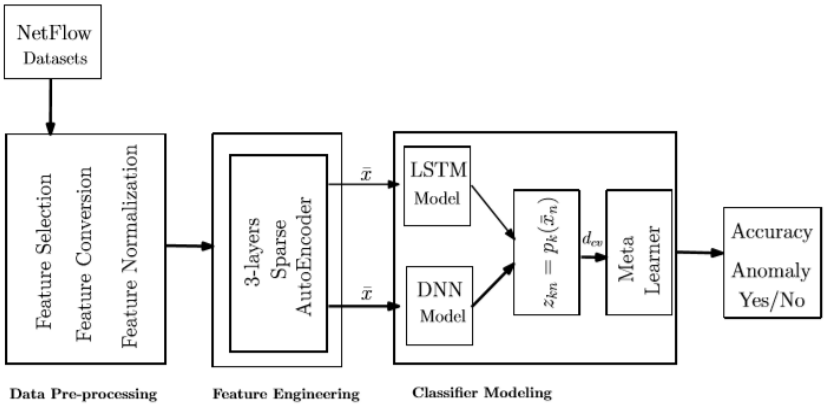

Figure (4-2): Box plot for the proposed model in Research Paper Number 1 [31].

The proposed approach outperforms the latest individual classifiers and meta-classifiers such as random forest and support vector machine. Table (4-4) illustrates the results related to the IoT-23 dataset.

Table (4-4): Results of the proposed model in Research Paper Number 1 [31].

Method	Feature Engg.	Pr	Re	FPR	F1-Score	MCC
Random Forest [27]	AE	0.92	0.89	1.22	0.896	0.87
DNN	DSAE	**1.00**	0.89	0.23	0.87	0.98
LSTM	DSAE	**1.00**	0.92	0.19	0.95	0.99
Stacked (proposed)	DSAE	**1.00**	**0.95**	**0.13**	**0.98**	**0.99**

Despite the good results and high accuracy achieved by this research in detection, real-time detection was not considered because the model was applied to flows existing in the datasets after the attack had ended. Additionally, the detection is of the Anomaly detection type, meaning that when detecting network anomalies, they will be handled regardless of the type of anomaly or the type of attacks that occurred on the network.

4.3.2- Model Number 2

Rafał Kozik, Marek Pawlicki, and Michał Choraś also published a research paper in 2020 [32]. This paper proposes a method that supports real-time detection of aggregated communication flows in time windows (here the time windows are 3 minutes long) from PCAP files available in the IoT-23 dataset. It efficiently processes a massive amount of data with a low-memory footprint, as illustrated in Figure (4-3). In this approach, these windows are input into a pre-trained model for anomaly detection, which ultimately produces detection outputs (benign for normal traffic or anomaly for traffic containing suspicious patterns). This model uses a classifier based on a transformer's encoder followed by a feed-forward neural network. The proposed method was compared with other classical machine learning algorithms in detailed experiments conducted on the IoT-23 dataset.

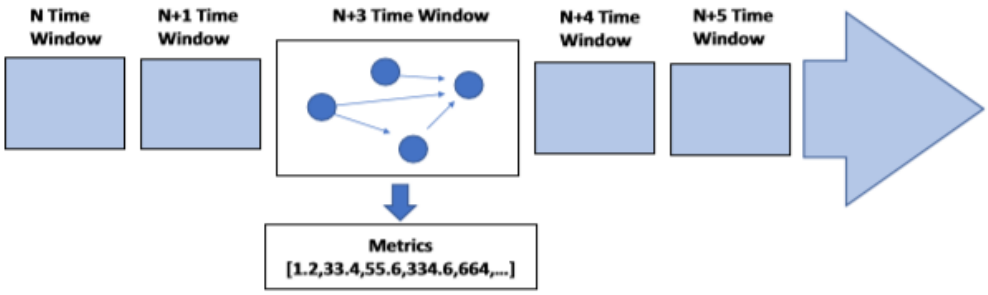

Figure (4-3): Time windows in the proposed model in Research Paper Number 2 [32].

Results for the F1-Score metric are presented in Figure (4-4). The proposed approach was compared with various common machine learning techniques, such as decision tree and classifier ensembles, one being AdaBoost, and the other being Random Forest.

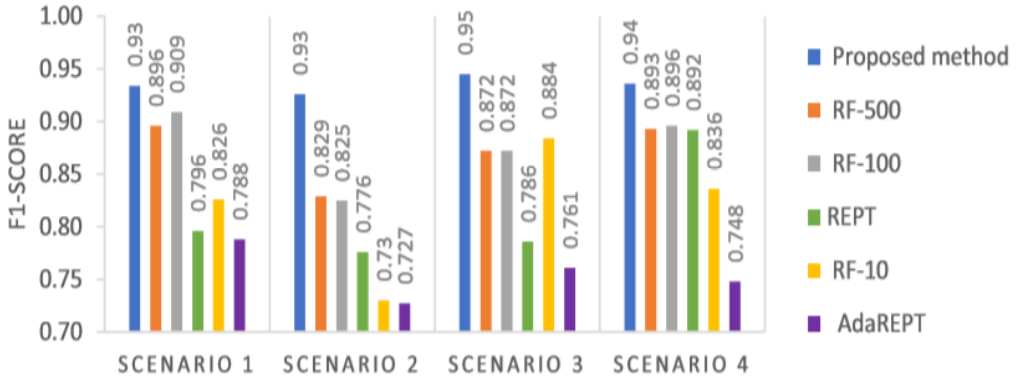

Figure (4-4): Results of the proposed model in Research Paper Number 2 [32].

We observe the superiority of the proposed approach over the previously discussed machine learning techniques in all scenarios. However, despite the proposed approach providing real-time detection, the detection time is in the order of minutes (3 minutes reading for each window, excluding processing time and result display). Additionally, the detection is of the Anomaly detection type, meaning that when detecting, all types of attacks will be handled in the same way.

4.3.3- Model Number 3

IMTIAZ ULLAH and QUSAY H. MAHMOUD published a research paper in 2021 [33]. The research focused on designing and developing a new model for intrusion detection based on anomaly for Internet of Things (IoT) networks. The paper proposed a model using a convolutional neural network to create a multi-class classification model. The proposed model was implemented using 1D, 2D, and 3D convolutional neural networks. The model was applied to the BoT-IoT, IoT Network Intrusion, MQTT-IoT-IDS2020, and IoT-23 datasets, utilizing transfer learning for binary and multi-class classification using a pre-trained multi-layered neural network model. The proposed binary and multi-class models achieved high accuracy for all performance metrics compared to current deep learning applications. Table (4-5) illustrates the results for the IoT-23 dataset after implementing the proposed model.

Table (4-5): Results of the proposed model in Research Paper Number 3 [33].

Model	CNN1D				CNN2D				CNN3D			
Class	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score
Normal	99.97	99.99	99.91	99.95	99.98	99.96	99.96	99.96	99.92	99.83	99.89	99.86
DDoS	99.99	99.98	99.98	99.98	99.99	99.96	99.98	99.97	99.99	99.95	99.98	99.97
Attack	100.00	100.00	100.00	100.00	100.00	99.55	100.00	99.77	100.00	99.55	100.00	99.77
Mirai	99.99	98.29	99.57	98.93	99.99	99.34	99.09	99.21	99.99	99.34	99.05	99.19
File Download	99.98	97.60	99.81	98.69	99.92	90.38	99.15	94.56	99.92	89.77	99.15	94.23
HeartBeat	99.99	99.69	99.85	99.77	99.93	99.63	94.56	97.03	99.87	98.18	91.36	94.65
C&C	99.99	99.98	99.98	99.98	99.99	99.96	99.96	99.96	99.99	99.97	99.97	99.97
Torii	100.00	99.99	100.00	99.99	99.99	99.99	99.99	99.99	99.99	99.99	99.99	99.99
Port Scan	99.99	99.999	99.99	99.99	100.00	100.00	100.00	100.00	99.99	99.99	99.99	99.99
Okiru	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.000	99.99	100.00	99.99

The proposed model in this research demonstrated high accuracy compared to contemporary classification strategies and deep learning applications. However, real-time detection was not taken into consideration.

4.3.4- Model Number 4

N. ABDALGAWAD, A. SAJUN, Y. KADDOURA, I. A. ZUALKERNAN published a research paper in 2022 [34]. This paper suggests that generative deep learning methods, such as Adversarial Autoencoders (AAE) and Bidirectional Generative Adversarial Networks (BiGAN), can be used for detecting network-based attacks. Generative deep learning models were trained to detect various attacks, such as DDoS and different types of botnets like Mirai, Okiruk, and Torii, using over 1.8 million network flows for training. The model outperformed traditional machine learning techniques like random forest, as shown in Table (4-6). Both AAE and BiGAN-based models achieved an F1 score of 0.99. BiGAN was also trained to detect unknown attacks, achieving an F1-Score from 0.85 to 1. However, the model does not support real-time detection.

Table (4-6): Results of the proposed model in Research Paper Number 4 [34].

Model	Average Macro F1 Scores	Std
KNN	0.6019	0.0050
RF	0.5518	0.0055
AAE + KNN	0.9743	0.0182
BiGAN + KNN	0.9741	0.0186

4.3.5- Model Number 5

Michael Austin published a research paper in 2021 [35]. The research utilized PCAP files and network path analysis tools to develop a detection model. Features were extracted at the packet level, including time of day, history, protocol, and count of origin bytes sent.

The experiments used the IoT-23 dataset and four machine learning algorithms: Random Forest, Support Vector Machine, Naïve Bayes, and a linear classifier with Stochastic Gradient Descent as an optimizer. Performance metrics used for evaluation were Accuracy, Precision, Recall, and F1-Score. Results in Table (4-7) show that Random Forest performed the best in all performance metrics except for recall. The Naïve Bayes classifier ranked second. The features deemed most important by Random Forest for row classification were time, history, protocol, responding IP address, and count of original IP bytes. The research did not use optimized machine learning algorithms like Adaboost or neural networks, leaving room for innovation in this space. Due to the inclusion of packet captures, future research may involve extracting additional fields not analyzed by Zeek.

Table (4-7): Results of the proposed model in Research Paper Number 5 [35].

F-1	**Mean**	**Median**	**Variance**	**IQR**
Random Forest	97.39%	97.41%	3.2197E-07	0.001201
Naïve Bayes	94.49%	94.50%	1.8005E-08	0.000267
Linear SVM	92.35%	92.33%	8.1935E-08	0.000603
Stochastic Gradient Descent	94.44%	94.41%	1.9727E-07	0.000753

The study can be summarized in Table (4-8).

Table (4-8): Comparison between the reviewed previous works and the proposed model in the project.

Model Number	Previous Proposed Models	Multiclass Classification	Machine Learning and Deep Learning Algorithms	Packet-Based Features	Real-time Detection
1	Aggregative method using various models for deep learning algorithms (DNN, LSTM, Meta classifier) to create an anomaly detection model.	✗	✓	✗	✗
2	Incorporating a time window that efficiently processes a massive amount of data with a low-memory footprint in an anomaly detection model to achieve real-time detection.	✗	✗	✗	✓
3	Creating and implementing a multiclass classification model using convolutional neural networks in 1D, 2D, and 3D.	✗	✗	✓	✗
4	Using deep learning algorithms AEE and BiGAN to detect network-based attacks.	✗	✓	✗	✗
5	Utilizing PCAP files and network path analysis tools to develop an anomaly detection model in real-time.	✓	✓	✗	✓
Proposed Model	Utilizing PCAP files in IoT-23 to create a packet-level test set and inputting windows of packets into a multiclass classification model to achieve real-time detection.	✓	✓	✓	✓

4.4- Proposed Model

We propose a model for real-time multiclass classification of IoT-23 dataset attacks, where the model's input is a window taken from the beginning of each traffic flow. This window contains a certain number N of packets, which will be chosen optimally at the end of this project for optimal detection. To implement the proposed model, a test and training set must be created at the packet level (Packet-based dataset) derived from the selected IoT-23 test and training set after initial preprocessing of its data and extracting features at the packet level (Packet-based features). The most significant stage in this model is the Data Pre-processing stage, illustrated in Figure (4-5), consisting of several sub-stages:

a- Packet-based Feature Extraction.

b- Reshaping rows and columns to obtain readings for packets within windows.

c- Preparing data for entry into classifiers (Data Cleaning).

