

# **Machine Learning-Based DDoS (Distributed Denial of Service) Detection System**

CS39440 Major Project Report

Author: Thomas Roethenbaugh (tpr3@aber.ac.uk)

Supervisor: Dr Muhammad Aslam (mua19@aber.ac.uk)

1st May 2025

Version: 1.2 (Release)

This report was submitted as partial fulfilment of a BSc degree in Computer Science  
(with integrated year in industry) (G401)

Department of Computer Science  
Aberystwyth University  
Aberystwyth  
Ceredigion  
SY23 3DB  
Wales, U.K.

## **Declaration of originality**

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Registry (AR) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work I understand and agree to abide by the University's regulations governing these issues.

Name Thomas Roethenbaugh

Date 14/03/2025

## **Consent to share this work**

By including my name below, I hereby agree to this project's report and technical work being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name Thomas Roethenbaugh

Date 14/03/2025

## **Acknowledgements**

I acknowledge the great help I received from my supervisor Dr Muhammad Aslam who's guidance and insight were invaluable to completing this project.

I would also like to acknowledge the support I have received from my family and friends throughout both these 14 weeks working on this project, and 4 years of university.

## Abstract

This project aimed to achieve the detection of malicious network traffic in the form of distributed denial of service (DDoS) attacks, through the use of established machine learning (ML) models such as decision tree (DT), random forest (RF), and k-nearest neighbours (KNN), as well as ensemble voting methods. The increasing number of attacks and their severity is well documented. With 25% of declared hacking incidents in the UK now being declared as DDoS attacks by the Financial Conduct Authority (FCA) [1]. With many of these attacks targeting cloud computing, its affects have become more widespread as more and more companies utilise cloud computing for their businesses. According to Cloudflare (a cloud computing company), they have recently broken records for the scale of some of their attacks [2]. Detailed in this report, it breaks down the methodology of each attack. Presenting HTTP, DNS, UDP and SYN attacks (among others), showing the modern, multi-vector approach of today's DDoS attacks.

In order to combat this growing problem, new techniques have been developed in detection and mitigation. For example, the paper FMDADM: A Multi-Layer DDoS Attack Detection and Mitigation Framework Using Machine Learning for Stateful SDN-Based IoT Networks [3] uses a novel ensemble ML as well as a software defined networking (SDN) controller to more effectively counter DDoS attacks. Outperforming traditional firewalls and rate-limiting. In this project, similar techniques were implemented and tested on a well-established dataset. This being the DDoS evaluation dataset (CIC-DDoS2019) [4]. Several statistical methods were used to estimate its accuracy, F-1 score, precision and recall, with a final score of 97% accuracy rate. A simulated environment was also created to both test generating real DDoS traffic, as well as assist in the creation of a custom dataset. This was used to help train the previously mentioned ML models and was even used for a real-time DDoS detection system. Said real-time detection system was successfully created and tested in this virtual environment.

# Contents

<b>1</b>	<b>Background and Objectives</b>	<b>1</b>
1.1	Background . . . . .	1
1.1.1	Problem overview . . . . .	1
1.1.2	Functional requirements & research question . . . . .	1
1.1.3	DoS and DDoS . . . . .	2
1.1.4	Machine learning . . . . .	3
1.2	Analysis . . . . .	4
1.2.1	Overview of the solution proposed by this project . . . . .	4
1.2.2	Choosing a dataset . . . . .	4
1.2.3	Virtual environment . . . . .	4
1.3	Research Method and Software Process . . . . .	5
1.3.1	Software methodology . . . . .	5
1.3.2	Research methodology . . . . .	5
1.4	Literature review . . . . .	5
1.4.1	Enhancing DDoS Attack Detection and Mitigation in SDN Using an Ensemble Online Machine Learning Model . . . . .	6
1.4.2	Detecting DDoS Threats Using Supervised Machine Learning for Traffic Classification in Software Defined Networking . . . . .	6
1.4.3	FMDADM: A Multi-Layer DDoS Attack Detection and Mitigation Framework Using Machine Learning for Stateful SDN-Based IoT Networks . . . . .	6
1.4.4	Distributed Denial of Service Attack Detection for the Internet of Things Using Hybrid Deep Learning Model . . . . .	7
<b>2</b>	<b>Experiment Methods</b>	<b>8</b>
2.1	Decision Tree . . . . .	8
2.2	Random Forrest . . . . .	9
2.3	K-Nearest Neighbours . . . . .	9
2.4	Performance metrics . . . . .	9
2.4.1	Accuracy . . . . .	10
2.4.2	Precision . . . . .	10
2.4.3	Recall . . . . .	10
2.4.4	F-1 score . . . . .	10
2.4.5	Area under the receiver operating characteristic curve (ROC AUC) . . . . .	11
2.4.6	Cross-Validation Score . . . . .	11
2.5	Creating a custom dataset . . . . .	11
2.5.1	Computed particular window entropy (CPWE) . . . . .	11
2.5.2	Computed packet rate feature (CPRF) . . . . .	12
2.5.3	Received flow packets standard deviation (RFPD) . . . . .	12
2.5.4	Received flow bytes standard deviation (RFBSD) . . . . .	12
2.5.5	Computed flow entry rate (CFER) . . . . .	12
2.5.6	Duration . . . . .	13
2.5.7	Flow count . . . . .	13
2.6	Ensemble voting . . . . .	13
2.7	Tools used . . . . .	13

<b>3</b>	<b>Design</b>	<b>15</b>
3.1	Data processing . . . . .	15
3.1.1	Data cleaning . . . . .	15
3.1.2	Data analysis . . . . .	16
3.1.3	Feature engineering . . . . .	16
3.2	ML training/testing . . . . .	16
3.3	Simulated environment . . . . .	16
3.4	Real-time application . . . . .	17
<b>4</b>	<b>Implementation</b>	<b>22</b>
4.1	CICDDoS-2019 dataset Jupiter notebook . . . . .	22
4.2	Traffic generation BASH . . . . .	23
4.3	Custom dataset Python . . . . .	24
4.4	Real-time SDN DDoS detection system Python/BASH . . . . .	25
<b>5</b>	<b>Testing</b>	<b>29</b>
5.1	Jupiter notebook testing . . . . .	29
5.2	Real-time DDoS detection system testing . . . . .	29
<b>6</b>	<b>Results and Conclusions</b>	<b>32</b>
6.1	CIC experiments results . . . . .	32
6.1.1	ML table scores (CICDDoS-2019) . . . . .	32
6.1.2	Random forest . . . . .	32
6.2	Custom dataset results . . . . .	34
6.2.1	ML table scores (custom dataset) . . . . .	34
6.2.2	Decision tree / Random forest . . . . .	34
6.3	Real-time application results . . . . .	35
6.4	Comparisons with other publications . . . . .	36
6.5	Conclusions & Improvements . . . . .	36
<b>7</b>	<b>Critical Evaluation</b>	<b>38</b>
	<b>References</b>	<b>40</b>
	<b>Appendices</b>	<b>43</b>
<b>A</b>	<b>Use of Third-Party Code, Libraries and Generative AI</b>	<b>45</b>
1.1	Third Party Code and Software Libraries . . . . .	45
1.2	Generative AI . . . . .	46
<b>B</b>	<b>Extra documentation</b>	<b>47</b>
2.1	Duplicate removal . . . . .	47

## List of Figures

3.1	Data pre-processing pipeline . . . . .	18
3.2	Network topology design . . . . .	19
3.3	Real-time detection system flow-chart . . . . .	20
3.4	Real-time detection class diagram . . . . .	21
4.1	Split of Benign and DDoS traffic (CICDDoS-2019) . . . . .	23
4.2	Network topology implementation . . . . .	24
4.3	Split of Benign and DDoS traffic (Custom dataset) . . . . .	26
4.4	SYN attack detection . . . . .	27
4.5	Benign traffic detection . . . . .	27
4.6	Real-time SDN network architecture . . . . .	28
6.1	CICDDoS-2019 ML model performance . . . . .	33
6.2	Feature importance for RF (CICDDoS-2019) . . . . .	33
6.3	Feature importance for RF (Custom dataset) . . . . .	35
A.1	AI assisted intellisense . . . . .	46
B.1	Duplicate removal . . . . .	47
B.2	Frequency Distribution (CICDDoS2019) . . . . .	48
B.3	Flow Duration (CICDDoS2019) . . . . .	48
B.4	sFlow traffic benign . . . . .	49
B.5	sFlow traffic DDoS . . . . .	49
B.6	Real-time test . . . . .	50
B.7	Decision tree confusion matrix (CICDDoS-2019) . . . . .	51
B.8	Random forest confusion matrix (CICDDoS-2019) . . . . .	52
B.9	GitLab Kabana dashboard . . . . .	52

## List of Tables

5.1	Real-time application test . . . . .	30
6.1	Model performance (CICDDoS-2019) . . . . .	32
6.2	Model performance (Custom dataset) . . . . .	34
6.3	Real-time DDoS detection tests . . . . .	35



# Chapter 1

## Background and Objectives

### 1.1 Background

#### 1.1.1 Problem overview

To begin, the problem trying to be solved is one characterised by the dynamic and ever-changing issue DDoS attacks. These attacks come in many forms and often are completely unpredictable when they may occur. Traditional methods involving firewalls for example, can often be too simple in its programming to handle modern attacks. Because of this, the use of ML models can be increasingly helpful in detecting and mitigating said attacks. Classification is task that ML models have become very good at, for this project, the successful conversion of network traffic into classifiable data would become paramount to its success.

The importance of working on this issue is something that this student experienced first hand while working during their industrial placement. During the middle of a working day in July 2024, Azure experienced a DDoS attack that left it unable to provide its services to the company and many others. This left the department unable to perform regular tasks for over an hour [5]. The damage and scale of these attacks can cause widespread damage in both financial costs and reputation to said companies/organisations. With the rise of cloud computing, the effects of these attacks could spread to multiple companies at once. For these reasons and others, the detection and mitigation of DDoS attacks has become an important problem to solve.

#### 1.1.2 Functional requirements & research question

For this project, it was a combination of engineering and research. As such, there is 1 minor research question and 3 functional requirements for this project. The research question is "Can an ensemble voting machine learning model be used to detect DDoS attacks with an accuracy rate above 99.9%?"

Here are the functional requirements for the real-time DDoS detection system. Only 3

were made due to the small software size expected of the application itself, as well as time constraints.

- **FR1** - It must correctly classify real network data-packet traffic into either DDoS or non-DDoS traffic.
- **FR2** - It must be using some form of machine learning to accomplish its classification, no other methods will be permitted to be used.
- **FR3** - It must run unassisted, able to classify traffic in real-time. Reacting to changing traffic conditions, able to change its output depending on if an attack is currently occurring or not.

### 1.1.3 DoS and DDoS

Distributed denial of service (DDoS) is a family of denial of service (DoS) attacks that can be broken down by type of attack. There is no widely agreed upon 'first' DoS or DDoS attack. However, according to the The Internet Protocol Journal Volume 7 Number 4 [6] one of the first DDoS attacks documented was the September 6 1996 Panix attack, a SYN Flood which is a protocol-based attack that exploits the Transmission Control Protocol (TCP). This protocol works via a 'three-way-handshake', where several packets are used to ensure a strong, consistent and secure connection. SYN floods exploit the protocol by continuously sending connection request packets to a victim and not following through with a response packet to complete the connection request. This leaves the victim with too many TCP connections to process, blocking further connections from being established.

Very broadly, the types of DDoS attacks can be categorised as:

- Volumetric - Overwhelm the network with a huge amount of useless traffic
- Protocol - Exploit an existing network protocol flaw to disrupt network connectivity
- Application layer - Specifically exploits a server running a web-based application
- Multi-vector - A combination of these three types of attacks

Each of these types of attacks can be further broken down into sub-categories and specific types of attacks such as Ping of death, where the attacker creates a packet too large to be handled by the network, potentially causing a crash or reboot. As well as SYN floods (which have already been discussed) among many others. The main attacks that are covered in this project are:

- **SYN flood attacks** - A protocol-based attack, as mentioned before it involves exploiting the TCP protocol to force too many connects open on a particular network.

- **Portmapper-based DDoS attack** - Another protocol-based attack, this more modern method of attack can exploit either TCP or UDP (User Datagram Protocol). Portmapping exploits the call protocol used for sending phone calls through the internet, sending requests with a large amount of data, amplifying the response and slowing the network.
- **UDP flood attack** - A volumetric attack, it floods the network and overwhelms firewalls with a large amount of UDP network packets.
- **UDP-based DDoS with lag** - Multi-vector version of the previous attack.
- **MSSQL** - A web application layer exploitation attack focusing on a protocol used by Microsoft SQL server.
- **NetBIOS-related DDoS attack** - An out-of-date and hardly used service, NetBIOS if enabled on the system and given public access to the internet can be flooded with spoofed IP addresses to overwhelm the victim device.
- **Lightweight Directory Access Protocol-based attack (LDAP attack)** - This attack involves abusing an open port 389 by sending requests that will then prompt a large amount of traffic responses. Slowing down the victim's network connectivity.

These 7 attacks were chosen as the main focus for this project for both their commonality in DDoS attacks, but also their representation of a variance of attack types. They are also attacks with enough data in the CICDDoS-2019 dataset [7] sufficient to train the ML models. Further reasoning of the choice of dataset will be discussed in a subsequent section.

#### 1.1.4 Machine learning

As part of the research surrounding this project, all the papers that were read used ML models in some form or another to assist their detection systems. Many used several academic datasets to pre-train their models for a classification task on simulated network traffic [8]. This allowed for a more dynamic form of identification, especially when it came to new forms of attacks (mentioned in the papers as 0-day attacks). The performance of these models could also be demonstrated as being far more accurate than possible without. Some listing an accuracy rate of 98.97%, with false alarm rate of 0.023 [9].

Regarding these ML models that have been used for this project, three established models were tested, as well as an ensemble voting methodology. These were, decision tree, random forest and k-nearest neighbours. The reason for the selection of these models was as follows, decision tree - simple and easy-to-use as well as highly performant for the amount of data that it needed to be trained on. Random forest, prone to higher accuracy than decision tree and less likely to overfit for the training data. It was also used by several of the academic papers that formed part of the research, one of the most important being Enhancing DDoS Attack Detection and Mitigation in SDN Using an Ensemble Online Machine Learning Model [8]

k-nearest neighbours, chosen for its simplicity and performance. It is generally less accurate than the others in the tests performed by the research papers but is straightforward to understand and implement. Similar reasoning was used for the ensemble voting model, with performance being even more important since several ML model's decisions were being combined.

## 1.2 Analysis

### 1.2.1 Overview of the solution proposed by this project

The solutions for this project can be broken down into multiple sections and stages. For the initial few weeks, research would be conducted on the subject topic itself. What machine learning models would need to be used, how to implement them, what datasets to select and why choose one over another? The next stages was experimenting and testing. At this stage code would start being written on a Google colab the Jupiter notebook and the first visualisations/cleaning of the datasets would begin. After this stage, two branching paths would emerge. The first was a continuation of the previous stages, with greater emphasis on testing on the ML models on the now clean dataset. The second path was the creation of a virtual environment, from which a virtual network was created and various DDoS attacks were tested on it. The last stages were improving the performance of the ML models on the CICDDoS-2019 dataset, as well as creating a custom dataset using the virtual environment, which would then be used to help create the real-time DDoS detection system.