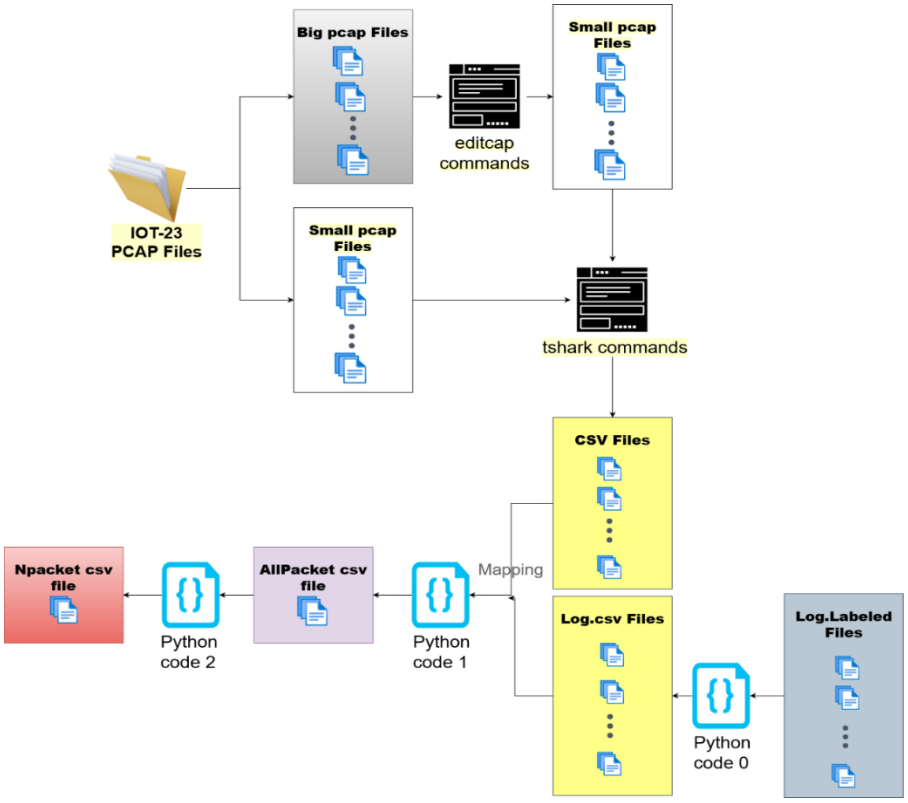

Figure (4-5): Pre-processing operation to access the dataset derived from IoT-23 dataset.

4.4.1 Packet-based Feature Extraction

IoT-23 consists of 23 sets of Capture files, 20 sets containing data for various attack types and three sets containing benign data. Each set includes at least two files:

- **pcap file:** A raw data file for the packet set captured from the network.
- **log.labeled file:** Contains flow-based features extracted from the first file (pcap file) with the classification (Label) for each flow.

In the proposed model, new features not available in the log.labeled file (features listed in Table (4-9)) will be used. These features will be directly extracted from pcap files using the tshark tool in Wireshark with the following command:

```
tshark -r pcap_file_path -T fields -e features_name > csv_file_path
```

Table (4-9): Feature names selected in the proposed model.

Field name in the command	field name in Wireshark
ip.len	Total Length
ip.id	Identification
ip.flags	Flags
ip.ttl	Time to Live
ip.proto	Protocol
ip.checksum	Header Checksum status
eth.src	Ethernet Source
ip.src	IP Source Address
ip.dst	IP Destination Address
udp.srcport	UDP Source Port
udp.dstport	UDP Destination Port
udp.length	UDP Length
udp.checksum	UDP checksum
eth.dst	Ethernet Distination
udp.time_relative	UDP time since first frame
udp.time_delta	UDP time since previous frame
tcp.dstport	TCP Destination Port
tcp.srcport	TCP Source Port
frame.number	Frame Number
frame.time	Arrival Time

We will later use the log.labeled file to identify the class or type of the corresponding flow for these features. We executed the previous command on a set of 35 pcap files, storing all of them in independent CSV files, as illustrated in the flowchart in Figure (4-6).

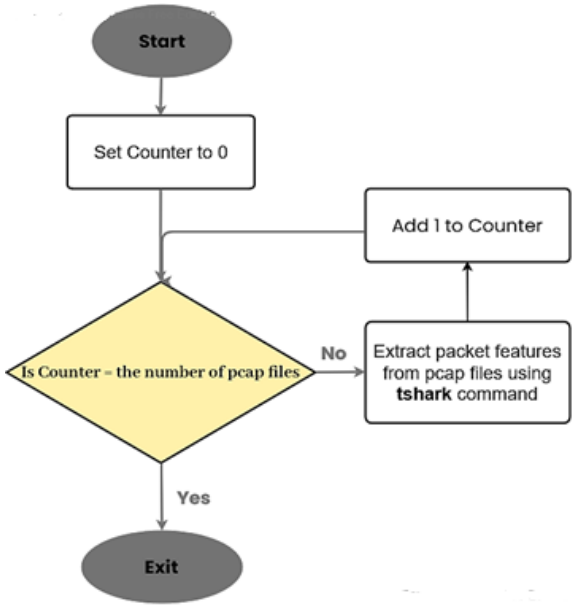

Figure (4-6): Flowchart of the packet-level feature extraction process using the tshark tool.

However, executing those commands on large-sized files is not feasible (sizes larger than 0.5G) as it requires high computational power. To address this issue, we utilized the editcap tool in Wireshark to split large-sized files into smaller files based on the number of packets, as follows:

```
editcap -c Number_of_packets   big_pcap_file_path    small_pcap_files_path
```

After splitting all the large-sized files, we obtained 1000 pcap files and converted all these files into CSV files using the tshark tool. After completing the Wireshark program usage, we obtained files containing packet-level features but were unclassified and did not contain a Label column.

4.4.2- Reformatting Rows and Columns to Obtain Readings for Packets within Windows

a- Labeling the rows at the packet level

After extracting packet-level features from within the pcap files in IoT-23, we wrote another Python script (Code 1 Python in Figure (4-5)) using the Python program. This script is responsible for adding labels (Labeling) to each packet group belonging to a specific flow. We relied on the

log.labeled files within the IoT-23 dataset, as previously discussed. These files contain flow-based features along with the classification (Label) for each flow, as illustrated in the flowchart shown in Figure (4-7). The process was conducted through a mapping operation between each flow in the log.labeled file and the set of packets belonging to it. The condition for a packet to belong to a specific flow is that they share the same following features:

1. Destination Address.
2. Source Address.
3. Source TCP/UDP port number.
4. Destination TCP/UDP port number.

However, during this process, we encountered another problem. The file type .log.labeled does not allow proper reading within the Python program. Therefore, we needed to apply another Python code (Code 0 Python in Figure (4-5)) to convert these files into CSV files for accurate reading and utilization in the Labeling process without encountering errors.

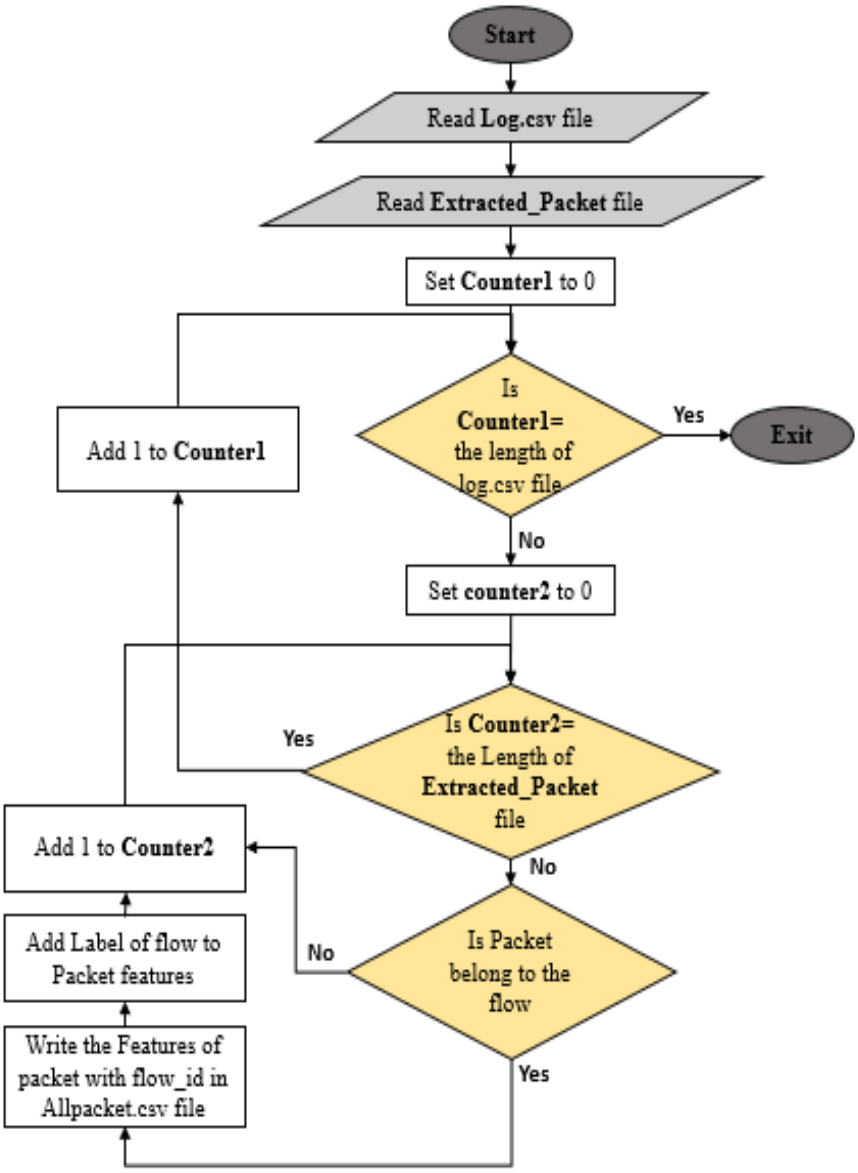

Figure (4-7): Flowchart of the labeling process.

After the matching process, we obtained a CSV file as shown in Table (4-10).

Table (4-10): Derived dataset after labeling.

Flow1-id	Flow1-Packet1 features	Flow1-Label
Flow1-id	Flow1-Packet2 features	Flow1-Label
⋮	⋮	⋮
Flow1-id	Flow1-PacketN_1 features	Flow1-Label
Flow2-id	Flow2-Packet1 features	Flow2-Label
Flow2-id	Flow2-Packet2 features	Flow2-Label

⋮	⋮	⋮
Flow2-id	Flow2-PacketN_2 features	Flow2-Label
⋮	⋮	⋮
FlowM-id	FlowM-Packet1 features	FlowM-Label
FlowM-id	FlowM-Packet2 features	FlowM-Label
⋮	⋮	⋮
FlowM-id	FlowM-PacketN_M features	FlowM-Label

This file contains all flows and all packets belonging to them. Flows may vary, as there are flows consisting of a single packet and flows containing thousands of packets. The primary goal of extracting packet-level features is to create a model that supports real-time detection based on a small number of packets. Therefore, we wrote another Python script (Code 2 Python in Figure (4-5)) to take N packets from each flow and reshape the rows so that each row contains N packets from each flow, as shown in Table (11-4) .

Table (4-11): The final form of the dataset derived from the original IoT-23 dataset.

Flow1-Packet1 features	Flow1-Packet2 features	…	Flow1-PacketN features	Flow1-Label
Flow2-Packet1 features	Flow2-Packet2 features	…	Flow2-PacketN features	Flow2-Label
⋮	⋮	⋮	⋮	⋮
FlowM-Packet1 features	FlowM-Packet2 features	…	FlowM-PacketN features	FlowM-Label

b- Handling special cases

The number of flows that achieve Packet count < 3 represents a significant portion of the total number of flows in the training and testing dataset. For this reason, we retained all flows that achieve Packet count < N by adding zeros in the remaining N-Packet count fields. This is necessary as omitting these flows may result in a significant loss of information and could lead to underfitting of the model. Figure (4-8) illustrates the flowchart of the written code (Code 2 Python in Figure (4-5)), and we note that the data cleaning process occurred in this section.

Figure (4-8): Flowchart of the row and column reshaping process with data cleaning.

The main goal of the proposed model is to choose the value of N to be small enough to achieve real-time detection (online detection) and large enough to achieve optimal detection. Therefore, we initially took a window containing Nmax=20 packets from the beginning of each flow and then gradually took a window containing fewer packets, reaching one packet from the beginning of each flow (N=1).

4.4.3- Data Preparation for Input to Classifiers (Data Cleaning)

Data entering classifiers must be strictly numerical because machine learning classifiers cannot be trained on alphanumeric input or character sequences. Additionally, errors may occur in the presence of empty or infinite values in the data within the training and testing dataset. Therefore,

it is essential to prepare the data before entering the classifier to ensure proper processing and training. This was achieved as follows:

1. Initially, we deleted columns that were not explicitly necessary in the learning model: eth.dst, frame.time, eth.src, ip.id, ip.src, and ip.dst. This was done to avoid overfitting.
2. We converted columns containing data represented in hexadecimal format to columns in decimal representation. The columns subjected to this conversion included ip.flags, ip.checksum, and udp.checksum.
3. We mapped columns containing string-type data to an array of integers equal to the number of unique data (Distinct value) in the intended column. This process is called mapping.
4. For columns containing empty values, each column was treated uniquely based on its type. Regarding columns tcp.dstport, tcp.srcport, udp.dstport, and udp.srcport, we replaced all empty values with zero values to avoid revealing a known port number. For other columns, empty values were replaced with the median of the corresponding column, and infinite values were treated in the same manner.
5. We applied the Zscore function to all data except the Label column in the table to ensure that values belong to the range [-1,1].
6. Special characters such as (-, *, _, #, (,), space) were removed, and some uppercase letters in the Label column were replaced with lowercase letters to standardize categories. This was necessary as categories might be written differently in various folders (e.g., benign, Benign, -benign-).
7. We deleted empty columns and rows.

5.4- Conclusion

In this chapter, we elaborated in detail on the data preprocessing process. Now, we have a training and testing dataset Npacket.csv derived from the IoT-23 dataset. The next step involves applying various machine learning algorithms to this dataset and discussing the results to find the optimal algorithm and the best value for N (the number of packets taken from the beginning of each flow) that achieves optimal and real-time detection.

Chapter Five

Practical Application and Results

In this chapter, we will present the results of applying various algorithms to the training and testing dataset. Subsequently, we will compare the results of the proposed model with those of previous models at the end of the chapter.

5.1 Introduction

Any machine learning model requires a variety of tools, libraries, and high-level programming languages to simplify the building and application process. Additionally, applying and evaluating the model necessitates suitable performance metrics for the type of problem presented since the importance of performance metrics varies depending on the task of the proposed model. In this chapter, we will discuss the essential tools used for building and applying the model, the key performance metrics used for result analysis, and, at the end of the chapter, we will present the results of applying various algorithms to the training and testing dataset Npacket.csv to choose the best scenario for achieving optimal real-time detection.

5.2 Performance Metrics Adopted in Evaluating the Proposed Model

The evaluation process of the proposed model adopted several metrics such as Precision, Recall, F1-Score, Accuracy, and the Area Under the Curve (AUC). All these metrics are related to several concepts illustrated in Table (5-1).

Table (5-1): Parameters Used in Calculating Performance Metrics [36]

Prediction	Actual value	Type	Symbol	Explanation
1	1	True Positive	TP	Predicted Positive and was Positive
0	0	True Negative	TN	Predicted Negative and was Negative
1	0	False Positive	FP	Predicted Positive but was Negative
0	1	False Negative	FN	Predicted Negative but was Positive

Figure (5-1) illustrates the hierarchical sequence of Precision, Recall, and F1-Score. F1-Score is highlighted as the most crucial metric among others since it is related to both Precision and Recall.

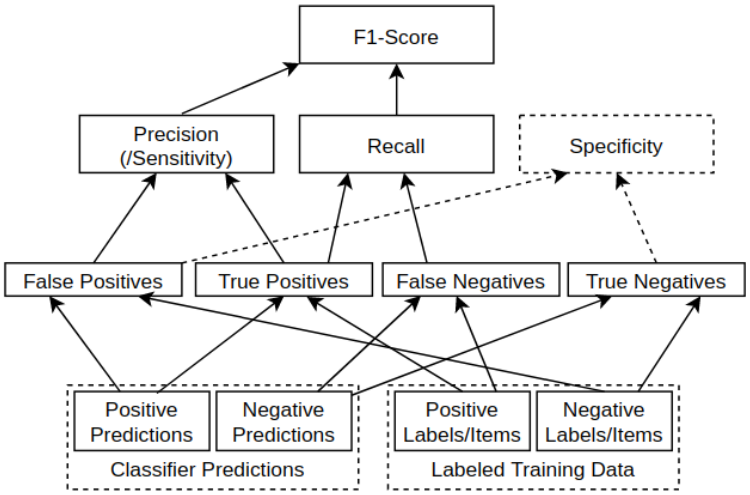

Figure (5-1): Hierarchical Sequence of Precision, Recall, and F1-Score [36].

5.2.1 Accuracy

Accuracy represents the ratio of correct predictions to the total number of predictions [36] and is given by the following relationship:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

This metric is useful only when the sample is balanced (i.e., the number of samples belonging to the first class is equal to or close to the number of samples belonging to the other class). Since our samples are unbalanced, accuracy was calculated for the best-case scenario only and was not adopted in the results discussion.

5.2.2 Area Under Curve

Abbreviated as AUC, the area under the curve is considered one of the classifier evaluation metrics. It is related to other concepts such as Sensitivity and Specificity.

Sensitivity (TP Rate) : The ratio of true positive points detected by the classifier to the total number of positive points [36]. It is given by the following relationship:

$$TP\ Rate = \frac{TP}{TP + FN} \quad (2)$$

Specificity (FP Rate) : Represents the ratio of negative points considered positive by the classifier when they are not to the total number of negative points in the sample [36]. It is given by the following relationship:

$$FP\ Rate = \frac{FP}{FP + TN} \tag{3}$$

Both sensitivity and specificity values fall within the range [0,1]. AUC represents the area under the curve resulting from plotting sensitivity against specificity, as illustrated in Figure (5-2).

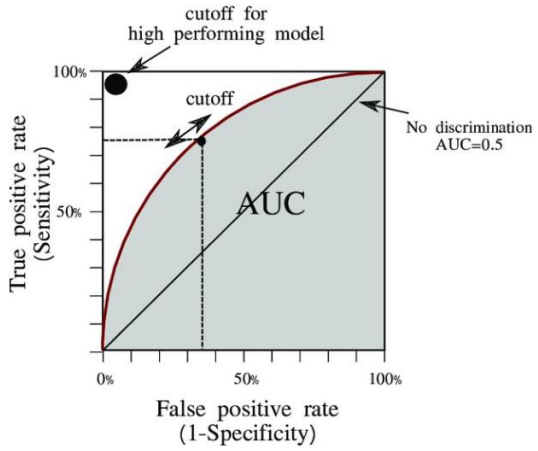

Figure (5-2): Area Under the Curve (AUC) Metric [36].

5.2.3 Recall

Recall represents the sensitivity mentioned earlier and is given by the following relationship:

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

5.2.4 Precision

Precision is the ratio of the number of positive samples that the classifier correctly predicted to the total number of samples that the classifier declared as positive [36]. It is given by the following relationship:

$$Precision = \frac{TP}{TP + FP} \tag{5}$$

5.2.5 F1-Score

F1-Score is the harmonic mean of both Precision and Recall [36]. It is given by the following relationship:

$$F1\ Score = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} \tag{6}$$

This metric achieves a balance between the Precision and Recall metrics, and the higher it is, the better the model's performance.

5.2.5 Confusion Metrix

The confusion matrix is a table composed of rows and columns equal to the number of classes to be classified. It contains information about the number of false positives (FP), false negatives (FN), true positives (TP), and true negatives (TN) for each class, as shown in Figure (5-3). This allows for a more detailed analysis than accuracy since accuracy can lead to misleading results if the dataset is imbalanced.

		PREDICTED classification			
	Classes	a	b	c	d
ACTUAL classification	a	TN	FP	TN	TN
	b	FN	TP	FN	FN
	c	TN	FP	TN	TN
	d	TN	FP	TN	TN

Figure (5-3): Confusion Matrix in the Case of Multiclass Classification[36] .

5.3 Tools Used in Model Implementation

5.3.1 Python Language

We utilized the Python language to implement classifiers on the training and testing dataset Npacket.csv. Python is known for its strength in research fields, especially in artificial intelligence and deep learning, as it has numerous libraries suitable for AI, such as Numpy and Scipy for numerical computation, Sklearn for machine learning, Matplotlib for data visualization, and more.

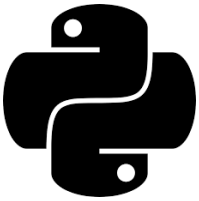

Figure (5-4)
Python Logo

We used Python version 3.8.0 within the deep learning server.

5.3.2 Jupyter Notebook Environment

Jupyter Notebook is an integrated development environment (IDE) that supports the Python language. It operates as a web application, allowing interaction through a web browser. This environment facilitates the creation and training of neural networks by enabling the division of code into separate units that can be executed independently and in any order. It also allows us to display results for each unit easily and efficiently, especially when the output is visual.

Figure (5-5)
Jupyter Logo

5.3.3 Tensorflow Environment

Tensorflow is a framework tailored for machine learning applications such as neural networks. It operates using the Python language and provides the necessary tools and libraries for implementing machine learning applications, whether for research or production.

Figure (5-6)
Tensorflow Logo

5.3.4 Keras Library

Keras is an open-source library that provides a high-level API for interacting with various machine learning frameworks, such as the Tensorflow framework. It facilitates easy creation and training of neural networks. The library also offers pre-trained networks, such as VGG16.

Figure (5-7)
Keras Logo

5.3.5 Numpy Library

Numpy is a Python library supporting the handling and processing of large, multidimensional arrays. It provides a wide range of high-level mathematical functions designed for working with arrays.

Figure (5-8)
NumPy Logo

5.4 Algorithms Applied to the Model

The primary objective of the project was to apply deep learning algorithms. However, after the sample collection and feature extraction processes over two full months, a substantial amount of data was not obtained as expected due to limited computational capacity. It is common for deep learning algorithms to exhibit suboptimal performance with a small number of samples. Upon reviewing previous research on the selected IoT-23 dataset, it was observed that machine learning algorithms still yield good and indispensable results in this field. This is evident in Model 5 (Section 5.4.3), where deep learning algorithms were not employed. The random forest algorithm outperformed other utilized algorithms. No enhanced algorithms from reinforcement learning were used. Consequently, the decision was made to employ the random forest algorithm and the extreme gradient boosting algorithm (XGBoost), known for their effectiveness in enhancing boosting algorithms. Additionally, the k-nearest neighbors algorithm (KNN) was employed due to its simplicity. Regarding the long short-term memory neural network (LSTM), it was used to demonstrate that deep learning algorithms require a large number of samples to achieve more detailed results compared to machine learning algorithms. Default parameters were used for each algorithm during the training process, with minor modifications to some parameters (the changed parameters are specified in the tables 5-2, 5-3, and 5-4).

Table (5-2): Default Parameters for the KNN Algorithm.

Parameter	Default Value
n_neighbors	3
weights	'uniform'
algorithm	'auto'
leaf_size	30
metric	'minkowski'
n_jobs	None

The default value for the number of neighbors in Table (5-2) is 6, but it was changed to 3 to increase classification accuracy.

Table (5-3): Default Parameters for the XGBoost Algorithm.

Parameter	Default Value
learning_rate	0.1
n_estimators	100
objective	'multi:softmax'
num_class	6
boostser	’gbtree’
n_jobs	1

The default value for the Objective parameter in the XGBoost classifier is 'binary:logistic,' but this value is suitable for binary classification. In the proposed model, a change was made to 'multi:softmax' since softmax is suitable for multiclass classification tasks. The num_class parameter represents the number of classes in the model, which, in our case, is 6.

Table (5-4): Default Parameters for the Random Forest Algorithm.

Parameter	DefaultValue
n_estimators	100
criterion	'gini'
max_depth	None
min_samples_split	2
min_samples_leaf	1
min_weight_fraction_leaf	0.0
max_features	'sqrt'
max_leaf_node	Flase
min_impurity_decrease	0.0

For the LSTM algorithm, Table (5-5) illustrates the layers used for training with dimensions and the number of neurons in each layer, along with the activation function used. For the optimizer of the examples (optimizer), Adam was chosen, considered one of the best optimization functions, and the categorical_crossentropy error function, suitable for multiclass classification.