### 1.2.2 Choosing a dataset

There were several datasets introduced at the beginning of the project. The first of these was the CIC-DDoS2019 [7] dataset, which had been created by simulating real DDoS attacks on a real network and then processing the network traffic into usable data flows using the CICFlowmeter [10]. Another dataset introduced was the NSL-KDD [11]. This dataset was based on the KDD Cup'99 [12] and was created in order to improve upon the previous one by removing duplicate data values. It is also intended for intrusion detection, rather than purely based on DDoS detection. This dataset also suffers great from the fact that its creation methodology is difficult to find or understand. Making replication of its data flows for a custom dataset impossible. For these reasons and more, the CIC-DDoS2019 was chosen.

### 1.2.3 Virtual environment

In order to complete some of the later sections of this project, a virtual environment had to be created in order to test a realistic network against realistic DDoS attacks. It would be required, for not only the final creation of a real-time detection system, but also to help create a custom dataset that would be needed to help inform said real-time detection

system. The design of the virtual environment was based on the research papers discussed further in this chapter [8]. Design of the environment itself is kept in 3.3

## 1.3 Research Method and Software Process

### 1.3.1 Software methodology

The Scrum methodology was used for this project. Each week was broken down into sprints where the objectives were listed on several GitLab issues organised into milestones. This methodology helped break down both the tasks for each week but also ensured each week had a dedicated problem that needed to be solved in time for the meeting with the supervisor. A Git Lab Kanban dashboard B.9 was also used to help with this process, helping to organise each issue on the project as well as determine its current stage. Whether or not it was in the backlog, in-progress, or ready to be reviewed before closing. There was also a weekly diary that would complement this process. It would list many of the issues listed as tasks for that week, the time spent on each of those tasks as well as listing which week it took place within.

### 1.3.2 Research methodology

The initial methodology involved reading the academic papers received at the beginning of the project. Over time, more research involved searching through program documentation, as well as reading code submitted by other researchers on Kaggle [13] to see what processes they had taken to complete their research and implementation. Eventually research became a continuous process in order to regularly check work already completed, but to also help break through a particularly difficult problem. One example of this was re-reading some of the first research papers to better understand how they had created their custom datasets so one could be created for this project.

## 1.4 Literature review

During the initial few weeks of the project most of the time spent was on reading several academic papers in order to gain a greater understanding of the project requirements and implementations by other academics. Several of these were provided before the start of the project as background research, more was discovered through online research and reading. One of these papers researched was ML-DDoS: A Blockchain-Based Multilevel DDoS Mitigation Mechanism for IoT Environments [14] which provided some very useful background information about recent DDoS attacks, as well as their effects on embedded devices. Its novel approach that utilises blockchain technology was an interesting way of obtaining heuristics for their detection models. However this, and many other papers were not the main focus of this project. Each of the papers used for this project are listed with a review of each of them.

### **1.4.1 Enhancing DDoS Attack Detection and Mitigation in SDN Using an Ensemble Online Machine Learning Model**

This paper describes its creation of a complex machine learning-based intrusion detection system (OML-based IDS) and an OML-based IPS. Both of which form the application-level networking portion of the detection system. Together they use a pre-trained model that continuously ingests new traffic data in order to ensure the system as a whole is able to proactively adjust to new threats. The IDS system is the pre-trained section that informs the IPS, which then uses bidirectional tuning of the classification rules. The experiment environment which was used to test their system was very similar to the one created for this project.

One positive of this paper is its detail, especially in how it constructed its DDoS detection system and its simulated environment. Clear diagrams with data illustrations are used to demonstrate their points. Negative are its lack of information what network topologies it tried, as well as very little on how the data features created its custom dataset were calculated. Only displaying the results of its custom dataset without a clear indication of how it was converted. There is also the issue of artificiality generated data, which this paper does not address at all. [8]

### **1.4.2 Detecting DDoS Threats Using Supervised Machine Learning for Traffic Classification in Software Defined Networking**

This paper demonstrates its creation of a custom dataset, benchmarked against other popular DDoS detection datasets, the usage of novel ML techniques used on both that is then illustrated on how it could be used for real-world applications. The paper itself contains a well-structured and descriptive list of the work it has completed and reasons for each choice during the research and development of their system.

In terms of positives, this paper claims one of the highest numbers of data features on one of the largest custom datasets out of the papers used for this report. 79 features and over 1.1 million instances of DDoS and benign traffic, making it unique in the amount of training data that could be used for the improvement of the models. The paper also goes into a great deal of detail when explaining their process for collecting this vast number of data points. It also addresses the issue of synthetic data generation, unlike some of the other research papers. Negatives are its lack of testing on a real-time detection system. Since, the real-time system proposed in this paper is one only in theory, not one that has allowed the models that have been trained on this huge dataset to be truly tested in a real-time environment. [9]

### **1.4.3 FMDADM: A Multi-Layer DDoS Attack Detection and Mitigation Framework Using Machine Learning for Stateful SDN-Based IoT Networks**

This paper details its novel data collection and processing technique that would form a large part of this report. Its modular approach to collection, processing and detection

provided an elegant explanation of its real-time detection system. The system itself focused on internet of things (IoT) devices being used as botnets for DDoS attacks. The data features collected for its custom dataset were Computed particular window entropy (CPWE), Computed packet rate feature (CPRF), Received flow packets standard deviation (RFPSPD) and Received flow bytes standard deviation (RFBSD). Further discussion of these data features is provided in 2.5.

In terms of positives, this paper contains the best explanation of its unique data features from the list read. Clearly laid out with mathematical illustrations of how each is calculated and why they were useful features to be collected. In terms of negatives, it uses many unique names for each of its module sections of the real-time detection application. This can lead to confusion when trying to understand which tool is being referred to while reading. [3]

#### **1.4.4 Distributed Denial of Service Attack Detection for the Internet of Things Using Hybrid Deep Learning Model**

This paper, similar to the paper before, specialises in IoT-based DDoS attacks and their effects on the wider world. It also utilises a series of deep learning (DL) techniques such as long short-term memory (LSTM), 1D convolutional neural network (CNN), and a 2D convolutional neural network (CNN) as well as ML techniques like K-Nearest neighbours (KNN). While these DL techniques were never used in this project, it did provide an interesting alternative approach compared to the typical ML techniques used by most of the other papers researched for this project.

Advantages of this paper is its deep analysis of the ML models used on its chosen dataset. A lot of time is spent analysing why certain techniques perform well and why some do not. Disadvantages are that it does not contain its own custom dataset or real-time application. This gives the paper a lot of focus, but this project required both. As such, the gaps created from this focus would need to be filled by other research surrounding the topic. [15]

## Chapter 2

# Experiment Methods

### 2.1 Decision Tree

A decision tree is a form of ML that can perform both regression and classification, it is a supervised form of ML. The starts with a root node that continuously expands through a set of decision nodes, each with their own leaf nodes that contain all possibilities for the data to take. Visibility is a huge asset with this model, as all potential paths can be viewed along the tree structure to see where the decisions of certain sections of the dataset are made. Decision trees will traverse these decision nodes until they reach their leaf nodes, each time using a simple rule-set until the data class has reached its intended leaf node.

An important aspect of decision tree is entropy and information per decision gained. Entropy in decision tree is broadly calculated as the 'messiness' of the its decisions on any given dataset. If it is high, it means the data has a multitude of classes for each data point. Low means that the data points stick to one type of class, making it more homogeneous and orderly. The reason for its importance it is a very good indicator of how well the decision tree is splitting its decisions into the correct classes. Generally the lower the entropy the better the decision tree will perform. Here bellow is a calculation of entropy for decision tree according to IBM.

$$\text{Entropy}(S) = - \sum_{c \in C} p(c) \log_2 p(c)$$

[16]

In this project it was used to classify the various forms of network traffic. The tree in decision tree is formed by a series of boolean logic gates using probabilities to determine which 'path' to traverse through the tree. This model worked incredibly well in practice, which will be discussed further in chapter 6.

Decision tree was picked for this project for several reasons. Firstly, its white-box structure would allow for analysis of it performance, particularly when random forest would be used later on. Its performance would also allow for a much more iterative



process when it came to experimentation. Other models that were tried could sometimes take over an hour to complete their training process, seriously slowing down testing of improvements. It also works well with non-normalised or correctly scaled data. For these reasons, decision tree was used for this project.

## 2.2 Random Forrest

Random forest (RF) is itself an ensemble ML method. Using multiple decision trees together in a forest which then gets an element of randomness added to it, this randomness is sometimes called 'bagging' and it involves generating a random subset of data features. This lowers the correlation between trees in the forest, which adds a level of dynamism to model making it better at working on a generalised dataset. It also helps with avoiding a problem known as 'overfitting' [17], which is when the ML model becomes overtrained on a specific kind of data. This stop the model from truly 'learning' what the data means. If new data would be introduced to an overfit model, it would likely fail at classifying said new data properly and would need to be re-trained as a result. This would turn out to be a serious problem when generating the custom dataset, with overfitting becoming a common feature. This is discussed later in 6.2

## 2.3 K-Nearest Neighbours

K-Nearest Neighbours (KNN) is another supervised classification and regression ML model that uses 'proximity' of data values in order to predict which class they belong to. It is decision based, taking a particular data point and comparing its closeness to other data points to help it decide if one belongs to a certain defined data category. In this project, the classification would be used to determine which data metrics match DDoS vs benign traffic. Euclidean distance was the standard distance calculation used by SciKitlearn [18] and was implemented for this project. This is defined as the straight line distance between two points without barriers. The calculation for this is demonstrated bellow.

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

[19]

## 2.4 Performance metrics

Several types of performance calculations were used on each dataset/ML model. This is to deterime the effectiveness of each one against another, as well as determine if there are any flaws in the implementation of the algorithms. The main basis for all of these

different values are the metrics of true positives (TP), false positives (FP), true negatives (TN) and false negative (FN). Below are listed the following performance metrics were chosen, along with an explanation of why they were chosen:

### 2.4.1 Accuracy

Accuracy is calculated by adding the TP and TN values, then dividing that by the sum of TP + FP + FN + TN. Showing how 'correct' the model is when making a prediction about whether or not a particular data row is benign or DDoS traffic.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}$$

### 2.4.2 Precision

The precision score is calculated by dividing the TP value by the sum of TP and FP. Demonstrating how similar each prediction is to another, or in other terms. How consistent the ML model is at making its accurate (or inaccurate) predictions.

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

### 2.4.3 Recall

Recall is calculated by dividing TP by TP + FN, working similarly to the precision score. Its purpose is very different however, being used to demonstrate how close the ML model has come to finding all the instances of a specific class. I.e. benign or DDoS (split into several classes).

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

### 2.4.4 F-1 score

The F-1 score is calculated by multiplying 2 x recall x precision and then dividing that by the recall + precision. The F-1 score takes the already calculated precision and recall scores and takes the mean of these two scores to generate a new indicator of the model's overall performance.

$$\text{F-1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \cdot \text{TP}}{2 \cdot \text{TP} + \text{FP} + \text{FN}}$$

### 2.4.5 Area under the receiver operating characteristic curve (ROC AUC)

This metric is a probability calculation. It is used to provide a more precision indication of accuracy. It works by plotting the true positive rate against the false positive rate in a 2D representation, allowing a comparison of whether the model is better or worse than random assignment. I.e. if the model were to just simply guess each prediction instead of using a ML model, how well would the ML model outperform it.

### 2.4.6 Cross-Validation Score

A prediction of how well the ML model will perform on potential new data, i.e. how well is it at generalising the problem it has been tasked with solving. This metric was calculated for the ML models but was not included in the final metrics.

## 2.5 Creating a custom dataset

The creation of a custom dataset required both the creation of new data tools, as well as a compromise on the number of data features that could be used compared to the CICDDoS-2019 dataset. This was a decision made as a result of the lack of tooling available to experiment with this subject. The tool originally used to create the CICDDoS-2019 dataset has not been maintained for several years and was impossible to get running even with IntelliJ IDEA [20]. As a result, a custom methodology had to be used in order to fulfil the project outline objectives of creating a custom dataset and a real-time DDoS detection system. Since, without a method of capturing and processing the network traffic into usable data it would have been impossible. In order to solve this problem a solution was found after re-reading the research papers given to the student at the beginning of the project. On pages 10 and 11 of FMDADM: A Multi-Layer DDoS Attack Detection and Mitigation Framework Using Machine Learning for Stateful SDN-Based IoT Networks [3] it provides the mathematical formulas for how they calculated their network traffic data features.

Broken down into 32 packet chunks, each calculation is performed on this list of packets to form a single row, with each being listed in columns as a data features. The details within the paper are sparse in some areas, specifically in how defines entropy calculations. As a result, many educated guesses had to be made for how some of these data features were calculated. Other aspects of how the detection/prevention system were also unable to be replicated, such as their specific SDN controller, as well as their ensemble ML model implementation. This may have formed part of a later problem in custom dataset results 6.2. The data features created for this project are as follows.

### 2.5.1 Computed particular window entropy (CPWE)

This data feature captures destination ip addresses and compares them to each other in a 32 packet chunk to see how different they are. This comparison depends on the

number of unique values. In this case, for benign traffic where the ip addresses stays the same for most of the flows, this data feature stays very close to the value of 1.0. In DDoS traffic however, there is a large variance in the ip addresses. For all of the attacks simulated, the ip addresses were randomised to better simulate a realistic attack pattern. This resulted in a data feature with a high correlation with DDoS attacks.

### 2.5.2 Computed packet rate feature (CPRF)

This value is the calculated number of data packets sent per second. This results in the average rate across each flow in a 32 packet window. This is the full calculation created by the paper:

$$\text{CPRF} = \frac{\text{FP}_{\text{Count}}}{T \cdot M_{\text{Int}}}$$

### 2.5.3 Received flow packets standard deviation (RFPSD)

This data feature is the standard deviation of the packet per flow (PPF) data feature. This is a relatively straightforward data feature compared to some of the others. Very similar to the next data feature in this list.

$$\text{RFPSD} = \sqrt{\left( \left( \frac{1}{f} \right) \cdot \sum_{i=1}^f (n - \text{AvgP}) \right)}$$

### 2.5.4 Received flow bytes standard deviation (RFBSD)

This data feature is the number of bytes transferred per flow. Similar to above but instead demonstrating how much data is actually being transferred per data packet instead of the volume of packets. Very useful indicator for attacks like ping of death or showing if information is being transferred on a busy, but otherwise 'normal' network.

$$\text{RFBSD} = \sqrt{\left( \left( \frac{1}{f} \right) \cdot \sum_{i=1}^f (n - \text{AvgB}) \right)}$$

### 2.5.5 Computed flow entry rate (CFER)

The number of flows that enter an open-flow switch per 32 packet chunk. This data feature was very difficult to replicate with just .pcap files as they would typically just be captured on once switch at a time. As a result, this data feature would only calculate the number of flows on a single 32 packet capture.

$$\text{CFER} = \frac{F_{\text{Count}}}{M_{\text{Int}}}$$

### 2.5.6 Duration

Duration is the time calculated in the dataApp.py file between each flow that is processed by the script. This would prove to be a very good predictor of DDoS or benign traffic during classification. Most likely due to DDoS having extreme variances in its duration times, either being very low or very high.

### 2.5.7 Flow count

This data feature represents the number of counted flows in a 32 packet chunk. As will be discussed further in the results section 6.2, this turned into the best predictor of benign or DDoS traffic.

## 2.6 Ensemble voting

Ensemble voting is the methodology chosen to combine multiple ML model's predictions to try and improve the prediction scores on the datasets. Hard voting was chosen for the implementation as well, this was due to its generally better prediction rates during testing. Hard voting works by taking the majority vote for a particular classification, it is a very simple implementation of ensemble that is computationally efficient. There are some advantages to using soft voting however.

Soft voting was also considered and tested. This works by taking a probabilities measurement of each decision made by each ML model. They are then all averaged, with the highest average probability being selected. This allows the prediction to better avoid bias when working with an imbalanced dataset. It was however, not needed in this project and hard voting provided almost exactly the same results.

## 2.7 Tools used

Several different ML tools were used for the experiments related to training the ML models in a Jupiter notebook:

- Python - A programming language used for both the Jupiter notebook as well as data processing and the real-time application
- Pandas - A data science library for Python, used to manipulate data
- matplotlib - A Python library to help visualise data

- seaborn - A Python data visualisation library, similar to Matplotlib
- Google Colab - A virtual machine environment where a Jupiter notebook can be created, tested and saved in a cloud environment without needing any setup beforehand.
- Sklearn.model\_selection - Training testing split, cross-validations and performance metrics
- Sklearn.metrics - Further performance metrics
- Sklearn.preprocessing - Encode data features
- Sklearn.ensemble - Used for random forest implementation
- Sklearn.tree - Used for decision tree implementation
- Sklearn.neighbors - Used for K-nearest neighbours implementation
- Sklearn.compose - Used for creating ensemble voting model

As well as the above tools, another set of tools were used to help create the virtual environment. They also serviced in the creation of the custom dataset and SDN-controlled real-time detection system.

- Python - A programming language
- BASH - A shell language used to run commands and code on the virtual machine
- Mininet - Used to create a virtual network on the virtual machine
- Hping3 - A shell library used create realistic DDoS attacks
- tcpdump - Used to capture data packets into .pcap files
- Wireshark - A network traffic tool that was used to both capture traffic and observe trends in already captured .pcap files
- Xterm - Used with Mininet in order to open multiple shell terminals on hosts throughout the network
- Iperf3 - Used to help generate benign traffic for the custom dataset
- Sflow-RT - A traffic visualisation tool that can help demonstrate the amount of traffic travelling through the network

## Chapter 3

# Design

### 3.1 Data processing

For both the CICDDoS-2019 and custom datasets, a visual design was created to demonstrate the pipeline that both would be process them before any data analysis or testing could be performed.

The CICDDoS-2019 dataset is a very large dataset (over 29GB) kept in csv files. This presented a very serious problem when it came to importing/processing the dataset. One method that was thought of was chunking the dataset, splitting it into manageable chunks and processing that way. However this felt far too restrictive and made the iterative process to trailing new techniques and processes very difficult. The solution chosen was using a different data format utilised by another researcher on Kaggle [13], which converted the dataset into .parquet files, dramatically reducing the storage usage of the dataset. This allowed the full dataset to be uploaded to a Google drive and then imported with a few lines of code to the Jupiter notebook.

#### 3.1.1 Data cleaning

The first step in processing the data would be to clean the dataset. This involved merging the data files into one large data frame so that it could be manipulated and understood using one codebase. Finding and fixing irregularities in the dataset would come next, such as columns with only one data value, null removal and reducing the dataset by removing highly correlated columns. A small part of this process could be considered data analysis, since, in order to fix some problems an understanding of the dataset would be needed.

Using Pandas [21] functions like `.describe()` and `.value_counts()` would help greatly in quickly gaining an understanding of the data columns and rows themselves. Pandas in general, would be used to a large extent during this stage as short, simple functions would allow for rapid changes and understanding of those changes.

### 3.1.2 Data analysis

Once the data cleaning is complete, the next stage would be to analyse the dataset in order to better understand what the dataset contains. This involves checking the column types, what each of them represent. Finding out how much of each type of DDoS attack is represented in the data, as well as what percentage of the data is benign traffic compared to the DDoS traffic. As part of this process tools like Matplotlib [22] would be used to create visualisations to help understanding. Comparisons would primary focus on the differences between DDoS and benign traffic. This would be done to help spot what data features best indicate the chances that something is either regular or malicious traffic, as well as what each type of attack indicates which.

### 3.1.3 Feature engineering

The next stage of the design for data processing would be feature engineering. This stage forms part of the analysis stage where certain data features would be chosen as more or less important to the training of the ML models. This would be where visualisations such as confusion matrices and distribution charts would be essential in understanding which data points fit well together and which do not. Many columns would be dropped during this stage, this would be done to both improve the speed of training, but also improve the performance of the models, as irrelevant data is removed from the dataset.

## 3.2 ML training/testing

Once the dataset is cleaned and fully understood, the ML models can be trained/testing on it. This would involve loading all of the models into the Jupiter notebook, splitting the data into a training and testing dataset. Various percentages would be tested but the first was a 80 20 percentage split. Before which the data would be encoded, a technique that converts data object values such as strings into numeric features. This is done so that the models can perform all of their calculations on the data columns correctly. Finally, the data will be scaled and the ML models trained on the dataset. The planned ML models to be used as mentioned before were decision tree, random forest, k-nearest neighbours and an ensemble voting system. There was also plan to use support vector machine (SVM) with linear kernel, but the training time used for it added anywhere between an extra 10 minutes to half an hour for something that initially only took 5 minutes. Its performance when compared with the other models only barely beat KNN in terms of accuracy, this meant that it could not be justified to be kept in the project.

## 3.3 Simulated environment

For the simulated environment, a simple network topology was created. Given it would be created using Mininet, keeping the design simple so that basic commands could be



used was imperative. Below is the visual diagram of how the network topology was intended to look 3.2, as well as a demonstration of how the software defined networking (SDN) controller would interact with the network. There is also a visual representation of malicious traffic being sent from the open internet to various hosts across the network.

All of this simulated infrastructure would be kept on a Windows 10 laptop. Separate Ubuntu VM's would handle both the virtual network and SDN controller, which would communicate with each of the simulated hosts using a REST API. The full environment works very similarly to Enhancing DDoS Attack Detection and Mitigation in SDN Using an Ensemble Online Machine Learning Model [8]. Sticking closely to the environment setup in which their system was tested in, but not the detection system itself. This way could best determine if any problems arise, the system itself would be mostly likely the source of the problem instead of the environment. This and several other papers have almost this exact setup, making

### 3.4 Real-time application

For the real-time application, the design took a more engineering based approach. For example, a class diagram was created in order to represent a more complex type of application. Later on during the project, this initial design would be totally unnecessary and overly complicated. Further detail will be covered in the implementation chapter. A UML use case diagram however did provide some useful information for the project implementation.

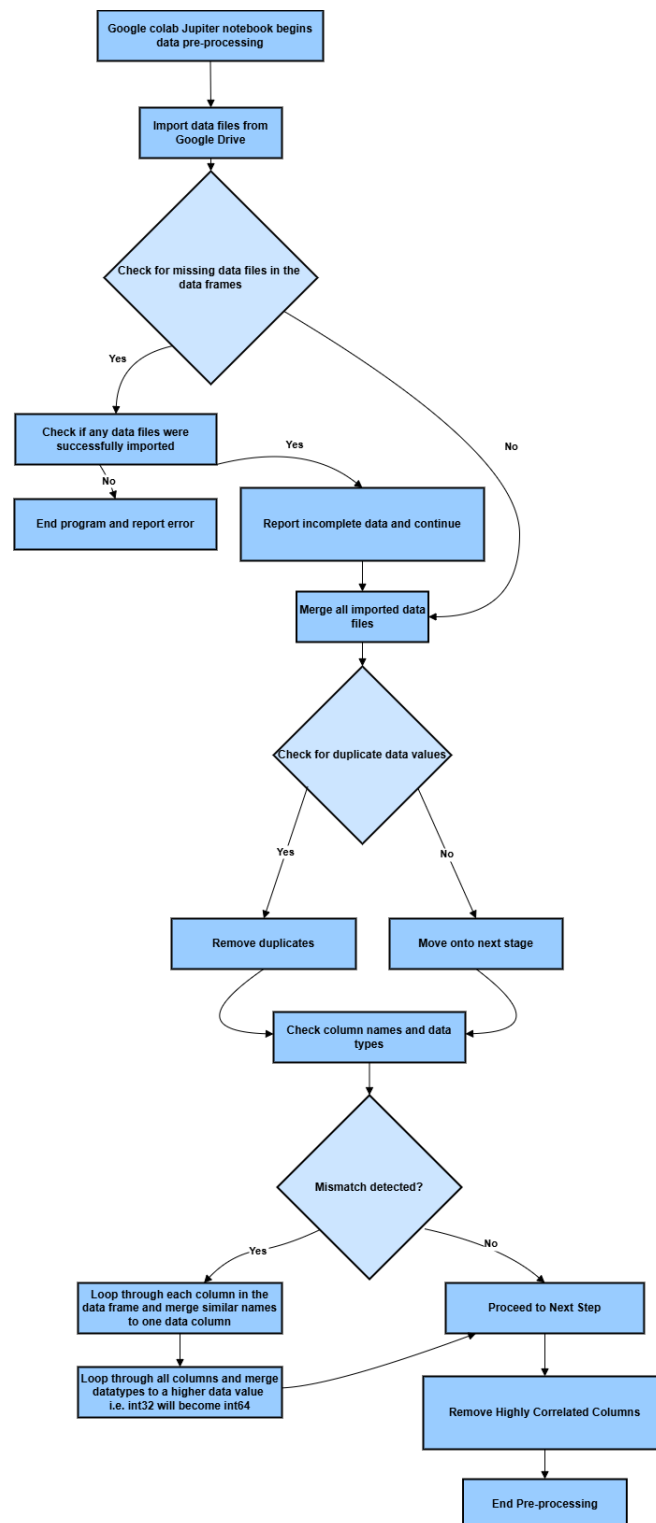


Figure 3.1: Data pre-processing pipeline

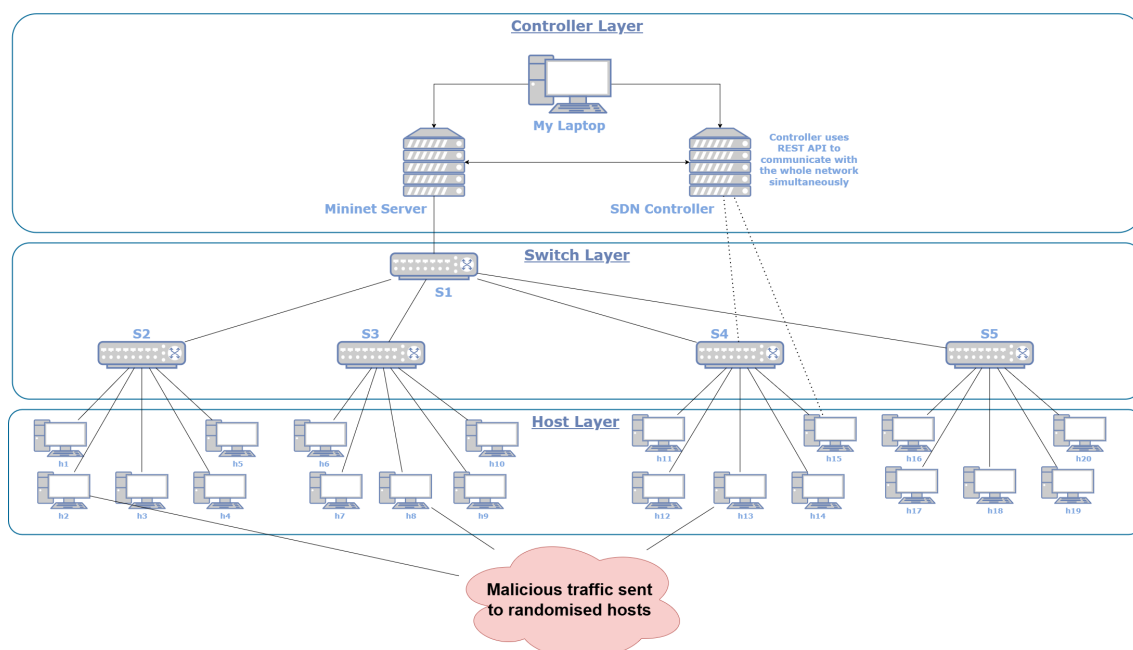


Figure 3.2: Network topology design

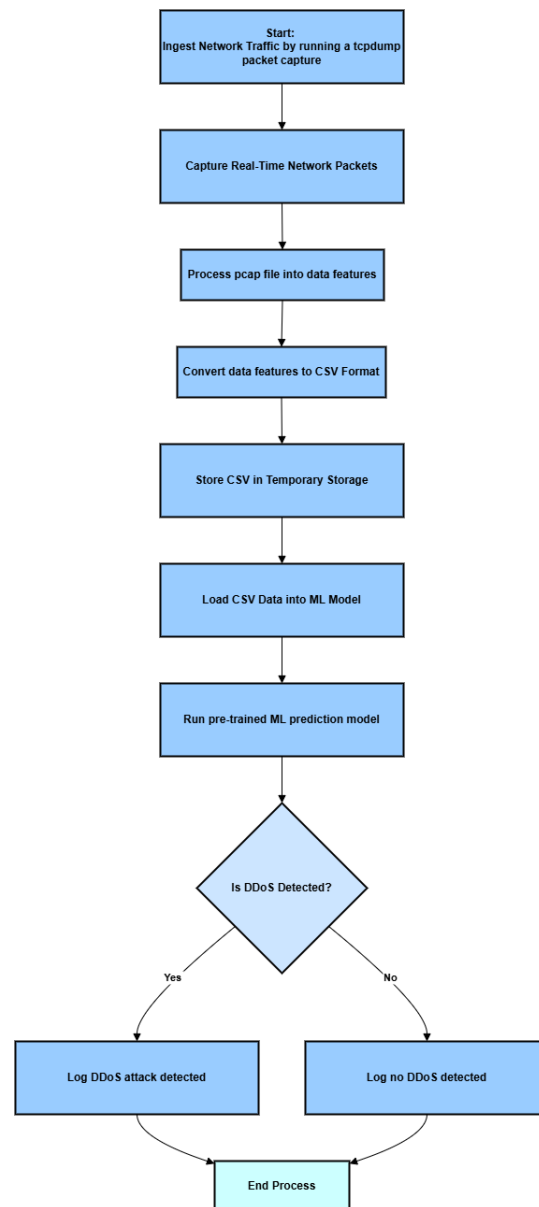


Figure 3.3: Real-time detection system flow-chart

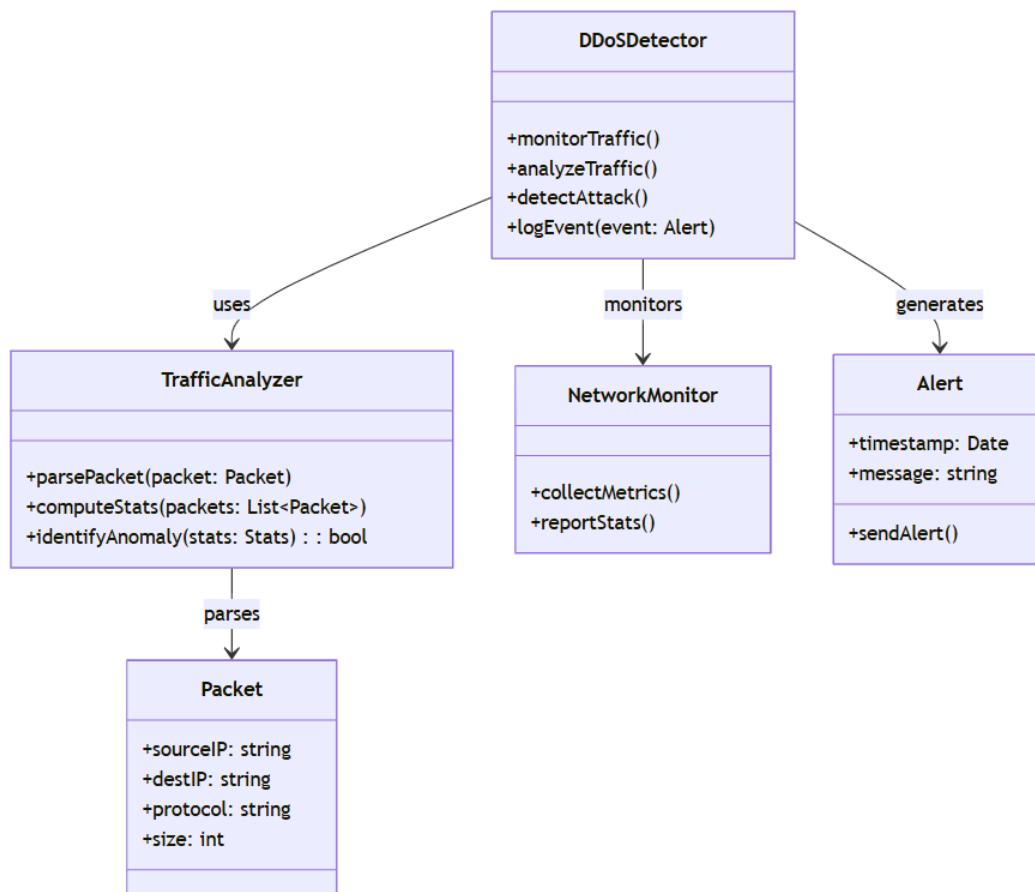


Figure 3.4: Real-time detection class diagram

## Chapter 4

# Implementation

### 4.1 CICDDoS-2019 dataset Jupiter notebook

The implementation of the CIC dataset was an iterative process following the stages listed in the design. In the beginning a decision had to be made about which parts of the full dataset to include, for example the DNS-testing.parquet file and several others which supposedly only contained 'testing' or 'training' data were not included. Other than this there were some initial issues when it came to importing the data files from a Google drive. Requiring mounting permissions that were not setup initially and once completed allowed for a seamless import of the files.