Table (5-5): Parameters Used in the LSTM Algorithm.

Layer (type)	Output shape	Param#	Activation Function
Lstm (LSTM)	(None, 100)	80000	none
dropout (Dropout)	(None, 100)	0	none
dense (Dense)	(None, 100)	10100	Relu

dense_1 (Dense)	(None, 6)	606	Softmax

5.5 Applying Algorithms and Presenting Results

The previous algorithms were applied to the dataset NPacket.csv under the following conditions:

- Taking 1 packet from the beginning of each flow (N=1).
- Taking 10 packets from the beginning of each flow (N=10).
- Taking 15 packets from the beginning of each flow (N=15).
- Taking 2 packets from the beginning of each flow (N=2).
- Taking 20 packets from the beginning of each flow (N=20).
- Taking 3 packets from the beginning of each flow (N=3).
- Taking 4 packets from the beginning of each flow (N=4).
- Taking 5 packets from the beginning of each flow (N=5).

The algorithms were applied to two different scenarios, differing in the deletion of certain features, as explained during the presentation of the results.

5.5.1 Scenario 1

Applying algorithms with port numbers

We applied the four algorithms to the NPacket.csv dataset, and the results are clearly presented in tables (1 to 8) in Appendix A at the end of the report. Subsequently, some statistical operations were performed on the results to present them clearly and interpretable. Initially, the weighted average for performance criteria, Precision, Recall, and F1-score, was plotted against the number of packages taken from the beginning of each flow. The results are illustrated in Figures (5-9), (5-10), ..., (5-14). The weighted average for Recall, for example, is given by Equation (11).

$$\text{Wieghted average(Recall)} = \frac{\sum_{i=1}^{6} \text{Recal}(Class_i) * Class_i\ Rate}{\sum_{i=1}^{6} Class_i\ Rate} \qquad (7)$$

Where $Class_i\ Rate$ represents the percentage of samples belonging to $Class_i$ relative to the total number of samples. In our case, $\sum_{i=1}^{6} Class_i\ Rate = 1$.

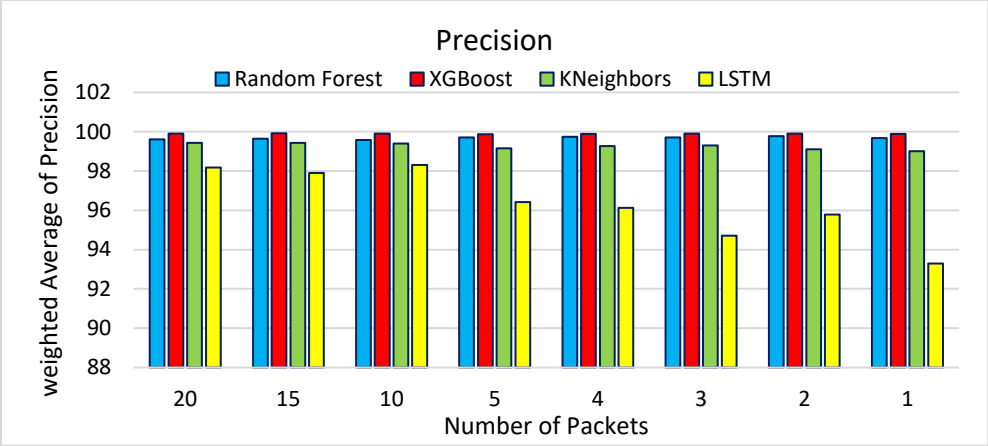

Figure (5-9): Weighted Average of the Adjustment Criterion as a Function of N in Scenario 1.

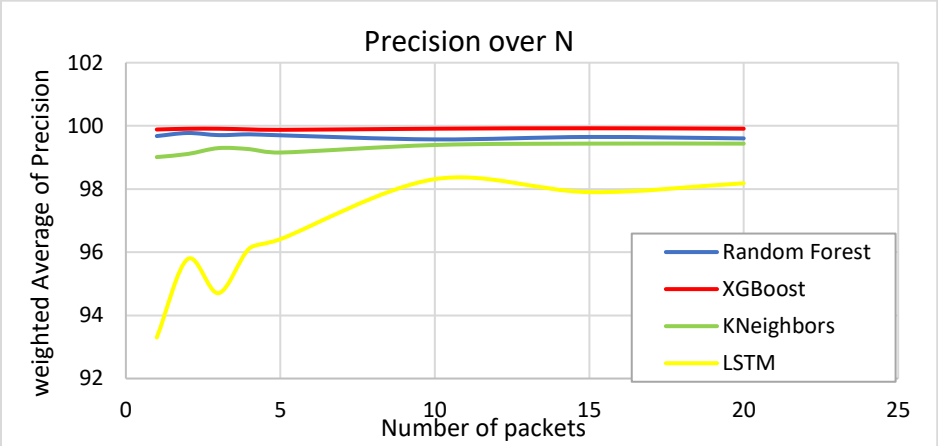

Figure (5-10): Weighted Average of the Adjustment Criterion for the Applied Algorithms Along N in Scenario 1.

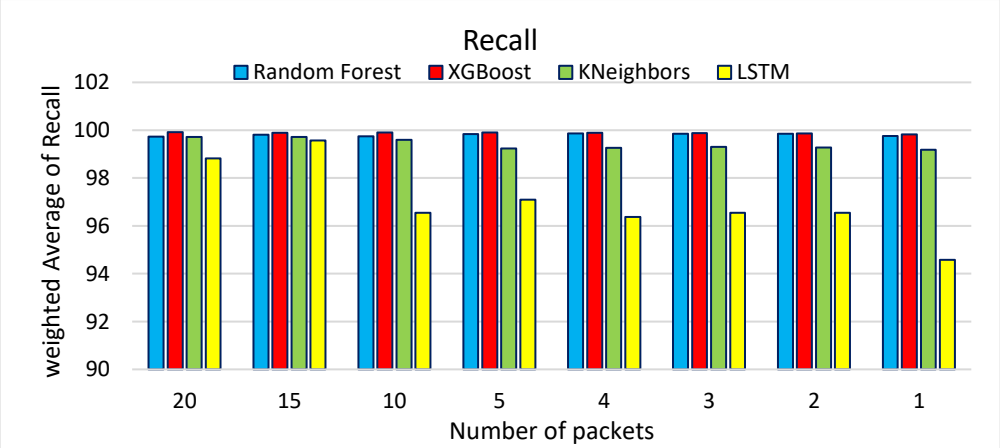

Figure (5-11): Weighted Average of the Recall Criterion as a Function of N in Scenario 1.

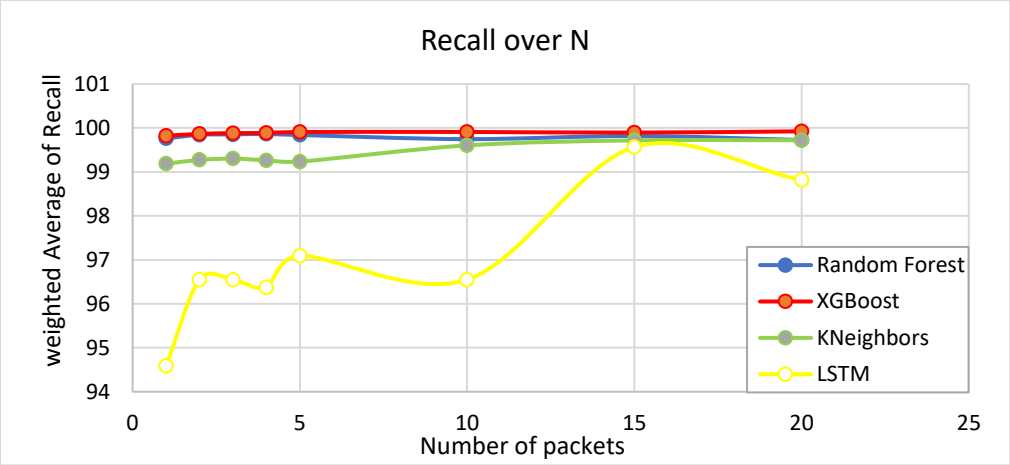

Figure (5-12): Weighted Average of the Recall Criterion for the Applied Algorithms Along N in Scenario1.

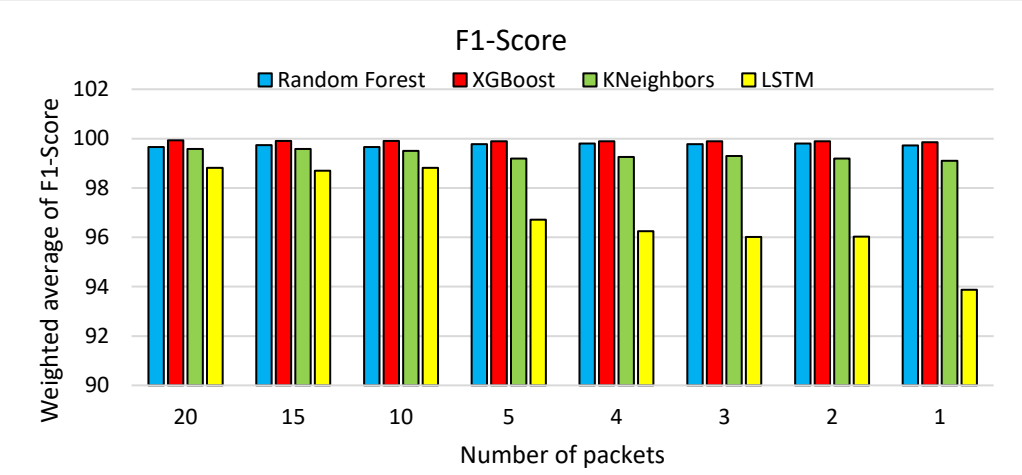

Figure (5-13): Weighted Average of the Score F1 Criterion as a Function of N in Scenario 1.

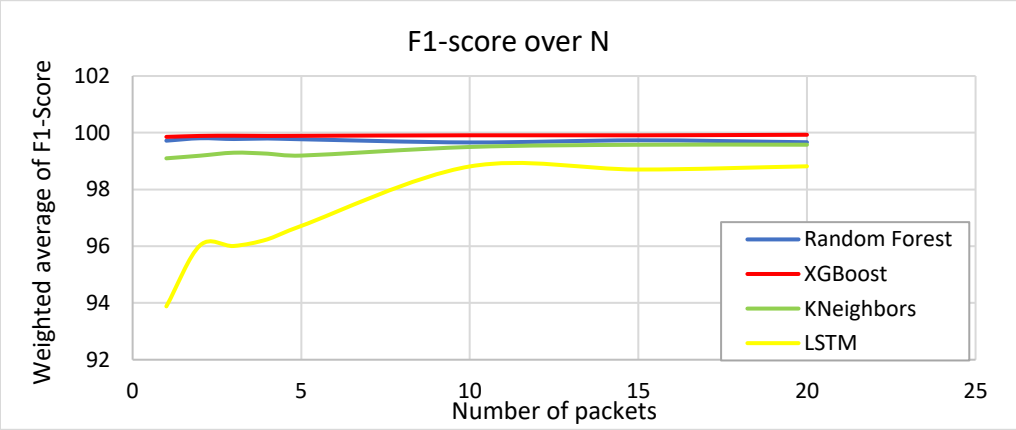

Figure (5-14): Weighted Average of the Score F1 Criterion for the Applied Algorithms Along N in Scenario 1.

We observe from the above results that machine learning algorithms have outperformed the deep learning algorithm used for all values of N and for all performance criteria. It is clearly noticeable that the performance of the LSTM algorithm drops as the number of batches taken from the beginning of each stream decreases. Therefore, the LSTM deep learning algorithm requires a large amount of information to give good results and to outperform machine learning algorithms. On the other hand, machine learning algorithms have maintained their performance despite the decrease in the number of batches taken from each stream. Additionally, the XGBoost algorithm excels over all algorithms for all numbers of batches taken from each stream and for all performance criteria, maintaining stable performance even at N=1.

To view the results from another perspective, the F1-Score standard was plotted with respect to the uncovered classes. Figure (5-15) shows the distribution of different classes in the training and testing set.