There were also many issues with dataset formatting of its columns that had to be corrected. For example, some columns will have the same name but might be listed as float64 in training but float32 in another. Here bellow is an example of the code used in the Jupiter notebook where this conversion took place, looping through each column and checking if there is a mismatch. By default it would convert whatever is the smaller data type to a larger data type.

```
if training_dtype == 'int64' and testing_dtype == 'int32':
    target_dtype = 'int64'
elif training_dtype == 'int32' and testing_dtype == 'int64':
    ...
```

Another example of this was data label mis-match. For example, one label on a dataset would be listed as UDPlag, while another would be listed as UDP-Lag. This again, would cause problems with the dataset and had to be corrected. This was resolved very similarly to the data type mis-match issue by looping through all the columns and changing each of the column names to the same string to avoid confusion during data analysis/ML training.

There were also no null columns found when cleaning the dataset, making this step a very easy and straightforward one. After this section was completed, the removal of duplicate data rows was ran. There turned out to be many more duplicates than expected, 7,417 total needed removal. A significant reduction in the size of the dataset.

Another smaller reduction on the dataset came by removing the WebDDoS DDoS attack rows. The number of rows was so small (only 51) that it was likely doing more harm than good by skewing the dataset values. To remove these values without disrupting the integrity of the dataset, the `.drop(index, inplace=True)` was used, shuffling the indexes of the data frame to avoid null rows forming and properly reducing the size of the dataset.

Here below is the distribution of benign vs DDoS traffic in the CICDDoS-2019 dataset. An important part of the data analysis this helps display how skewed the data is from one class to another.

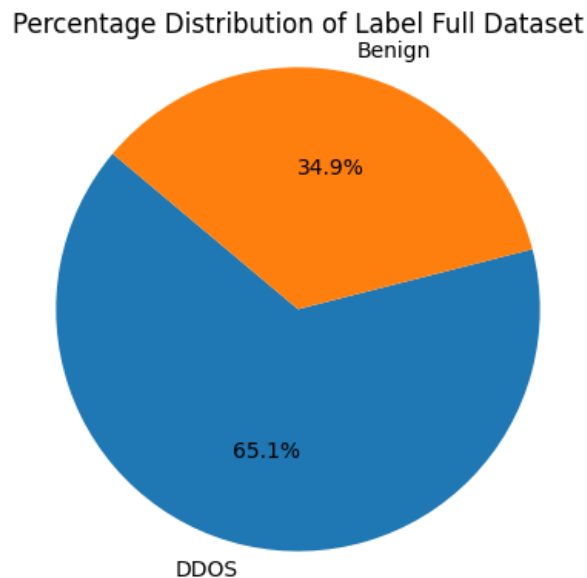


Figure 4.1: Split of Benign and DDoS traffic (CICDDoS-2019)

## 4.2 Traffic generation BASH

In order to fulfill this project's objectives, a simulated environment was created in order to both test and generate various types of DDoS attacks. The tools that were used for this were as follows:

- Ubuntu 24.04.2 LTS - A Linux distribution that was used as a simulated environment
- VirtualBox - An Oracle created virtual machine tool allowing for the use of other operating systems on the Windows laptop
- Mininet - A network simulation tool that uses OpenFlow Switches
- Hping3 - A networking traffic tool that was used to generate realistic DDoS attacks
- tcpdump - A network traffic capture tool similar to Wireshark
- Wireshark - A network traffic and analysis tool

This process began by installing Ubuntu 24.04.2 LTS [23] as a virtual machine on a Windows 10 laptop using the open source tool Virtual box developed by Oracle. Once installed on Virtual Box as a virtual machine, it was powered on and setup using the standard Ubuntu interface. After this step was completed, time was taken to install all the required libraries, packages and tools that would be needed for the experiments to run properly. As seen above, there were a considerable number of packages that required both installation and updates once installed. This took several days, with some packages being installed later as need for the project. Once completed, some initial tests were conducted on operability of the environment. Some bugs were discovered when trying to use the visualisation tools such as Sflow-RT, where the simulated networks would not appear to be correctly interfacing with the tool.

This problem was resolved after readding through the documentation of both Sflow-RT and Mininet, as there some special setup requirements that needed to be completed before the two programs could effectively communicate with each other. After this was completed, some time was dedicated to using visualisation tools to better illustrate the network topology implemented according to the design 3.2

Once the network topology was created, a visualiser tool called Narmox [24] was used to help see exactly how the network topology looked while running in a live environment. As can be seen in the figure network implementation figure 4.2.

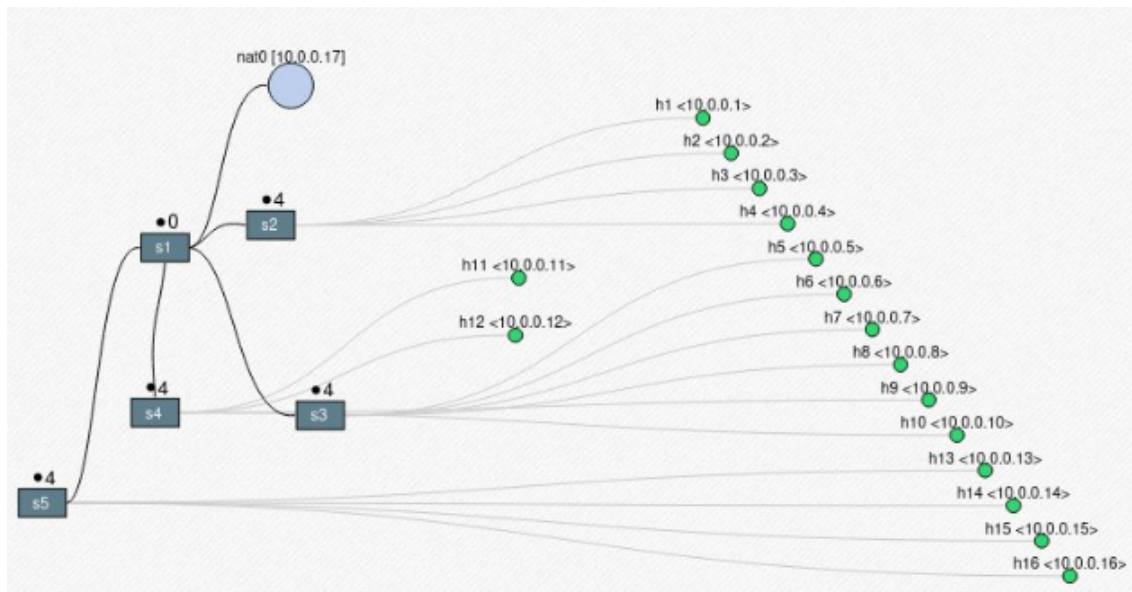


Figure 4.2: Network topology implementation

### 4.3 Custom dataset Python

Creating the custom dataset had many setbacks when implementing the design. The first problem encountered was the usage of the CICFlowmeter [10] tool. This Java application was created to assist in the development of network traffic datasets. It is the same tool that was used to create the CICDDoS-2019 dataset used in this project, and



would have been very useful in creating a new dataset. However, this tool had not been maintained by its developers for over 7 years. As a result, getting the program to compile properly, let alone work on a modern network was a challenge. After spending one sprint trying to get it working, it was realised that creating a new dataset using this old tool would be an impossible task.

If this problem was not resolved, creating a custom dataset and a real-time application would also be impossible as there would be no way to process any generated traffic into usable data. Upon further research, it seemed there were very few tools available publicly that could be used as a replacement for CICFlowmeter. However, re-reading the research papers given at the beginning of the project provided a potential solution. In FMDADM: A Multi-Layer DDoS Attack Detection and Mitigation Framework Using Machine Learning for Stateful SDN-Based IoT Networks [3], the researchers detail how they created their own method of processing raw network traffic into a usable dataset that they then later used to help train SDN controller to detect DDoS attacks.

As listed in chapter 2 these were Computed particular window entropy (CPWE), Computed packet rate feature (CPRF), Received flow packets standard deviation (RFPSTD), Packet per flow (PPF), Received flow bytes standard deviation (RFBSD) and Computed flow entry rate (CFER). Combined with the data features Duration and Flow-count a new method of transforming network traffic into usable data was available. While the code used to create the data was not available in the paper, there was enough detail in how each feature was created that an attempt at programming them could be made. This eventually resulted in the dataApp.py program.

This script would (as closely as possible) iterate through an entire .pcap file, stopping each time it reaches the end of a 32 packet window. Once this window is reached, the chunk of packets is used to calculate each of the data features and then write the output of each to a new data row to a .csv file until it reaches the end of said pcap file. An example of one data feature that was calculated was RFPSTD.

The final dataset generated for this project would contain this split of DDoS vs benign traffic. With a slightly higher level of DDoS traffic when compared with the CIC dataset.

## 4.4 Real-time SDN DDoS detection system Python/BASH

The real-time application used both the custom dataApp.py program to process new data, as well as several controller scripts written in both BASH and Python in order to make it all work together. The main file is the realTimeController.sh which runs continuously in the background while the virtual network is live. It begins by running a packet capture on the s1-eth1 interface, once it reaches 40 packets (the limit was set to 40 to avoid errors if packets were ever dropped) it will write all captured packets to the packetChunk.pcap file. The dataApp.py script will then be run to convert the packetChunk.pcap file into a usable packetChunk.csv data file.

Once converted, the trainedModel.py file is ran. This script will read the packetChunk.csv and use the pre-trained trainedCustomModel.pkl file to predict whether or not this particular 32 packet chunk was DDoS traffic or benign traffic. Depending on the answer,

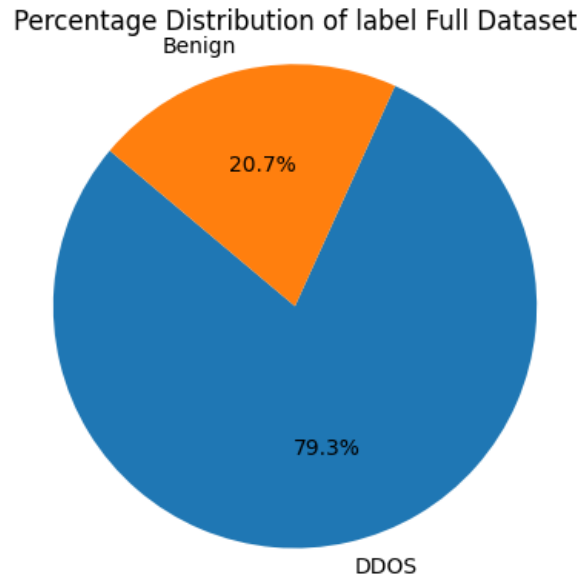


Figure 4.3: Split of Benign and DDoS traffic (Custom dataset)

it would write either a 1 or a 0 to the `isddos.txt` file, which the `realTimeController.sh` will then use to display that there either is or is not a DDoS attack occurring. It was also supposed to then stop a DDoS attack if detected by dropping network traffic from the hosts but there was not enough time to implement this feature correctly.

Below is an example of one of the tests run on the Ubuntu machine. Sflow-RT is running on the Chrome browser on the left, showing how much traffic is travelling through the network and its direction. On the right is the controller script running in the background. This example shows the detection of a SYN flood attack from a host outside the simulated network onto host 2 4.4.

And below here is another example, a minute or two after the SYN attack has been stopped. With some simulated benign traffic now running 4.5.

Now that the real-time application had been successfully created. A architecture diagram would be created to represent the final topology, could be compared to the design version in the previous chapter 3.2. The final SDN architecture would look like this 4.6 demonstrating attacks from both internal and external sources. With clear paths across the network for the SDN controller and sFlow REST API.

Several methodologies were used in determining how successful this real-time system was in detecting attacks. Further details of which are contained both within the subsequent testing chapter, but also in the results and conclusions chapter.

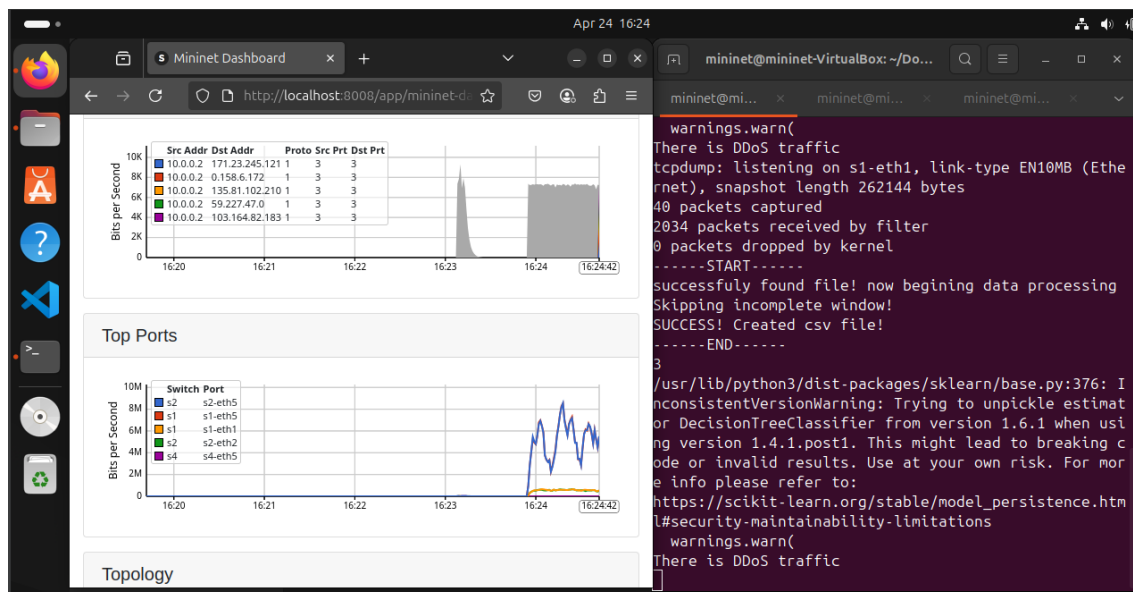


Figure 4.4: SYN attack detection

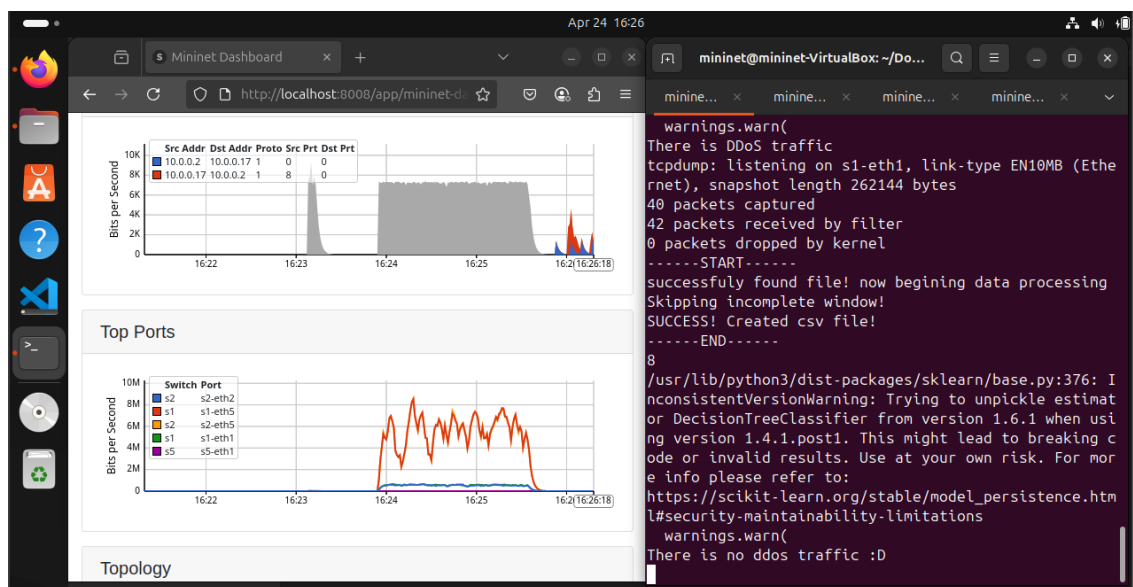


Figure 4.5: Benign traffic detection

]

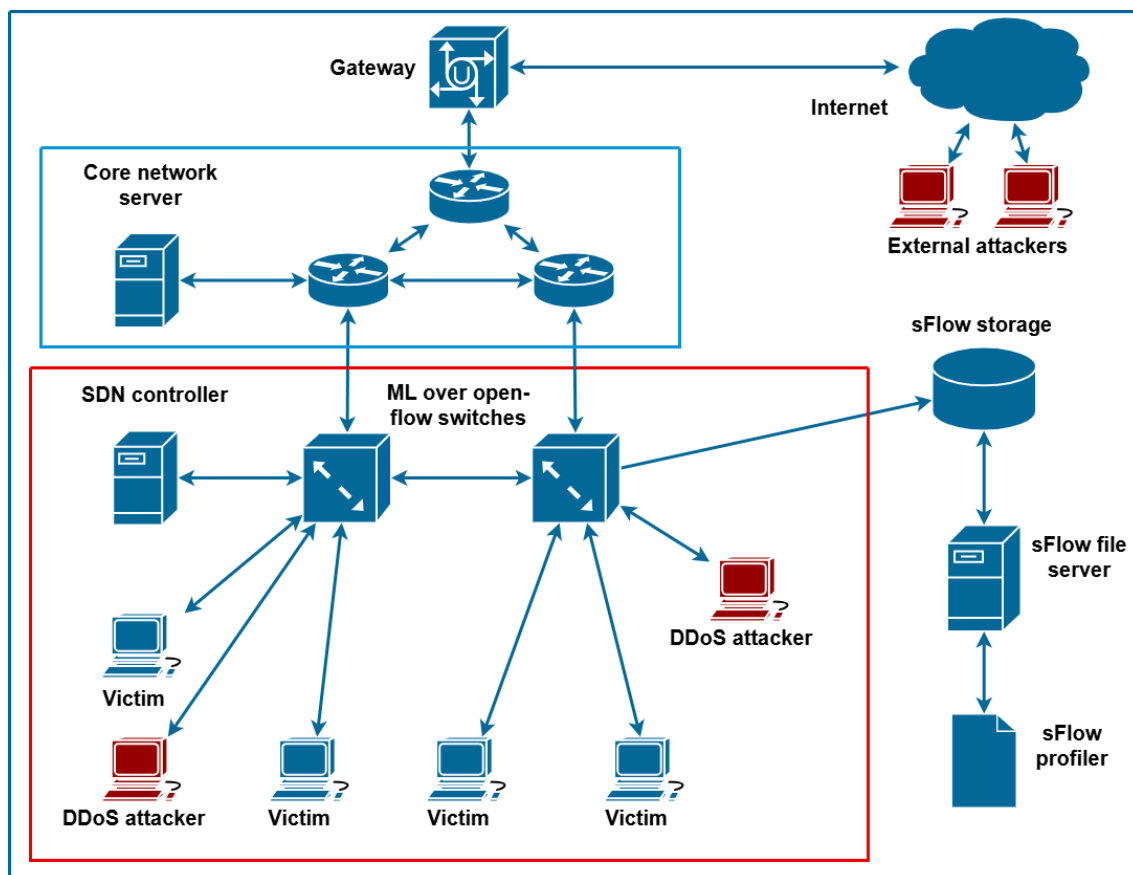


Figure 4.6: Real-time SDN network architecture

## Chapter 5

# Testing

### 5.1 Jupiter notebook testing

For this project a Google Colab environment was used which ran a Jupiter notebook. In this programming environment it is possible to run code incrementally, only running a section of code one at a time. Because of this, it becomes possible to test each section every time an experiment is run, even changing small sections of one cell to see its effects. The main method of testing implemented for this part of the project was prolific use of print statements for both output of functions, but also as a way of verifying if some functionality ran correctly. Here is one example of this in the DDoS-data-manipulation.ipynb which handles the CICDDoS2019 dataset:

```
if not mismatchedDtypesAfterCast:
    print("\nData type correction successful!")
else:
    print("\nData type correction may be incomplete")
```

### 5.2 Real-time DDoS detection system testing

Testing the real-time application could follow a more traditional engineering-based methodology, since, it works just like a regular software application. The following is the complete manual test table for the application.

Automatic unit testing was not completed for this application, although for the real-time detection application some automated testing most likely would have assisted in catching problems during development. An example of this could be giving the detection system several pre-made network pcap files with lists of which ones were DDoS or benign. The application would fail the automated tests if it predicted the traffic incorrectly, allowing changes to be made without having to setup the simulation every time to check whether or not the new changes had broken the detection system or made the predictions less accurate. The automated system that would be chosen if this were to be implemented would be Pytest [25]

Test ID	Description	Input	Expected Output	Actual Output	Pass/Fail
RT-01	Covert packets into data	tcpdump .pcap file	The dataApp.py file converts the .pcap into a .csv file	Files converted successfully	Pass
RT-02	Output correct prediction of DDoS traffic category for a SYN attack	DDoS Syn attack traffic across the mininet network	"There is DDoS traffic" written to the terminal	"There is DDoS traffic"	Pass
RT-03	Output correct prediction of benign traffic category	Regular pings and http traffic between network hosts	"There is no DDoS traffic :D"	"There is no DDoS traffic :D"	Pass
RT-04	Run continuously until finished	The script will be started in a terminal window	The script will complete its set number of loops and then exit	The script ran until it finished its designated number of loops, and then exited	Pass
RT-05	Collect network traffic	tcpdump will collect raw network packets into a pcap file	The dataApp.py file outputs successfully found file! now beginning data processing	successfully found file! now beginning data processing	Pass
RT-06	Process pcap file into csv data	Pcap file ingested by dataApp.py file	Output SUCCESS! Created csv file! —END—	SUCCESS! Created csv file! —END—	Pass
RT-07	Work with multiple topologies	Setup several different network topologies with the app running	App will either output there is no DDoS traffic or there is DDoS traffic correctly	"There is no DDoS traffic" and "There is DDoS traffic" correctly	Pass

Table 5.1: Real-time application test

The dataApp.py file, which handled the processing of raw packets into usable data, could have had better testing methods than what was implemented. The file itself used simple print statements and try-catch blocks, but not automated unit tests to determine if the application itself performed as expected within extreme examples. Below is a block of code that demonstrates the limited print statement/try-catch methods that try to give user feedback on what stage the program may have failed at.

```
if not os.path.exists(pcapPath):
    print("ERROR! 'No' file 'found!'")
    return

print("-----START-----")

try:
    rawPackets = rdpcap(pcapPath)
    print("successfully'found' file!'now'begining'data'processing'")
except Exception as e:
    print("ERROR! 'Could'not'process'packets'")
    return
```

## Chapter 6

# Results and Conclusions

### 6.1 CIC experiments results

The results of the DDoS-data-manipulation.ipynb Jupiter notebook provided a mostly successful completion of the intended accuracy scores of the CICDDoS-2019 dataset. Each of the full model's performances were logged in a table that was also converted into a visual representation to help illustrate the closeness of each of the models performances.

#### 6.1.1 ML table scores (CICDDoS-2019)

Each of the three models logged their performance scores each time they were trained. Below were the final scores logged for the CICDDoS-2019 dataset. As an overview, the scores are very promising, despite the fact that they are below the original target of 99.9%. However, after discussion with the supervisor it seems this was perhaps an unrealistic target and may even indicate overfitting for the models on this dataset, as well as the custom dataset which will be discussed later in this section.

#### 6.1.2 Random forest

Random forest consistently performed the best in terms of accuracy, precision and F-1 score. It did also have the longest training time by a large margin, though not large

Model	Training Time (s)	Accuracy	Precision	Recall	F1 Score	ROC AUC
Random Forest	17.215159	0.973742	0.974355	0.973742	0.972864	0.996387
Decision Tree	4.580372	0.971399	0.971621	0.971399	0.970496	0.984008
KNN	0.023968	0.966385	0.965724	0.966385	0.965311	0.963290
Hard voting classifier	22.95	0.9728	0.9733	0.9728	0.9717	0.9657

Table 6.1: Model performance (CICDDoS-2019)



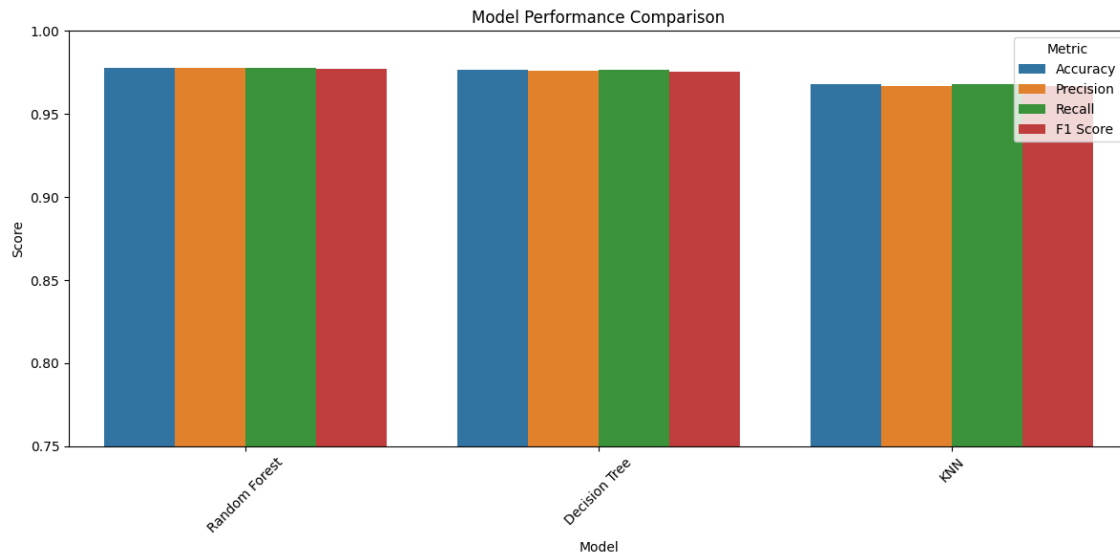


Figure 6.1: CICDDoS-2019 ML model performance

enough to represent a problem for the iterative testing process. When compared to the others though in terms of training time vs overall performance, it seems to generate diminishing returns on accuracy, precision, etc. KNN for example, reaching a similarly high score for less than a seconds worth of training time.

Here below are the data features listed from most to least important for random forest on the CICDDoS-2019 dataset. This graph was generated using Matplotlib [22]:

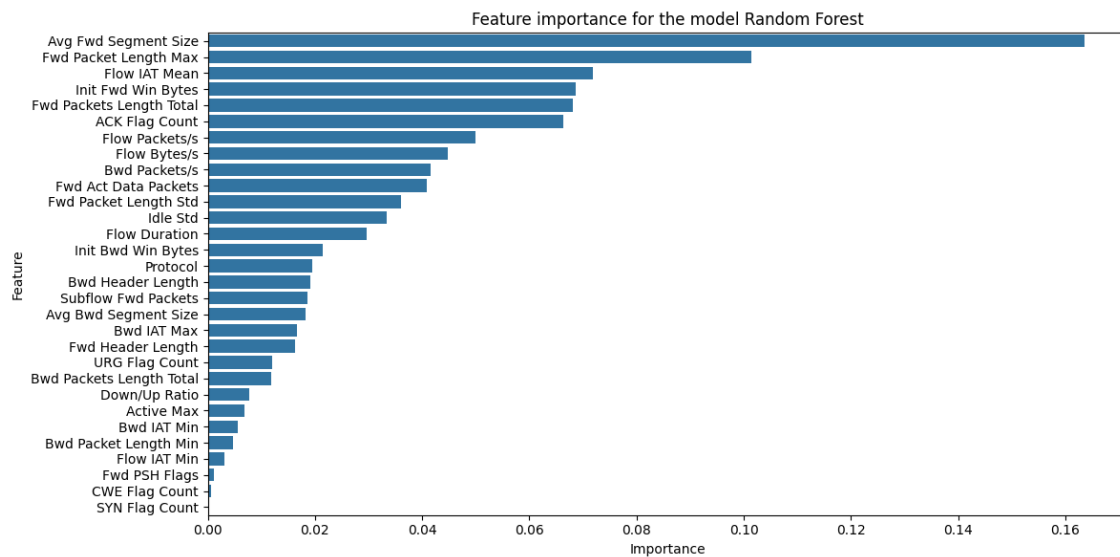


Figure 6.2: Feature importance for RF (CICDDoS-2019)

Model	Training Time (s)	Accuracy	Precision	Recall	F1 Score	ROC AUC
Decision Tree	0.350817	1.0	1.0	1.0	1.0	NaN
Random Forest	4.241582	1.0	1.0	1.0	1.0	NaN
KNN	0.172194	1.0	1.0	1.0	1.0	NaN

Table 6.2: Model performance (Custom dataset)

## 6.2 Custom dataset results

The custom dataset had many differences from the CICDDoS-2019 dataset. There of course is the obvious performance and 'overfitting' problems, as well as a lack of generalisation. But there were also interesting developments when it came to the data features when being trained on Random forest.

### 6.2.1 ML table scores (custom dataset)

For the custom dataset, the result were very different. At first glance they look very impressive, however the reality is that the models are completely suffering from a problem called 'overfitting'. Meaning that the models have not truly 'learned' how to solve the general problem but only how to fit the given data.