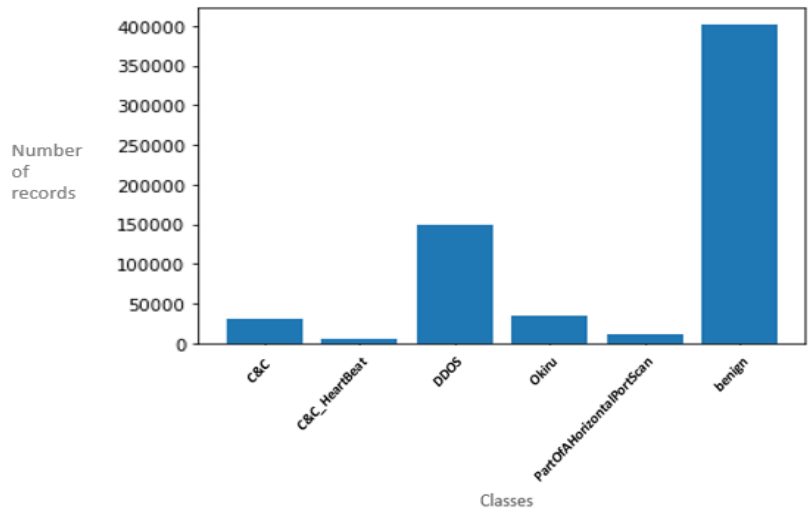

Figure (5-15): Distribution of classes in the NPacket.csv dataset.

We notice from Figure (5-15) that the distribution of classes is unbalanced. Despite the existence of various algorithms (such as the SMOTE algorithm) that help balance the classes, we preferred not to apply any algorithm of this type to keep the samples taken as real samples. These algorithms may give approximate values, but they have no meaning from a network perspective (for example, they may give a non-existent IP address or a port number with no meaning). Although these algorithms help increase the detection rate for classes with low frequency, they increase the noise ratio for classes with high frequency and may reduce their detection accuracy. For these reasons,

and because we want our model to be closer to real-world application, we did not perform any class balancing.

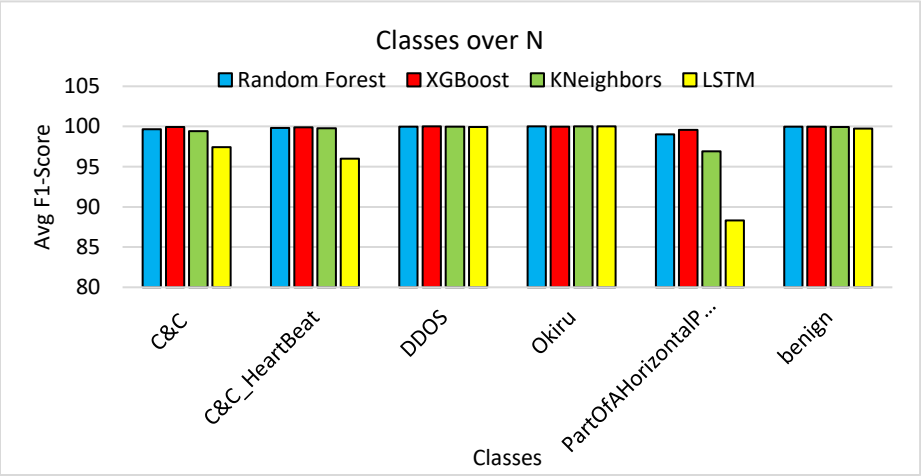

Figure (5-16): Performance of the four algorithms for each class in Scenario 1.

We observe from Figure (5-16) that the performance of the XGBoost algorithm was not affected by the frequency of classes compared to other algorithms. We also notice a significant impact on the performance of the LSTM algorithm with the frequency of the uncovered class within the testing and training set. After analyzing the previous results, the XGBoost algorithm was adopted as the best detection algorithm, and N=1 was adopted because the performance of the XGBoost algorithm was not affected by the number of batches taken from each stream. However, after calculating the importance of features for the detection model (Feature Importance) by taking one batch from each stream, Figure (5-17) was obtained.

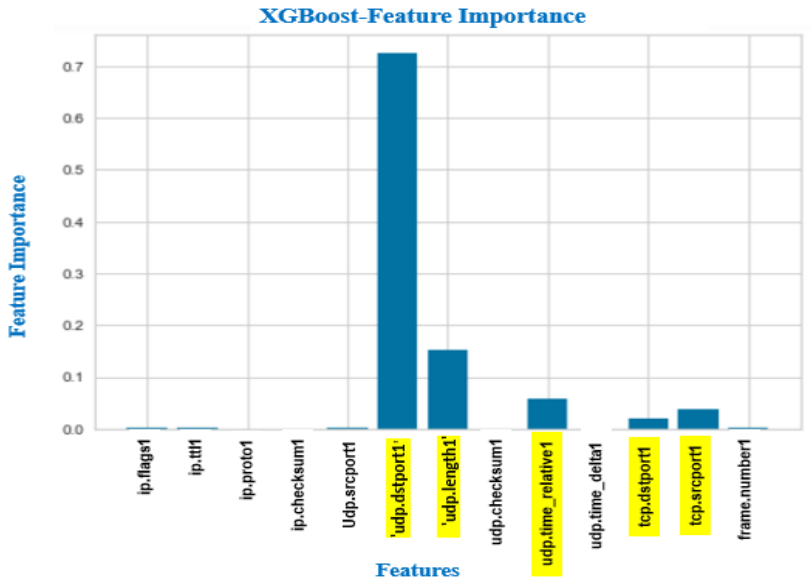

Figure (5-17): Importance of features for the XGBoost classifier for N=1 in Scenario 1.

We notice from Figure (5-17) that features related to port numbers have a very significant impact on the model. It is known that attacks like DDoS can change port numbers during the attack to avoid detection. This led us to reapply the algorithms again to the dataset after removing port numbers from the model.

5.5.2 Scenario 2

Applying algorithms without port numbers

When applying the four algorithms again in the same way, the results shown in tables (9 to 16) in Appendix A were obtained. The standards were also plotted in terms of the number of batches taken from each stream, and the figures were obtained (from (5-18), (5-19), ..., (5-23)).

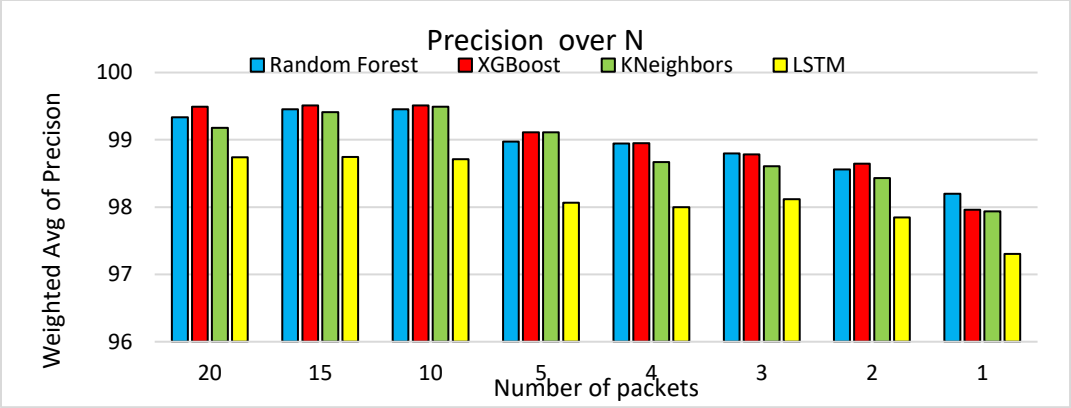

Figure (5-18): Weighted average of the tuning standard with respect to N in Scenario 2.

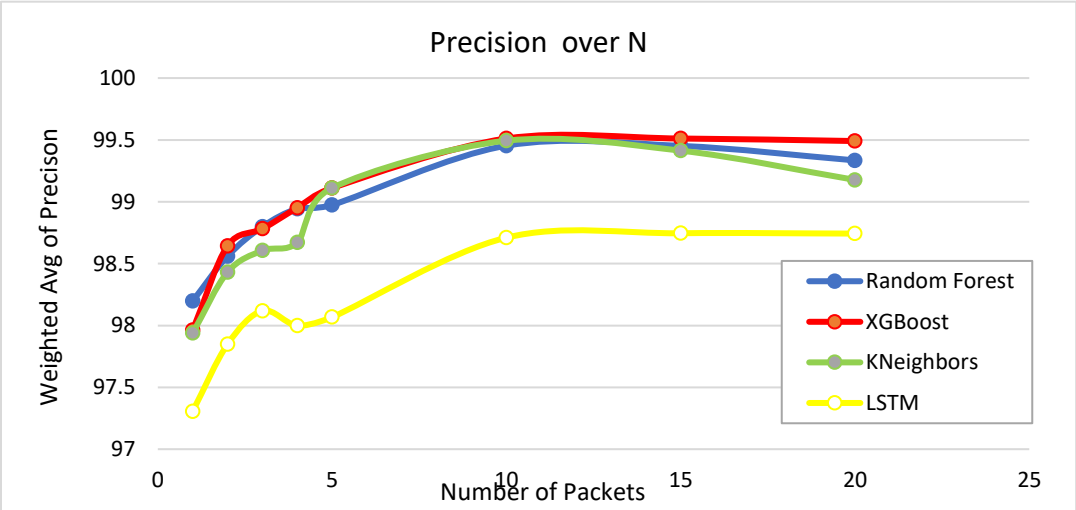

Figure (5-19): Weighted average of the tuning standard for the applied algorithms over N in Scenario 2.

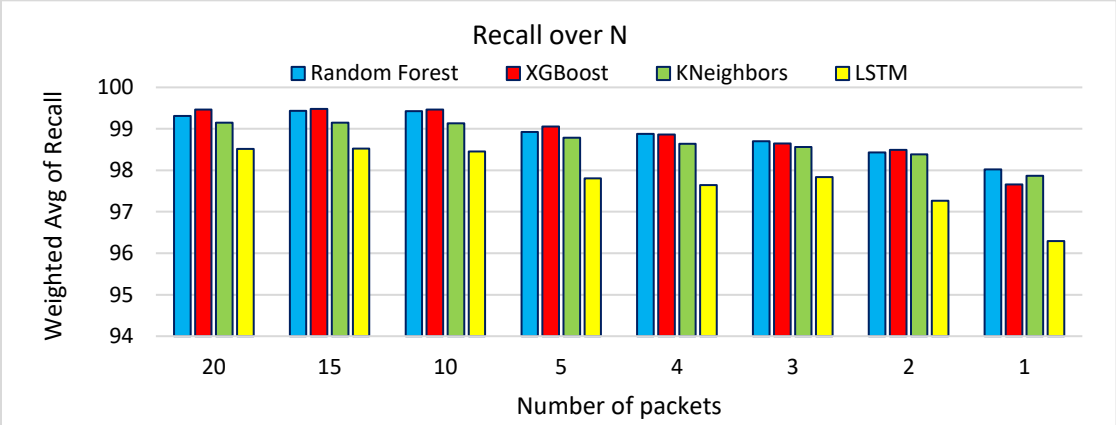

Figure (5-20): Weighted average of the recall standard with respect to N in Scenario 2.

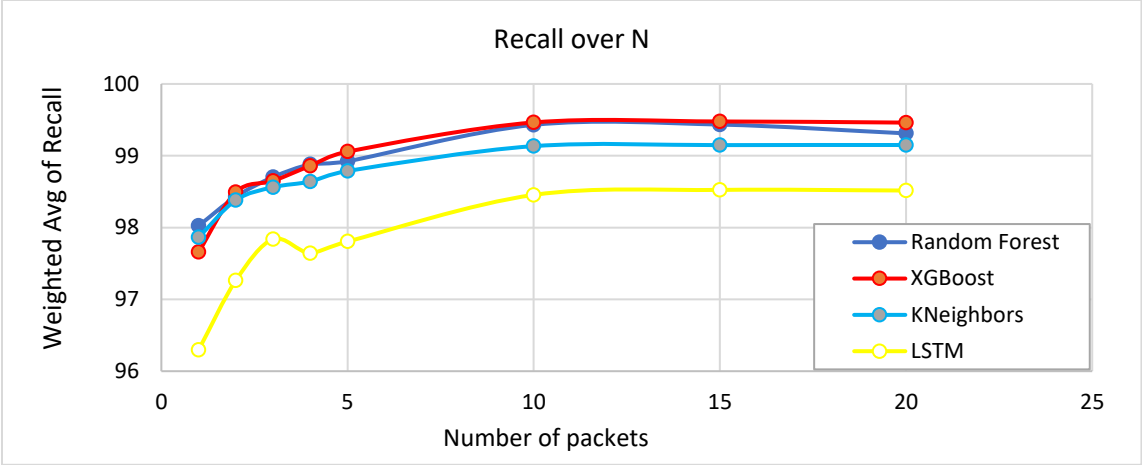

Figure (5-21): Weighted average of the recall standard for the applied algorithms over N in Scenario 2.

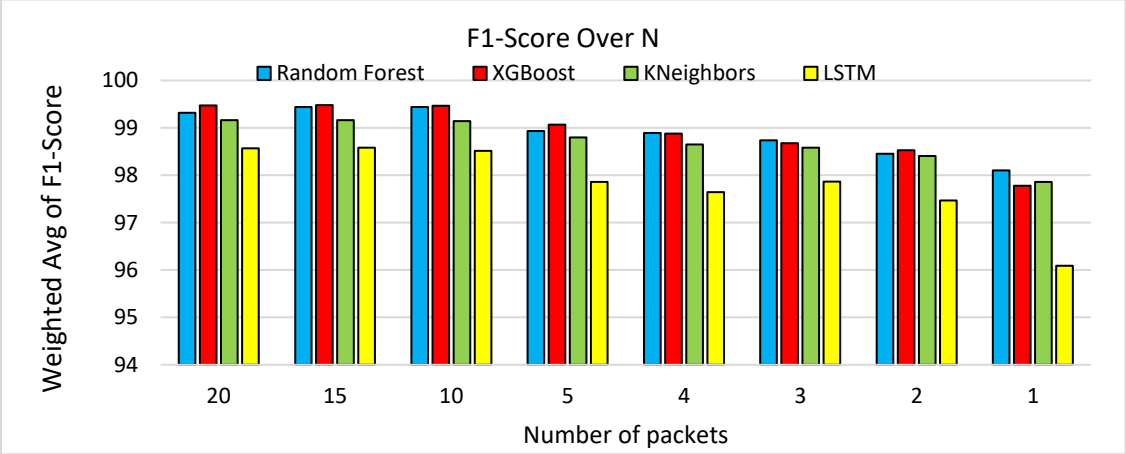

Figure (5-22): Weighted average of the F1-Score standard with respect to N in Scenario 2.

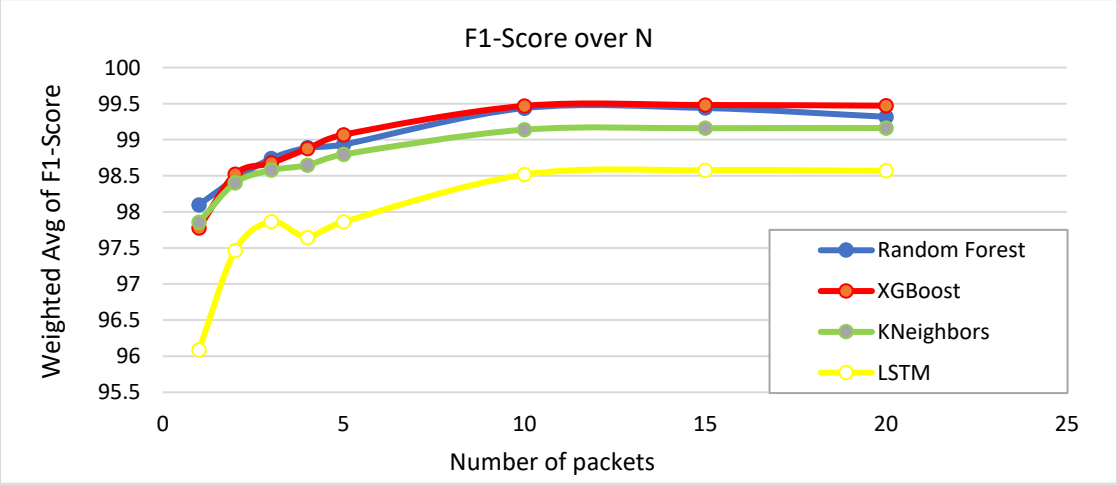

Figure (5-23): Weighted average of the F1-Score standard for the applied algorithms over N in Scenario 2.

From the previous figures, we can clearly observe that deleting the port numbers resulted in a decrease in accuracy for all standards with a decrease in N for all algorithms. In Scenario 1, machine learning algorithms maintained their performance for all values of N. As for the best detection algorithm, the XGBoost algorithm maintained the best performance compared to the other algorithms. Regarding the choice of the number of batches taken from each stream, this scenario differed from the previous scenario, where we observe a clear decrease in performance at N=1 for all algorithms, prompting us to reconsider this issue. The F1-Score standard was plotted with respect to the classes in this scenario, and Figure (5-24) was obtained.

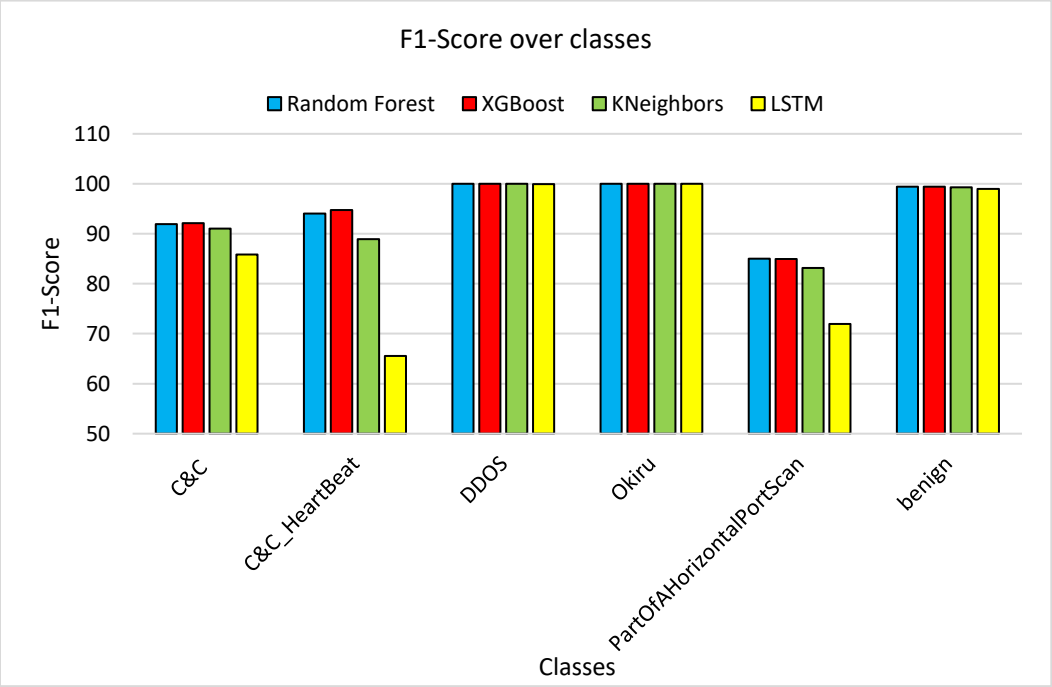

Figure (5-24): Performance of the four algorithms for each class in Scenario 2.

Comparing Figure (5-24) with Figure (5-16), we notice that all algorithms were affected by the frequency of classes within the testing and training set after removing port numbers. Therefore, the detection accuracy for all classes with low frequency decreased. However, for DDoS, Okiru, and benign classes, the detection accuracy remained high and close to 100% for all performance criteria. For choosing the value of N, the F1-Score standard was plotted for all classes in the XGBoost algorithm with respect to N to choose the best value for it, as shown in Figure (5-25).

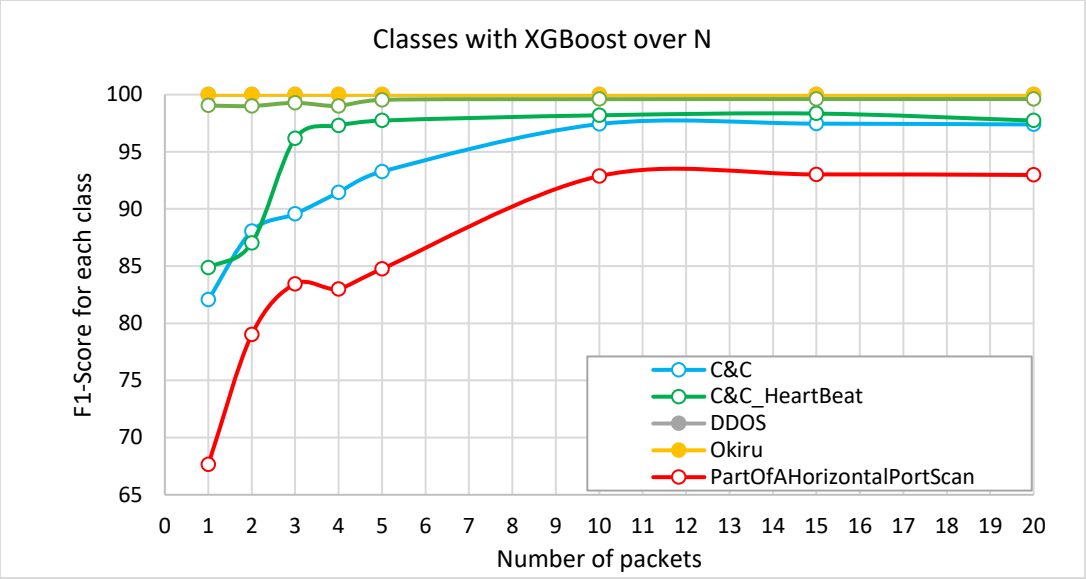

Figure (5-25): Performance of classes with the XGBoost classifier according to the F1-Score standard with respect to N.

We notice from Figure (5-25) that the best value for the F1-Score standard for all classes corresponds to N=10. However, we aim to have a model that supports real-time detection, so we are interested in choosing a small value for N that gives an acceptable detection accuracy. From Figure (5-25), we observe the following: A decrease in detection accuracy from 96.16% to 84.85% in the Malicious C&C_HeartBeat class when transitioning from N=3 to N=1. A decrease in detection accuracy from 89.57% to 82.05% in the Malicious C&C class when transitioning from N=3 to N=1. A decrease in detection accuracy from 83.44% to 67.65% in the Malicious PartOfAHorizontalPortScan class when transitioning from N=3 to N=1. For the remaining classes (Okiru, DDoS, and benign), the detection accuracy remained almost constant within the range N∈[1,3]. As a result of the above, N=3 was adopted to achieve optimal real-time detection.

5.5.3 The Best Scenario for Intrusion Detection

Based on the results in Scenario 2, adopting the approach of taking 3 bundles from each flow and applying the XGBoost algorithm, this section will present detailed results for the best scenario. Table (5-7) illustrates the results obtained when applying the algorithm for all performance metrics, while Figure (5-26) shows the confusion matrix.

Table (5-6): Results of the Best Scenario.

N=3	Classes	precision	recall	fscore	AUC
XGBoost	C&C	83.04%	97.21%	89.57%	1
	C&C_HeartBeat	97.71%	94.66%	96.16%	1
	DDoS	100.00%	100.00%	100.00%	1
	Okiru	100.00%	100.00%	100.00%	1
	PartOfAHorizontalPortScan	89.13%	80.00%	83.04%	1
	benign	99.76%	98.81%	99.28%	1

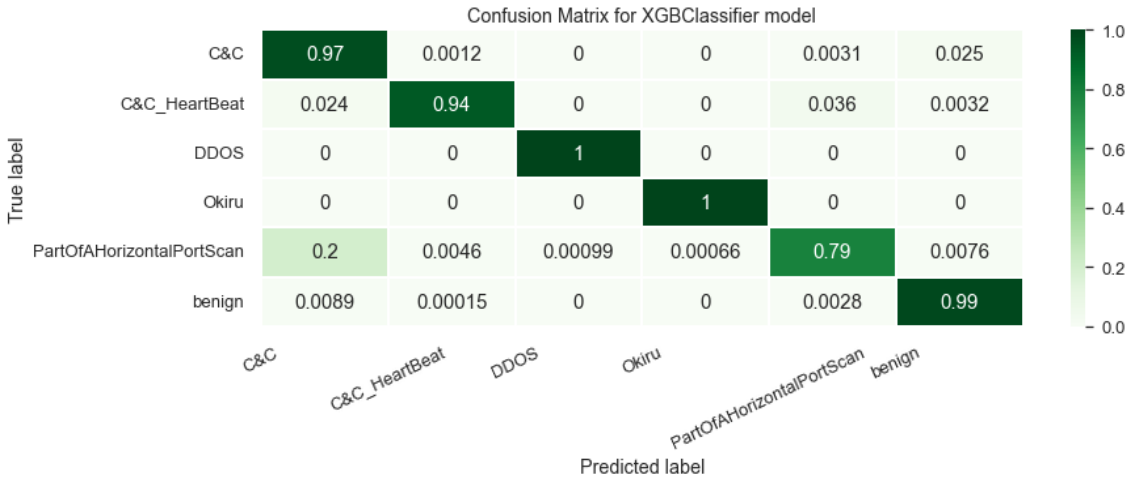

Figure (5-26): Confusion Matrix for the Best Scenario.