Some time was spent trying to understand what might have caused this problem. However, by the time this problem had been caught, there was very little time left to correct it. There many stages that would need to be explored in order to resolve the cause. Firstly, the testing environment. It is possible that the environment in which this data was collected was too artificial, or even worse, the methodologies used to generate both benign and DDoS traffic could be incorrect in some way. There is also the problem of the dataApp.py script, since it was created based on the interpretation of a PhD research paper. It is entirely possible (and in fact likely) that the script written is just incorrect. There are other possibilities beyond this, any of which could be the source of the issue. It is likely a combination of these problems are why this behaviour has emerged, unfortunately, time had to be spent on other aspects of the project before the deadline.

### 6.2.2 Decision tree / Random forest

For the custom dataset, each of the models did not have much that distinguished them from each other. Decision tree was listed as the best performing, and was chosen for the real-time application since it was slightly faster than random forest whilst giving the same accuracy score. There were some interesting visual results to inspect for the random forest algorithm, namely its feature importances.

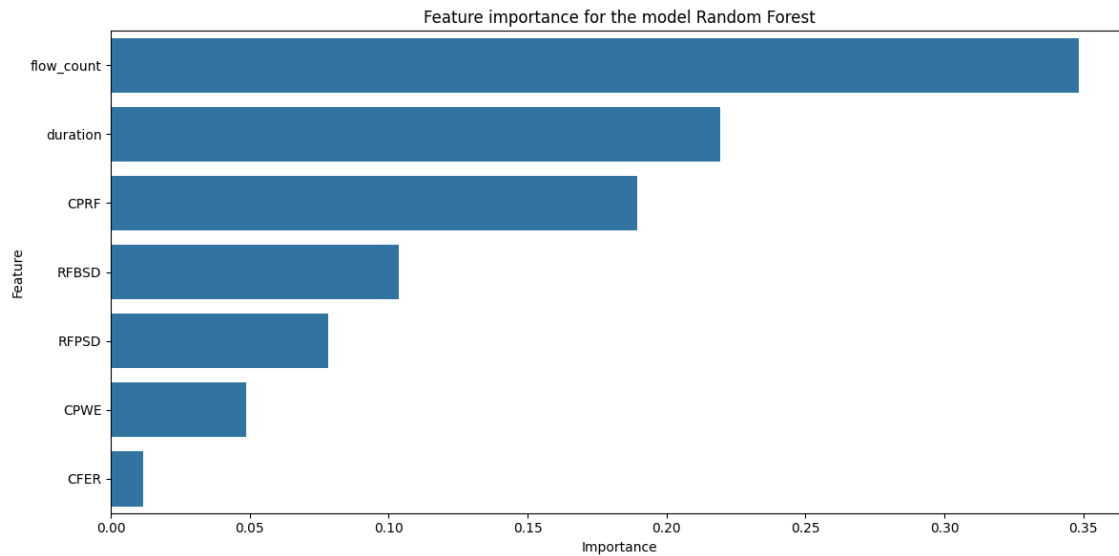


Figure 6.3: Feature importance for RF (Custom dataset)

Num of iterations	Correct pre-dictions	Time taken	Type of attack	Topology
30	30	09 mins 36 secs	Benign only traffic	Standard (16 hosts and 4 switches)
30	30	07 mins 33 secs	SYN attack	Standard
30	23	08 mins 17 secs	Benign only traffic	Large (13 switches and 27 hosts)

Table 6.3: Real-time DDoS detection tests

### 6.3 Real-time application results

For the real-time application, the results contained within the test tables would suggest the successful creation of a real-time detection system. Upon closer inspection, the system itself would likely not successfully detect a 'real' DDoS attack due to the models being overtrained on a specific set of scenarios. Some tests were created to help judge the real-time application's performance in the presence of real attacks, regular traffic, along with different types of network topologies. The test table is very limited, this was due to time constraints as the project was nearing its end when these tests took place.

The benign traffic that ran continuously through these tests were pings (h6 to h2 — h15 to h11 — h1 to h9) and then http traffic from the controller h17 to h3.

## 6.4 Comparisons with other publications

Compared with the other research papers used to help complete this project, the performance scores generally fell short of the achievements of the researchers. For example, in [4] the accuracy scores achieved were in the 99-99.9% range for the CICDDoS-2019 dataset. Much higher than the 97% achieved in the Jupiter notebook. When compared to the custom dataset, the performance scores are better than the examples listed by other research papers [3]. However, as discussed before, it is likely there is an issue with the dataset. Whether the problem lies with how it is process, or how it is generated. The result is the same, a failure to meet objectives set out at the beginning of the project.

The environment for simulating these experiments however, does match as closely as possible the ones created in Detecting DDoS Threats Using Supervised Machine Learning for Traffic Classification in Software Defined Networking [9]. Using a Windows 10 machine to simulate a Mininet virtual network on an Ubuntu virtual machine, with an SDN controller and DDoS attacks that were simulated using Hping3 and captured with tcpdump. This does indicate that the problems observed in the custom dataset may have resulted in a problem with the data conversion application and not with the simulated environment/traffic. It difficult to say for certain though, as the attacks themselves are not specified. The only example of which attacks were generated are the ones created for the CICDDoS-2019 dataset.

## 6.5 Conclusions & Improvements

There are many conclusions that can be drawn from the experiments created in this project. Firstly, the CICDDoS-2019 dataset is able to create a dynamic and rich dataset with its large number of data features, even when its data rows are reduced substantially. In the DDoS-data-manipulation.ipynb Jupiter notebook, a 97% accuracy rate was achieved, close to the scores achieved by many of the papers researched for this project. This score was also achieved with a multi-class framework, something unfortunately not accomplished on the DDoS-Custom-Dataset.ipynb Jupiter notebook.

For the custom dataset, there were many problems with its creation. While 'technically' able to detect several types of DDoS attacks and spot some differences between DDoS and benign traffic. There are still problems with the models being overfit. An example of this being when a large enough amount of benign traffic sent quickly enough can still be incorrectly classed as DDoS traffic. Not enough time has been spent either on testing of the real-time application, nor on the dataApp.py conversion script to ensure they are robust enough to handle DDoS attacks outside the simulated environment.

If this project were to be continued, several tasks would need to be completed. Firstly, more tests would need to be carried out to determine why the ML models are overfitting on the custom dataset. New data would need to be generated, preferably in a bare-metal networking environment to better simulate the networking environment. The use of multi-vector attacks would also likely make the data more realistic, potentially reducing

the likelihood of this problem.

Another feature that was not added was the blocking of DDoS traffic once detected. This problem had a simple solution that would utilise shell commands that interface with the open-flow switches used by Mininet to block traffic and drop packets. The reason for its lack of inclusion at the project was the same as a few other problems with the project, time. This and other aspects of the projects success will be discussed in the critical evaluation.

## Chapter 7

# Critical Evaluation

This project managed to achieve most but not all of the objectives set out in the project outline many weeks ago. To begin, the first deliverable was to create an ensemble ML model with a minimum 99% detection rate. While this has technically been achieved, there are some problems with its implementation. The ensemble model's overall performance is the same/worse than some of the other ML models, making its creation redundant. On top of this, the overall accuracy reached a high of 97.4% with random forest. Falling short of the original aim of the research question. One positive of this however, is that a better understanding of ML training has been gained, as such high accuracy rates are possibly the result of overfitting or bad generalisation.

The data report that was originally planned in the project outline, as it turns out, was not needed at all for this project. It was an interesting idea in theory, however the self-documenting nature of Jupiter notebooks made its inclusion irrelevant. The GitLab project however has been delivered fully and contains a wealth of documentation and code. Clearly adhering to the Scrum methodology with weekly sprints and a breakdown of each task into issues that would be staged based on their level of completion.

The virtual machine containing an SDN controller was mostly achieved. The SDN controller was less important than initially thought. However, the virtual environment was essential to many critical aspects of this project. It allowed for the testing of real DDoS attacks, the creation of a virtual network and without which a custom dataset and real-time detection system would have been impossible. As such, its completion merits some positives, especially as it managed to achieve all 3 of the functional requirements that were set out at the beginning of the project.

Leading on from this, the creation of a custom dataset had some successes, but mostly failed to live up to expectations. Upon initial testing with several data classes, the accuracy scores never managed to rise above 40%. When the classes were simplified to DDoS and benign traffic, the scores shot up to 100%. This clearly demonstrates overfitting of the models. When kept to a smaller number of classes the performance was better, around the 80% mark. However, this does not resolve the initial problem since the same experiments performed on the CICDDoS-2019 dataset did not have this issue. Because of how late in the project this problem was found, not enough time could be spent on trying to resolve or understand why this had happened. Though it is likely a

problem with how the dataset was generated due to it being created in a very artificial environment with a custom Python script created in a very short amount of time.

Despite the over-fitting of the model (or more likely because of it), the pre-trained model was successfully able to predict which traffic was benign and which was DDoS during tests. Marking that section of the project a success with plenty of room for improvement.

If repeated in the future, this project would need more time spent on the creation of a custom dataset. The environment and data gathering stage did not allow for the generation of a realistic dataset. One of the main factors of this is the lack of multi-vector attacks. Modern DDoS attacks very rarely use only one type attack, typically assaulting victim networks with an onslaught of multiple attacks at once. This make them both harder to predict, as the attacks use very different techniques, but also make them far more damaging to infrastructure. In this project, most of the attacks used one type of attack at a time, with a few multi-vector attacks used in some experiments.

To conclude, this project achieved its aims of creating a machine learning-based DDoS detection system. The system does contain many critical flaws which would need to be addressed if research were to continue, however, in the virtual environment created. It is able to correctly distinguish between a simulated DDoS attack and benign traffic. To improve, more time would need to spent on creating more realistic data, as well as research more/better data features to be used when converting network traffic into usable csv data.

An ensemble voting ML model was implemented and managed to achieve an accuracy rate of 97.4% which fell short of expectations. This, while being a disappointing result, is still respectable given the ambition of this project. Which leads to the final point, that this project tried to do too many things at once. Achieving some interesting results at the cost of consistency. Nonetheless, this student very much enjoyed the learning experience.

# References

- [1] Information Commissioner's Office. (2025, Feb.) Denial of service. ICO. [Online]. Available: <https://ico.org.uk/about-the-ico/research-reports-impact-and-evaluation/research-and-reports/learning-from-the-mistakes-of-others-a-retrospective-review/denial-of-service/>
- [2] Cloudflare. (2025, Jan.) Record-breaking 5.6 tbps ddos attack and global ddos trends for 2024 q4. The Cloudflare Blog. [Online]. Available: <https://blog.cloudflare.com/ddos-threat-report-for-2024-q4/>
- [3] W. I. Khedr, A. E. Gouda, and E. R. Mohamed, "Fmdadm: A multi-layer ddos attack detection and mitigation framework using machine learning for stateful sdn-based iot networks," *IEEE Access*, vol. 11, pp. 28 934–28 954, 2023.
- [4] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing realistic distributed denial of service (ddos) attack dataset and taxonomy," in *2019 International Carnahan Conference on Security Technology (ICCST)*, 2019, pp. 1–8.
- [5] K. O'Flaherty, "Microsoft confirms new outage was triggered by cyberattack," Jul 2024, accessed Apr. 17, 2025. [Online]. Available: <https://www.forbes.com/sites/kateoflahertyuk/2024/07/31/microsoft-confirms-new-outage-was-triggered-by-cyberattack/>
- [6] (2004, Dec.) The internet protocol journal, volume 7, number 4. [Online]. Available: <https://ipj.dreamhosters.com/wp-content/uploads/2022/07/ipj07-4.pdf>
- [7] (2019) DDoS 2019 Dataset. Canadian Institute for Cybersecurity, University of New Brunswick. [Online]. Available: <https://www.unb.ca/cic/datasets/ddos-2019.html>
- [8] A. A. Alashhab, M. S. Zahid, B. Isyaku, A. A. Elnour, W. Nagmeldin, A. Abdelmaboud, T. A. A. Abdullah, and U. D. Maiwada, "Enhancing ddos attack detection and mitigation in sdn using an ensemble online machine learning model," *IEEE Access*, vol. 12, pp. 51 630–51 649, 2024.
- [9] A. Hirsi, L. Audah, A. Salh, M. A. Alhartomi, and S. Ahmed, "Detecting ddos threats using supervised machine learning for traffic classification in software defined networking," *IEEE Access*, vol. 12, pp. 166 675–166 702, 2024.
- [10] (2017) Applications. Canadian Institute for Cybersecurity, University of New Brunswick. [Online]. Available: <https://www.unb.ca/cic/research/applications.html#CICFlowMeter>



- [11] (2025) NSL-KDD Dataset. Canadian Institute for Cybersecurity, University of New Brunswick. [Online]. Available: <https://www.unb.ca/cic/datasets/nsl.html>
- [12] (2025) KDD Cup 1999 Data. UCI KDD Archive. [Online]. Available: <https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>
- [13] Laurens. (2019) CIC-DDoS2019 Dataset. [Online]. Available: <https://www.kaggle.com/datasets/dhoogla/cicddos2019>
- [14] R. F. Hayat, S. Aurangzeb, M. Aleem, G. Srivastava, and J. C.-W. Lin, "MI-ddos: A blockchain-based multilevel ddos mitigation mechanism for iot environments," *IEEE Transactions on Engineering Management*, vol. 71, pp. 12 605–12 618, 2024.
- [15] A. Ahmim, F. Maazouzi, M. Ahmim, S. Namane, and I. B. Dhaou, "Distributed denial of service attack detection for the internet of things using hybrid deep learning model," *IEEE Access*, vol. 11, pp. 119 862–119 875, 2023.
- [16] (2021, Nov.) Decision trees. IBM. [Online]. Available: <https://www.ibm.com/think/topics/decision-trees>
- [17] M. Aslam, D. Ye, A. Tariq, M. Asad, M. Hanif, D. Ndzi, S. A. Chelloug, M. A. Elaziz, M. A. A. Al-Qaness, and S. F. Jilani, "Adaptive machine learning based distributed denial-of-services attacks detection and mitigation system for sdn-enabled iot," *Sensors*, vol. 22, no. 7, 2022. [Online]. Available: <https://www.mdpi.com/1424-8220/22/7/2697>
- [18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [19] (2017, Apr.) KNearest Neighbor(KNN) Algorithm. GeeksforGeeks. [Online]. Available: <https://www.geeksforgeeks.org/k-nearest-neighbours/>
- [20] ahlashkari. (2017) CICFlowMeter GitHub repository. [Online]. Available: <https://github.com/ahlashkari/CICFlowMeter>
- [21] (2024) pandas - Python Data Analysis Library. PyData. [Online]. Available: <https://pandas.pydata.org/>
- [22] (2024) Matplotlib — Visualization with Python. John D. Hunter, Michael Droettboom, et al. [Online]. Available: <https://matplotlib.org/>
- [23] (2025) Download Ubuntu Desktop. Ubuntu. [Online]. Available: <https://ubuntu.com/download/desktop>
- [24] (2025) SDN Narmox Spear. Narmox. [Online]. Available: <http://demo.spear.narmox.com/app/?apiurl=demo#!/mininet>
- [25] (2025) pytest documentation. pytest development team. [Online]. Available: <https://docs.pytest.org/en/stable/>

- [26] N. Taylor, "MMP\_S08 Project Report and Technical Work," <http://blackboard.aber.ac.uk/>, Feb. 2019, accessed February 2019.
- [27] W. Press *et al.*, *Numerical recipes in C*. Cambridge University Press Cambridge, 1992, pp. 349–361.
- [28] M. Neal, J. Feyereisl, R. Rascunà, and X. Wang, "Don't touch me, I'm fine: Robot autonomy using an artificial innate immune system," in *Proceedings of the 5th International Conference on Artificial Immune Systems*. Springer, 2006, pp. 349–361.
- [29] H. M. Dee and D. C. Hogg, "Navigational strategies in behaviour modelling," *Artificial Intelligence*, vol. 173(2), pp. 329–342, 2009.
- [30] Various, "Fail blog," <http://www.failblog.org/>, Aug. 2011, accessed August 2011.
- [31] S. Duckworth, "A picture of a kitten at Hellifield Peel," <http://www.geograph.org.uk/photo/640959>, 2007, copyright Sylvia Duckworth and licensed for reuse under a Creative Commons Attribution-Share Alike 2.0 Generic Licence. Accessed August 2011.
- [32] Apache Software Foundation, "Apache POI - the Java API for Microsoft Documents," <http://poi.apache.org>, 2014.
- [33] —, "Apache License, Version 2.0," <http://www.apache.org/licenses/LICENSE-2.0>, 2004.
- [34] Imperva, "Ddos attack types & mitigation methods," Dec 2024, accessed Apr. 17, 2025. [Online]. Available: <https://www.imperva.com/learn/ddos/ddos-attacks/>
- [35] T. E. Ali, Y.-W. Chong, and S. Manickam, "Comparison of ml/dl approaches for detecting ddos attacks in sdn," *Applied Sciences*, vol. 13, no. 5, 2023. [Online]. Available: <https://www.mdpi.com/2076-3417/13/5/3033>
- [36] —, "Machine learning techniques to detect a ddos attack in sdn: A systematic review," *Applied Sciences*, vol. 13, no. 5, 2023. [Online]. Available: <https://www.mdpi.com/2076-3417/13/5/3183>
- [37] (2025) Oracle VirtualBox. Oracle. [Online]. Available: <https://www.virtualbox.org/>
- [38] (2025) Enterprise Open Source and Linux. Ubuntu. [Online]. Available: <https://ubuntu.com/>
- [39] Mininet Project Contributors. (2022) Mininet: An Instant Virtual Network on Your Laptop (or Other PC). [Online]. Available: <https://mininet.org/>
- [40] (2024, May) hping3 — Kali Linux Tools. Kali Linux. [Online]. Available: <https://www.kali.org/tools/hping3/>
- [41] (2015) sFlow-RT. sFlow-RT. [Online]. Available: <https://sfllow-rt.com/>
- [42] (2025) OpenDaylight. OpenDaylight. [Online]. Available: <https://www.opendaylight.org/>

- [43] (2016) Welcome to OpenDaylight Documentation — OpenDaylight Documentation Potassium documentation. OpenDaylight. [Online]. Available: <https://docs.opendaylight.org/en/stable-potassium/>
- [44] V. GUEANT. (2025) iPerf - The TCP, UDP and SCTP network bandwidth measurement tool. [Online]. Available: <https://iperf.fr/>
- [45] (2024) Home — TCPDUMP & LIBPCAP. TCPDUMP & LIBPCAP. [Online]. Available: <https://www.tcpdump.org/>
- [46] (2025, Apr.) Welcome to Python.org. Python Software Foundation. [Online]. Available: <https://www.python.org/>
- [47] (2019) Google Colab. Google. [Online]. Available: <https://colab.research.google.com/>
- [48] (2024) seaborn: statistical data visualization. PyData. [Online]. Available: <https://seaborn.pydata.org/>
- [49] Awan-Ur-Rahman. (2023, Aug.) Understanding Soft Voting and Hard Voting: A Comparative Analysis of Ensemble Learning Methods. [Online]. Available: <https://medium.com/@awanurrahman.cse/understanding-soft-voting-and-hard-voting-a-comparative-analysis-of-ensemble-learning-methods-db>
- [50] (2025) Flowchart Maker Online Diagram Software. Diagrams.net. [Online]. Available: <https://app.diagrams.net/>
- [51] (2024, Feb.) How to Calculate Entropy in Decision Tree? GeeksforGeeks. [Online]. Available: <https://www.geeksforgeeks.org/how-to-calculate-entropy-in-decision-tree/>
- [52] S. Dash. (2022, Oct.) Understanding the ROC and AUC Intuitively. [Online]. Available: <https://medium.com/@shaileydash/understanding-the-roc-and-auc-intuitively-31ca96445c02>
- [53] B. L. Labs. (2015, Aug.) A New DDoS Reflection Attack: Portmapper; An Early Warning to the Industry. [Online]. Available: <https://blog.lumen.com/a-new-ddos-reflection-attack-portmapper-an-early-warning-to-the-industry/>
- [54] (2025) UDP flood DDoS attack. Cloudflare. [Online]. Available: <https://www.cloudflare.com/en-gb/learning/ddos/udp-flood-ddos-attack/>
- [55] (2017) CLDAP Used in DDoS Attacks. NHS England Digital. [Online]. Available: <https://digital.nhs.uk/cyber-alerts/2017/cc-1333>
- [56] (2025) What Is MS SQL Reflection Attack? — Knowledge Base. DDoS-GUARD. [Online]. Available: <https://ddos-guard.net/terms/ddos-attack-types/ms-sql-attack>
- [57] (2025) Wireshark. Wireshark. [Online]. Available: <https://www.wireshark.org/>

# Appendices

## Appendix A

# Use of Third-Party Code, Libraries and Generative AI

### 1.1 Third Party Code and Software Libraries

The full list of third party code and software libraries used for this project:

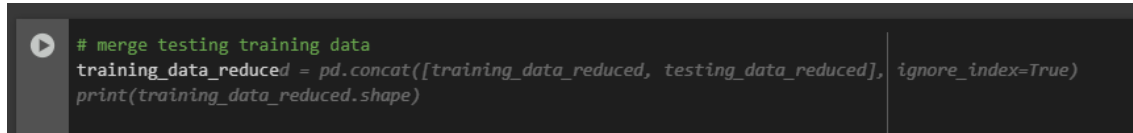
- Pandas [21]
- Scikit learn [18]
- Matplotlib [22]
- Seaborn [48]
- Draw.io [50]

Bellow is the software and libraries used for both the creation of a custom dataset, as well as the real-time detection system.

- Virtual box [37]
- Ubuntu [23]
- Mininet [39]
- hping3 [40]
- Sflow-RT [41]
- OpenDaylight (SDN) [42]
- Iperf3 [44]
- tcpdump [45]
- Wireshark [57]

## 1.2 Generative AI

Some AI-assisted intellisense was used in the Google colab Jupiter notebook. Below is an example of how it was used in this project to speed up the writing of code.



```
# merge testing training data
training_data_reduced = pd.concat([training_data_reduced, testing_data_reduced], ignore_index=True)
print(training_data_reduced.shape)
```

Figure A.1: AI assisted intellisense

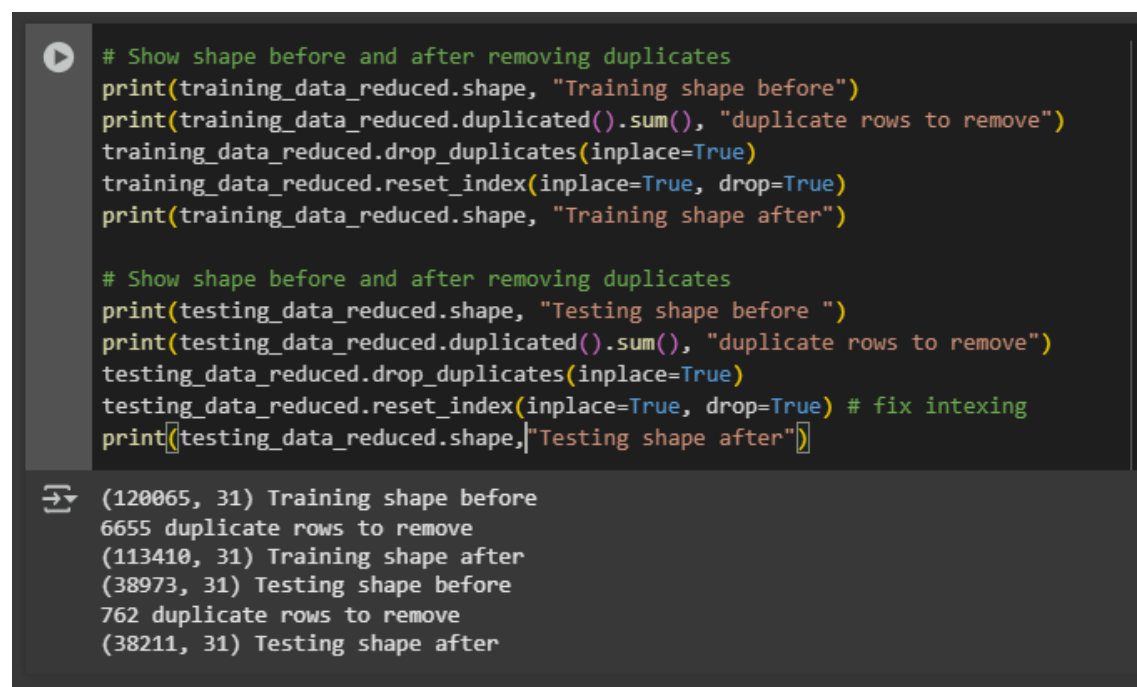
Other than this example, no other uses of AI tools were used in this project.

## Appendix B

# Extra documentation

### 2.1 Duplicate removal

A demonstration of the removal of duplicates on the CICDDoS-2019 dataset.



```
# Show shape before and after removing duplicates
print(training_data_reduced.shape, "Training shape before")
print(training_data_reduced.duplicated().sum(), "duplicate rows to remove")
training_data_reduced.drop_duplicates(inplace=True)
training_data_reduced.reset_index(inplace=True, drop=True)
print(training_data_reduced.shape, "Training shape after")

# Show shape before and after removing duplicates
print(testing_data_reduced.shape, "Testing shape before ")
print(testing_data_reduced.duplicated().sum(), "duplicate rows to remove")
testing_data_reduced.drop_duplicates(inplace=True)
testing_data_reduced.reset_index(inplace=True, drop=True) # fix indexing
print(testing_data_reduced.shape, "Testing shape after")
```

(120065, 31) Training shape before  
6655 duplicate rows to remove  
(113410, 31) Training shape after  
(38973, 31) Testing shape before  
762 duplicate rows to remove  
(38211, 31) Testing shape after

Figure B.1: Duplicate removal

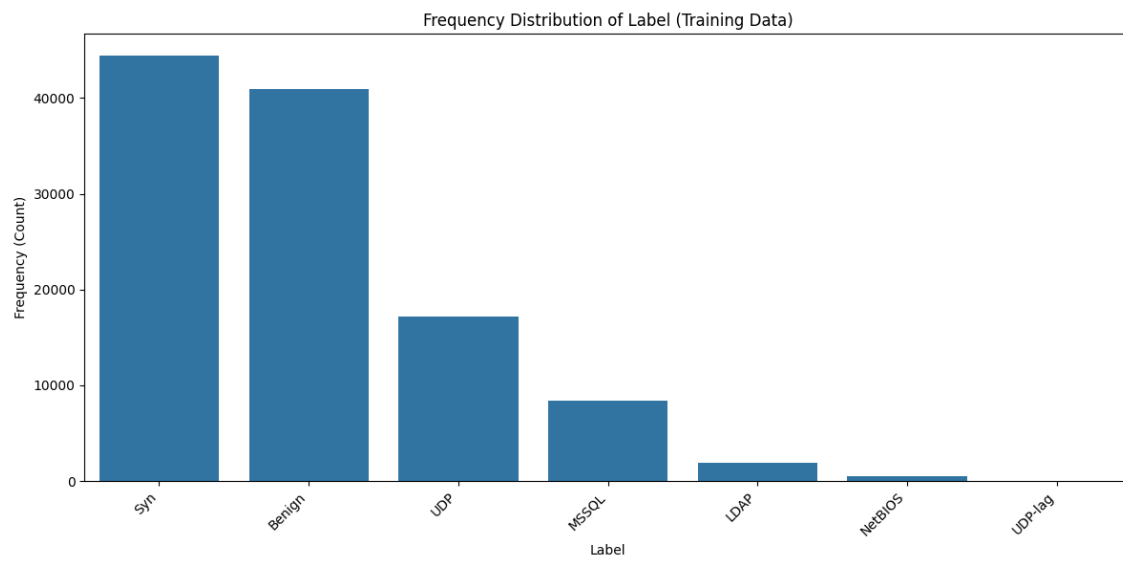


Figure B.2: Frequency Distribution (CICDDoS2019)

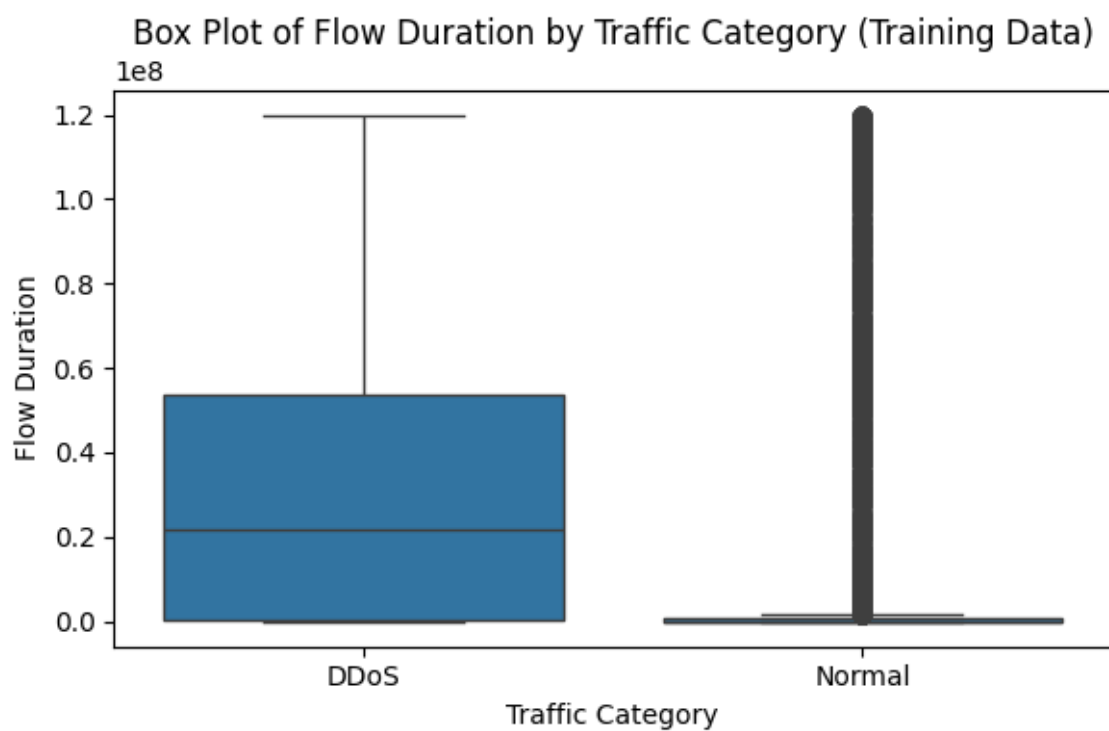


Figure B.3: Flow Duration (CICDDoS2019)