From Table (5-7) and Figure (5-26), it is observed that DDoS attacks were detected with a high F1-Score precision of 100%. The robot Okiru was also detected with an accuracy close to 100%, achieving the primary goal of the project. The detection accuracy for the XGBoost algorithm (Accuracy) was calculated and resulted in 98.66%. Figure (5-27) illustrates the importance of each feature within the test and training sets, containing three bundles from each flow.

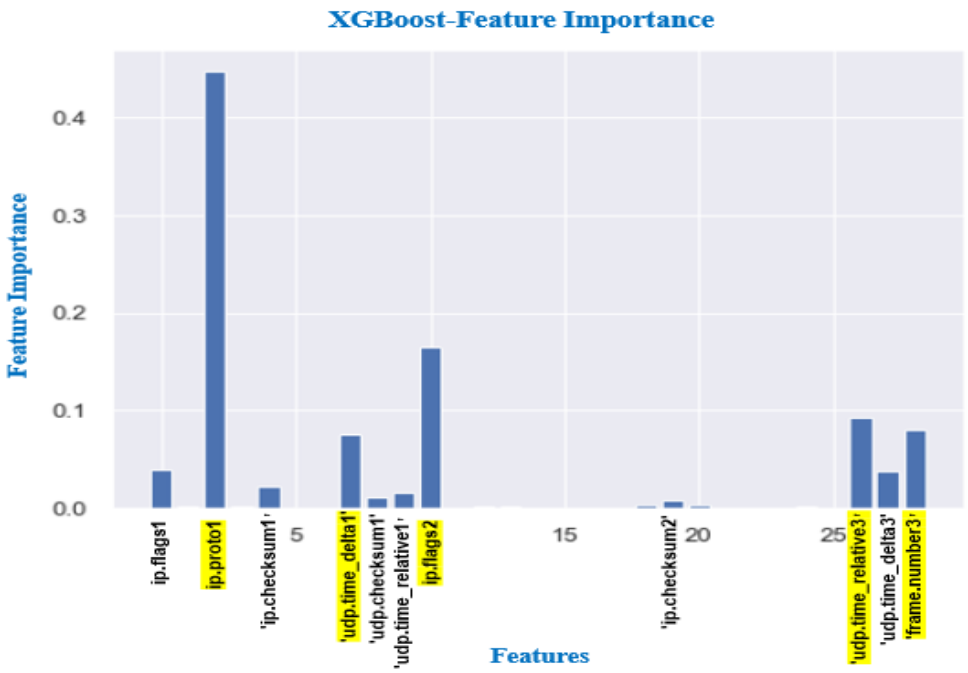

Figure (5-27): Importance of Features for N=3 Using the XGBoost Classifier.

Table (5-7): Ranking of Features According to Their Importance with the Significance of Each Feature.

Feature	Meaning	Score
ip.proto1	Protocol number in ip header of first packet	0.44761
ip.flags2	Enable fragmentation in second packet	0.16511
udp.time_relative3	the time since it received the first frame in the UDP session in third packet	0.09210
frame.number3	the third packet number based on the order in which the packets appear in the capture file	0.07932
udp.time_delta1	the difference in time stamps between the packet in question and the packet before it in the packet list.	0.07510
ip.flags1	Enable fragmentation in first packet	0.03891
udp.time_delta3	the difference in time stamps between the packet in question and the packet before it in the packet list.	0.03705
ip.checksum1	check on whether the data received is error free or not.	0.02120
udp.time_relative1	the time since it received the first frame in the UDP session in third packet	0.01554
udp.checksum1	check on whether the data received is error free or not.	0.01095

Table (5-7) helps build a less complex intrusion detection system by reducing the features from 30 to 11.

6.5 Summary of Results

Based on Scenarios 1 and 2 and their results, the following conclusions can be drawn:

1. Machine learning algorithms outperform deep learning algorithms in performance for small-sized samples.

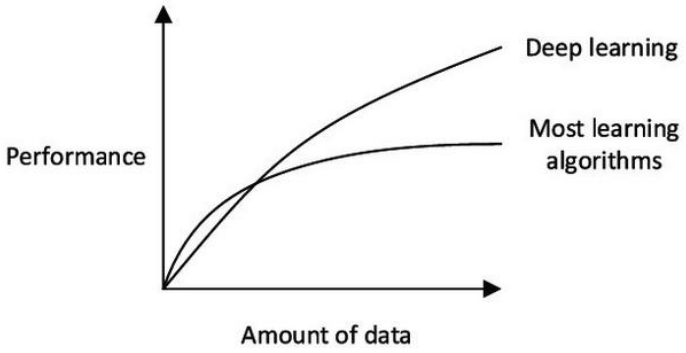

Figure (5-28): Performance of deep learning algorithms and machine learning algorithms based on the size of the trained data.

2. The best algorithm for detection among the machine learning algorithms used is XGBoost, achieving a weighted average F1-Score of 99.9% in Scenario 1 and 98.68% in Scenario 2.
3. Some features, like port numbers, cannot be relied upon for classification, as they may change during an attack, leading to detection errors.
4. The proposed model achieved optimal detection for DDoS and Okiru classes, reaching 100% in both scenarios, thus achieving the project's main goal.
5. XGBoost is a machine learning algorithm that requires less training time and is less complex than deep learning algorithms, making the detection model less complicated when applied in real-world scenarios.
6. In the proposed model, three bundles were taken from the beginning of each flow to achieve real-time detection due to a significant performance drop for all classes with a lower number of bundles in Scenario 2.
7. The model became less complex after reducing the features from 30 to 11 while maintaining the same detection accuracy.

7.5 Comparison with Previous Work

A comparison was made between the proposed model and models 2, 4, and 5 (discussed in paragraph 3.4) since all of them are real-time intrusion detection models. Figures (5-29) and (5-30) show the importance of features in models 1 and 5, respectively. It is noted that both models relied on port numbers as a feature for detection, resulting in higher accuracy for F1-Score in model 1 (95%) and model 5 (97.39%). Although removing port numbers from the test and training sets reduced the accuracy in our proposed model, we achieved a higher F1-Score of 98.68%.

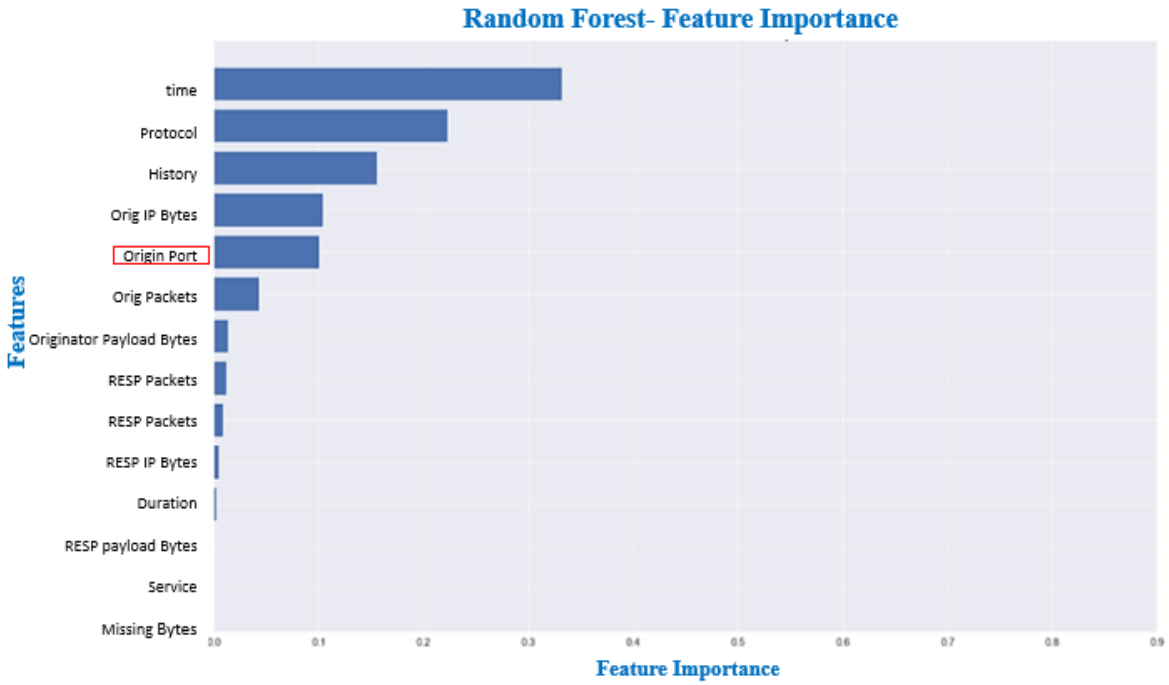

Figure (5-29): Importance of Features in Model 5.

Header	Features
Ethernet	eth_source, eth_destination, eth_type
IP	ip_version, ip_hdr_len, ip_tos, ip_length, ip_identification, ip_flags, ip_offset, ip_ttl, ip_protocol, ip_checksum, ip_source, ip_destination
TCP	tcp_source_port, tcp_destination_port, tcp_sequence, tcp_acknowledge, tcp_offset, tcp_flags_res, tcp_flags, tcp_window_size, tcp_checksum, tcp_urgent_point
UDP	udp_source_port, udp_destination_port, udp_ulen, udp_checksum

Figure (5-30): Features Used in Model 2.

The table illustrates the F1-Score metric for the previous models in comparison with the proposed model.

Table (5-8): F1-Score in the Proposed Model Compared to Previous Models Using Real-Time Detection.

Model	F1-Score
2	%95
4	%97.41
5	%97.39
Proposed Model	%98.68

From Table (5-8), it is evident that the proposed model outperforms all previous models relying on real-time detection.

8.5 Integration of the Proposed Model with Intrusion Detection System Placement

Intrusion detection systems should be strategically placed to analyze network traffic effectively. Most companies and organizations are typically equipped with Network Intrusion Detection Systems (NIDS) alongside firewalls. It is crucial to consider the placement of the intrusion detection system concerning the firewall. The system can be positioned between the firewall and the internal network, as shown in Figure (5-31), or behind the firewall, as shown in Figure (5-32). The former allows NIDS to see traffic coming from the internet, including attacks against the firewall, while the latter provides visibility and analysis of traffic passing through the firewall into the network [32].

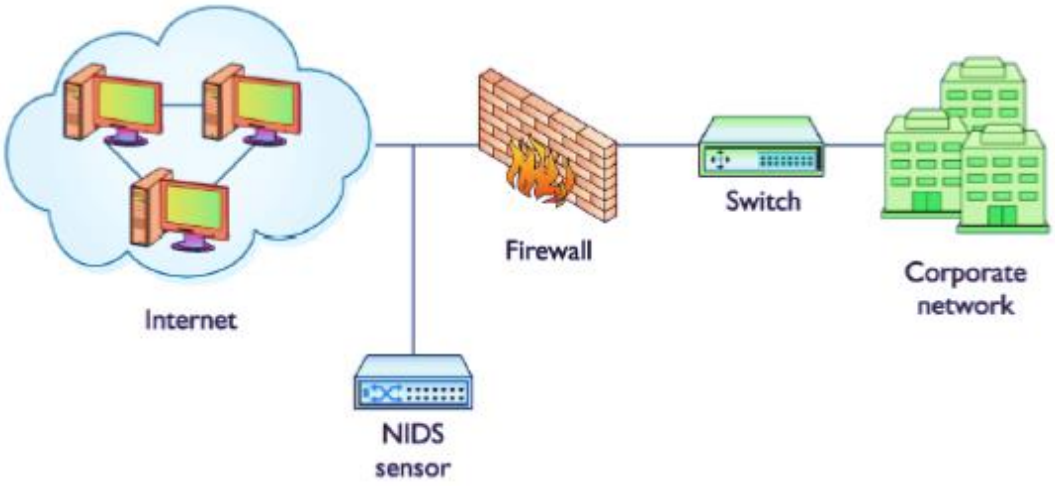

Figure (5-31): Placement of the Intrusion Detection System in Front [37].

Figure (5-31) illustrates an intrusion detection system deployed before the firewall. In this case, the Network Intrusion Detection System (NIDS) observes the incoming traffic from the internet, including attacks against the firewall. This encompasses traffic stopped by the firewall, preventing it from entering the network. In this distribution, NIDS will generate a large number of alerts, including those related to traffic that the firewall may block.

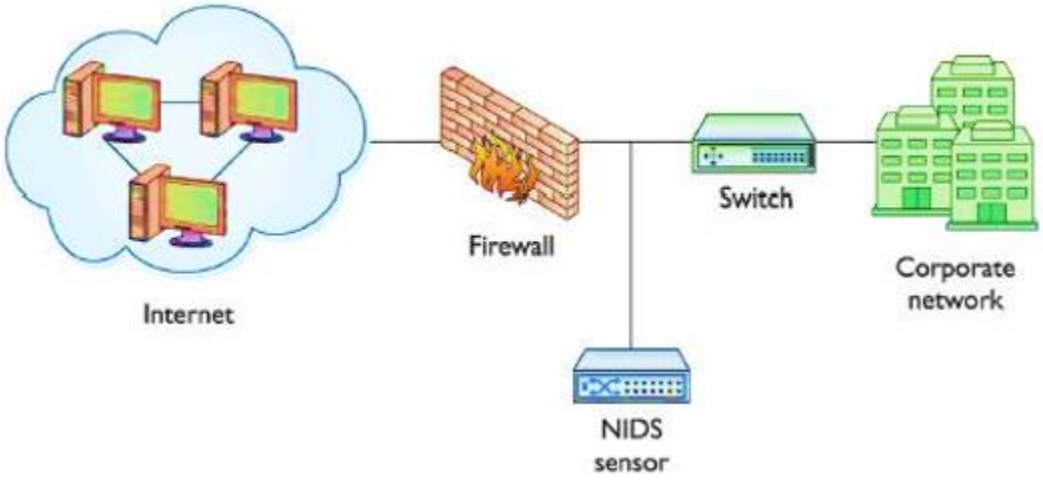

Figure (5-32): Intrusion detection system positioned behind the firewall [37].

Figure (5-32) demonstrates an intrusion detection system located behind the firewall. In this scenario, NIDS sees and analyzes the traffic passing through the firewall to the network. This distribution does not allow NIDS to see attacks prevented by the firewall, resulting in fewer alerts being produced.

9.5 Conclusion and Recommendations

In this chapter, we examined various tools used to construct a real-time multi-class classification model, along with the performance metrics used to evaluate the Random Forest, KNN, XGBoost, and LSTM algorithms. After analyzing the results, the XGBoost algorithm, applied to three packets from the beginning of each traffic flow, achieved the best detection scenario with a weighted average F1-Score of 98.68%.

Conclusion and Future Perspectives

In this project, we presented a real-time detection model that performs multiclass classification for Botnet attacks on Internet of Things (IoT) networks using the IoT-23 dataset. We began by discussing the types and impact of attacks on IoT networks, emphasizing the inadequacy of traditional systems in combating the new types of Botnet attacks on IoT devices. This led to the utilization of machine learning algorithms and artificial intelligence techniques to build more effective systems, and we transitioned to defining machine learning algorithms, their types, and the challenges faced, ultimately leading to the use of deep learning algorithms.

In the proposed model, we discussed the rationale for choosing the IoT-23 dataset, referencing research papers related to our project that were published after its release. We then outlined the steps for implementing the proposed model, starting with data preprocessing to extract features at the packet level. We moved on to the classifier model, which included algorithms such as Random Forest, KNN, XGBoost, and LSTM.

During the development of the proposed model, we conducted various experiments, varying the values of N and the type of algorithm applied to the derived dataset (Npacket.csv). Despite the relatively small size of the trained samples, all machine learning algorithms outperformed the deep learning LSTM algorithm. After result analysis, we identified the optimal real-time detection scenario using the XGBoost classifier, taking three packets from the beginning of each traffic flow in the IoT-23 dataset. This scenario achieved the best-weighted average F1-Score of 98.68%, surpassing all previous models relying on real-time detection. It also achieved a perfect F1-Score of 100% for the DDoS and Okiru classes.

In conclusion, we can affirm that we achieved the primary goal of the project. The proposed model successfully achieved optimal detection of various Botnet attacks present in the derived dataset. The machine learning model demonstrated compatibility with intrusion detection systems for monitoring network traffic and detecting anomalies, alerting administrators.

This model is easily upgradable, with an open field for using other deep learning algorithms after increasing the number of samples taken from the IoT-23 dataset. Additionally, adjustments to algorithm parameters can be made to yield better results, or a new machine learning model can be proposed to achieve a higher detection rate.

References

1. Thamaraiselvi, D.; Mary, S. Attack and Anomaly Detection in IoT Networks using Machine Learning. Int. J. Comput. Sci. Mob.Comput. 2020, 9, 95–103.

2. Abdullah Ameen DM SY. "Enhanced Mobile Broadband (EMBB): A review," Journal of Information Technology and Informatics. 2021;1:13-19.

3. Othman A, Ameen SY, Al-Rizzo H. "A new channel quality indicator mapping scheme for high mobility applications in LTE systems," Journal of Modeling and Simulation of Antennas and Propagation. 2015;1:38-43,

4. E. Bertino and N. Islam, “Botnets and internet of things security,” Computer, no. 2, pp. 76–79, 2017.

5. C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, “DDoS in the iot: Mirai and other botnets,” Computer, vol. 50, no. 7, pp. 80–84, 2017.

6. R. DoShi, N. Apthorpe, and N. Feamster, “Machine learning DDoS detection for consumer internet of things devices,” in 2018 IEEE Security and Privacy Workshops (SPW), May 2018, pp. 29–35.

7. Alzahrani, H.; Abulkhair, M.; Alkayal, E. A multi-class neural network model for rapid detection of IoT botnet attacks. Int. J. Adv. Comput. Sci. Appl. 2020, 11, 688–696.

8. Janaby AOAl, A. Al-Omary, S. Y. Ameen, H. Al-Rizzo, "Tracking and controlling highspeed vehicles via CQI in LTE-A systems," International Journal of Computing and Digital Systems. 2020;9:1109-1119.

9. Abdulla AI, A. S. Abdulraheem, A. A. Salih, M. A. Sadeeq, A. J. Ahmed, B. M. Ferzor, et al., "Internet of Things and Smart Home Security," Technol. Rep. Kansai Univ. 2020;62:2465-2476.

10. Hamed ZA, I. M. Ahmed, S. Y. Ameen, "Protecting Windows OS Against Local Threats Without Using Antivirus," relation. 2020;29:64-70.

11. Syed NF, Baig Z, A. Ibrahim, C. Valli, "Denial of service attack detection through machine learning for the IoT," Journal of Information and Telecommunication. 2020;4:482-503.

12. Manos Antonakakis et al. “Understanding the Mirai Botnet”. In: 26th USENIX Security Symposium, p. 19.

13. Stephen Hilt. “Worm War: The Botnet Battle for IoT Territory”. In : documents.trendmicro.com (), p. 30.

14. Avast. New Torii Botnet uncovered, more sophisticated than Mirai — Avast. url: https://blog.avast.com/new- torii- botnet- threat- research.

15. Syed NF, Baig Z, A. Ibrahim, C. Valli, "Denial of service attack detection through machine learning for the IoT," Journal of Information and Telecommunication. 2020;4:482-503.

16. Ageed ZS, Zeebaree SR, Sadeeq MM, Kak SF, Z. Rashid N, Salih AA, et al., "A survey of data mining implementation in smart city applications," Qubahan Academic Journal. 2021;1:91-99.

17. H. O. Alanazi, R. Noor, B. B. Zaidan, and A. A. Zaidan, “Intrusion Detection System : Overview,” J.Comput., vol. 2, no. 2, pp. 130–133, 2010.

18. http://nmi-lab.org/PSYCH239-NNML19 , slides_1.html#/73

19. Benuwa BB, Zhan YZ, Ghansah B, Wornyo DK, Banaseka Kataka F. A review of deep machine learning. International Journal of Engineering Research in Africa. 2016;24:124-36.

20. P. Cunningham and S. J. Delany, “k-nearest neighbour classifiers: 2nd edition (with python examples),” ACM Computing Surveys, vol. 54, no. 6, pp. 1–25, 2020.

21. Ali J, Khan R, Ahmad N, Maqsood I. Random forests and decision trees. International Journal of Computer Science Issues (IJCSI). 2012;9(5):272.

22. T. Chen, H. Li, Q. Yang, and Y. Yu. General functional matrix factorization using gradient boosting. In Proceeding of 30th International Conference on Machine Learning (ICML’13), volume 1, pages 436–444, 2013.

23. hen, T., Guestrin, C.: XGBoost: A scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 785–794. KDD ’16, ACM, New York, NY, USA (2016).

24. K. Cho, B. Van Merri ¨enboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation, arXiv Prepr. (2014) arXiv:1406.1078. doi:1406.1078.

25. Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput. 1997;9(8):1735–80. doi:10.1162/neco.1997.9.8.1735.

26. Canadian Institute for Cybersecurity. DDoS Evaluation Dataset (CIC-DDoS2019). 2019. Available online: https://www.unb.ca/cic/datasets/DDoS-2019.html.

27. C. D. McDermott, F. Majdani and A. V. Petrovski, "Botnet Detection in the Internet of Things using Deep Learning Approaches," in 2018 International Joint Conference on Neural Networks (IJCNN), IEEE, 2018, pp. 1-8.

28. Koroniotis, Nickolaos, Nour Moustafa, Elena Sitnikova, and Benjamin Turnbull. "Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset." *Future Generation Computer Systems* 100 (2019): 779-796.

29. Agustin Parmisano, Sebastian Garcia, and Maria Jose-Erquiaga. IoT-23 Dataset: A labeled dataset of Malware and Benign IoT Traffic. url: https://www.stratosphereips. org/datasets-iot23.

30. Saad Hikmat Haji1 and Siddeeq Y. Ameen1. Attack and Anomaly Detection in IoT Networks using Machine Learning Techniques: A Review. Asian Journal of Research in Computer Science. 2021 .

31. Dutta, Vibekananda, et al. "A deep learning ensemble for network anomaly and cyber-attack detection." Sensors 20.16 (2020): 4583.

32. Kozik, Rafał, Marek Pawlicki, and Michał Choraś. "A new method of hybrid time window embedding with transformer-based traffic data classification in IoT-networked environment." Pattern Analysis and Applications (2021): 1-9.

33. Ullah, Imtiaz, and Qusay H. Mahmoud. "Network Traffic Flow Based Machine Learning Technique for IoT Device Identification." 2021 IEEE International Systems Conference (SysCon). IEEE, 2021.

34. N. Abdalgawad, A. Sajun, Y. Kaddoura, I. A. Zualker-nan e F. Aloul. “Generative Deep

Learning to Detect Cyberattacks for the IoT-23 Dataset”. Em: IEEE Access10 (2022), pp. 6430–6441. ISSN: 21693536. DOI: 10.1109/ACCESS.2021.3140015.

35. Austin, Michael. "IoT Malicious Traffic Classification Using Machine Learning." (2021).

36. Goutte, C. & Gaussier, E. A probabilistic interpretation of precision, recall and F-score, with implication for evaluation. In European Conference on Information Retrieval (Losada, D. E. & Fernández-Luna, J. M.) 345–359.

37. Kraus, S. Feuerriegel, A. Oztekin, Deep learning in business analytics and operations research: Models, applications and managerial implications, Eur. J. Oper. Res. 281 (2020), 629.

38. WHITE, G.; CONKLIN, W. A.; WILLIAMS, D.; DAVIS, R.; COTHREN, C. Principles of Computer Security, CompTIA Security+ and Beyond. 3rd edition, McGraw-Hill, 2012,

684.