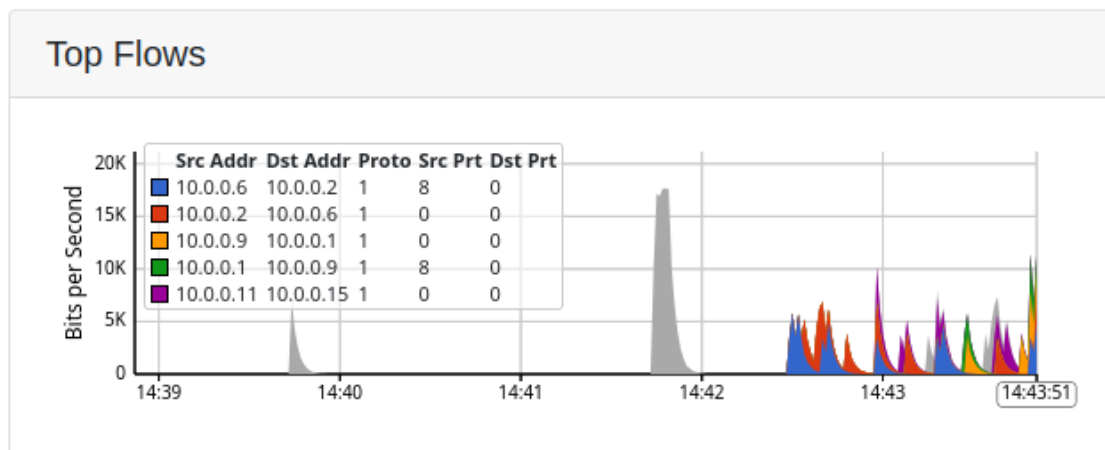


Figure B.4: sFlow traffic benign

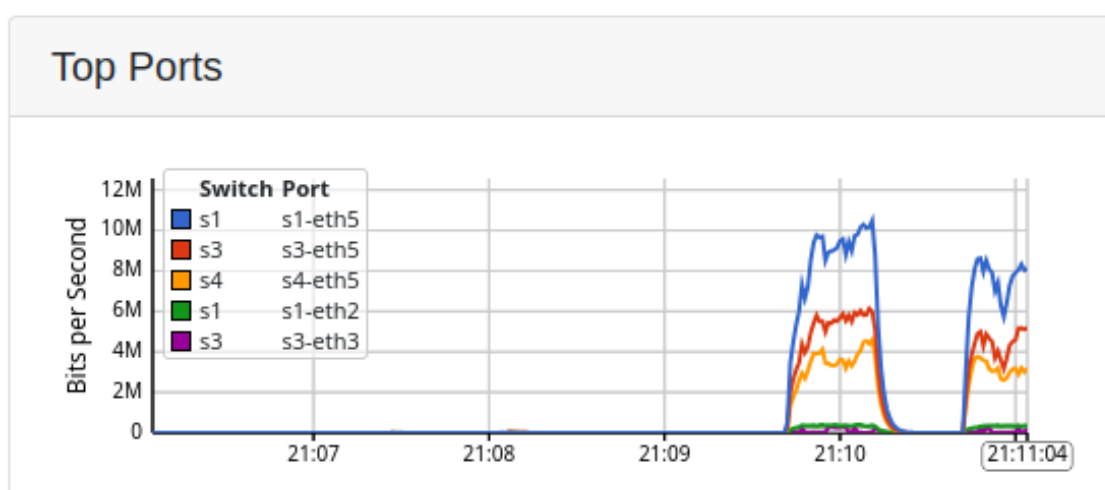
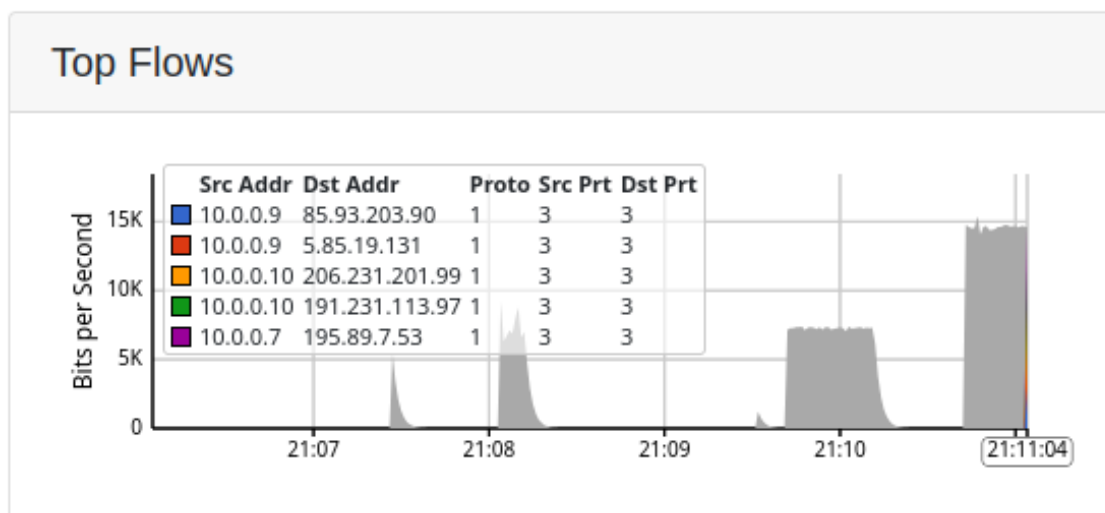
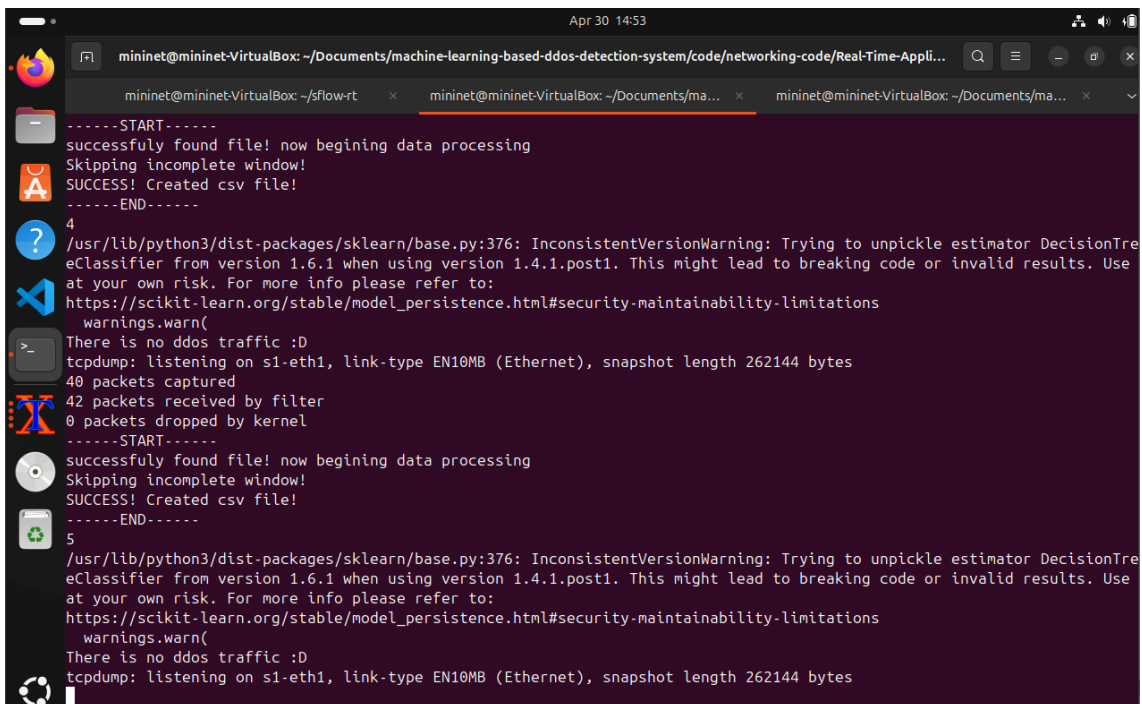


Figure B.5: sFlow traffic DDoS



```
mininet@mininet-VirtualBox: ~/Documents/machine-learning-based-ddos-detection-system/code/networking-code/Real-Time-Appli...
-----START-----
successfully found file! now begining data processing
Skipping incomplete window!
SUCCESS! Created csv file!
-----END-----
4
/usr/lib/python3/dist-packages/sklearn/base.py:376: InconsistentVersionWarning: Trying to unpickle estimator DecisionTreeClassifier from version 1.6.1 when using version 1.4.1.post1. This might lead to breaking code or invalid results. Use at your own risk. For more info please refer to: https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
There is no ddos traffic :D
tcpdump: listening on s1-eth1, link-type EN10MB (Ethernet), snapshot length 262144 bytes
40 packets captured
42 packets received by filter
0 packets dropped by kernel
-----START-----
successfully found file! now begining data processing
Skipping incomplete window!
SUCCESS! Created csv file!
-----END-----
5
/usr/lib/python3/dist-packages/sklearn/base.py:376: InconsistentVersionWarning: Trying to unpickle estimator DecisionTreeClassifier from version 1.6.1 when using version 1.4.1.post1. This might lead to breaking code or invalid results. Use at your own risk. For more info please refer to: https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
There is no ddos traffic :D
tcpdump: listening on s1-eth1, link-type EN10MB (Ethernet), snapshot length 262144 bytes
```

Figure B.6: Real-time test

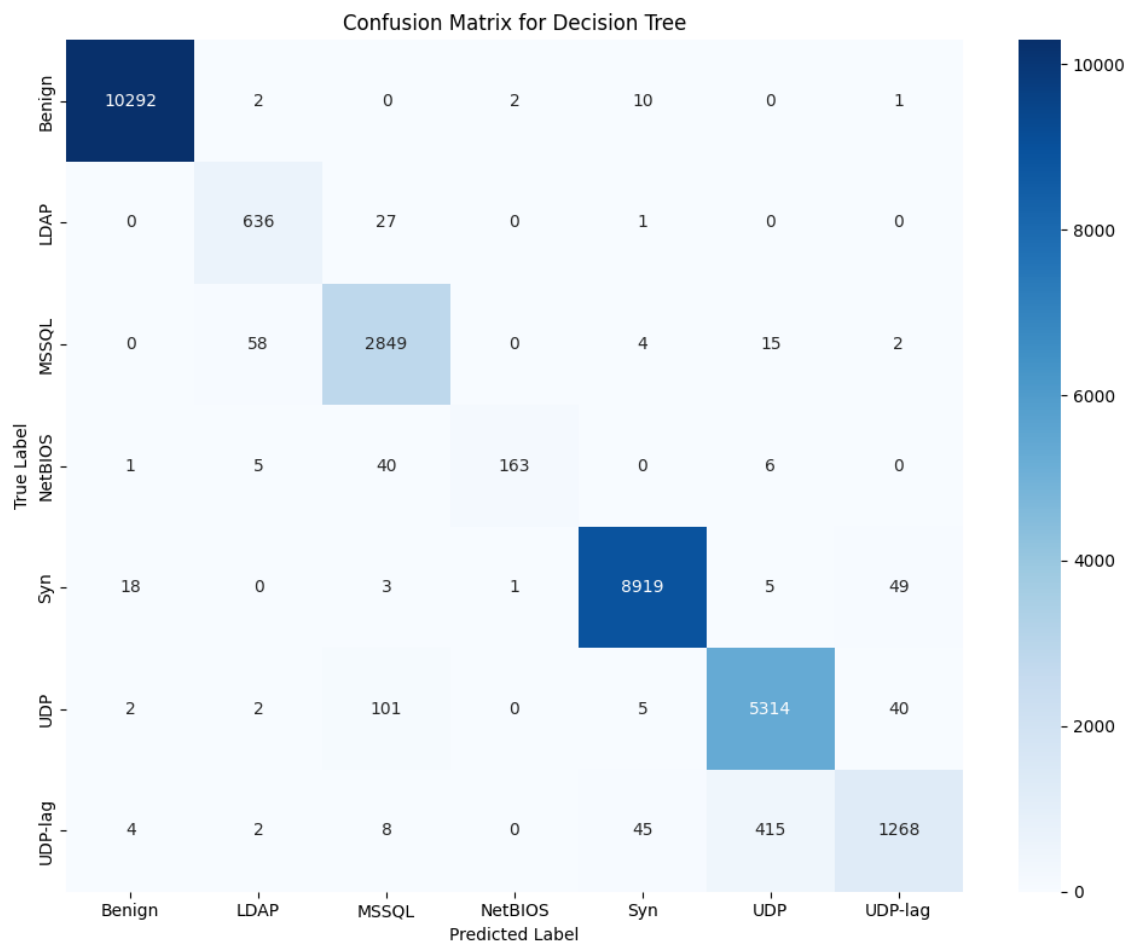


Figure B.7: Decision tree confusion matrix (CICDDoS-2019)

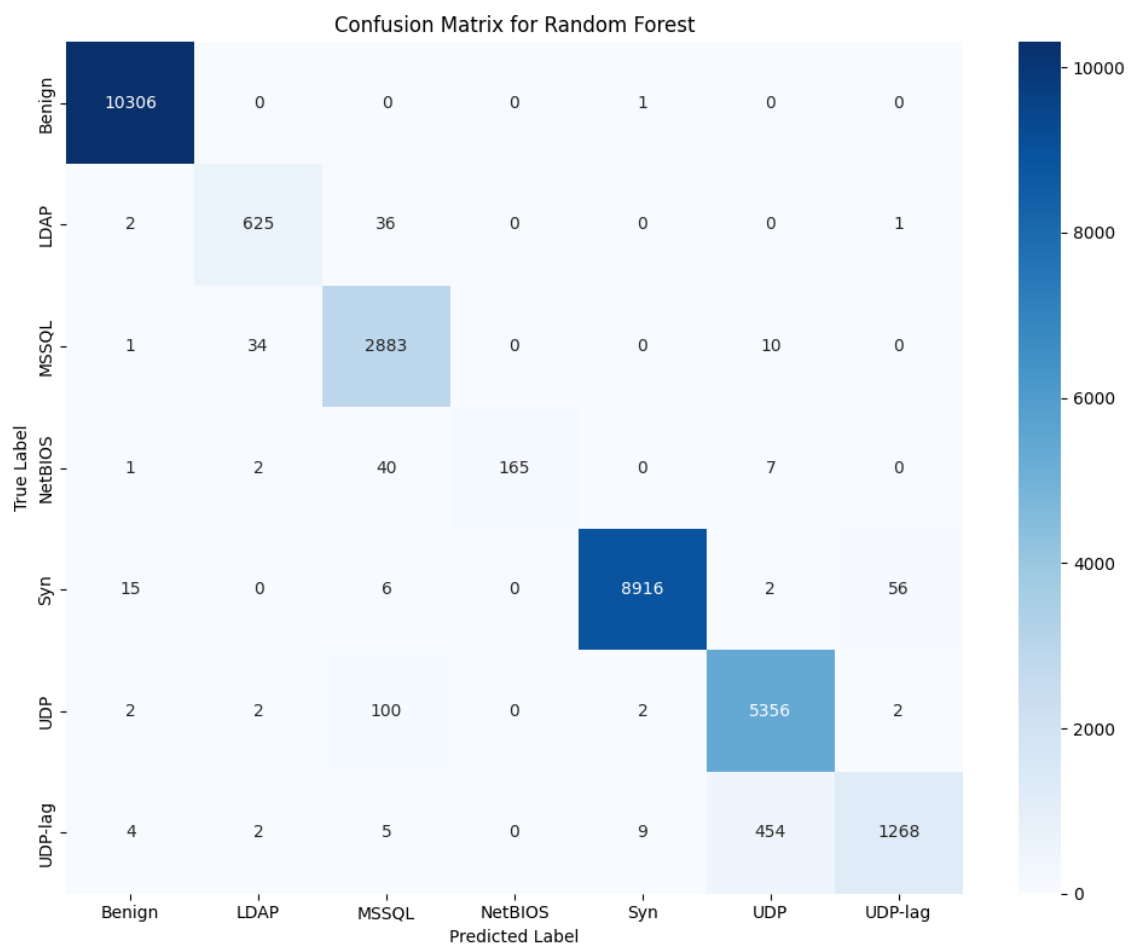


Figure B.8: Random forest confusion matrix (CICDDoS-2019)

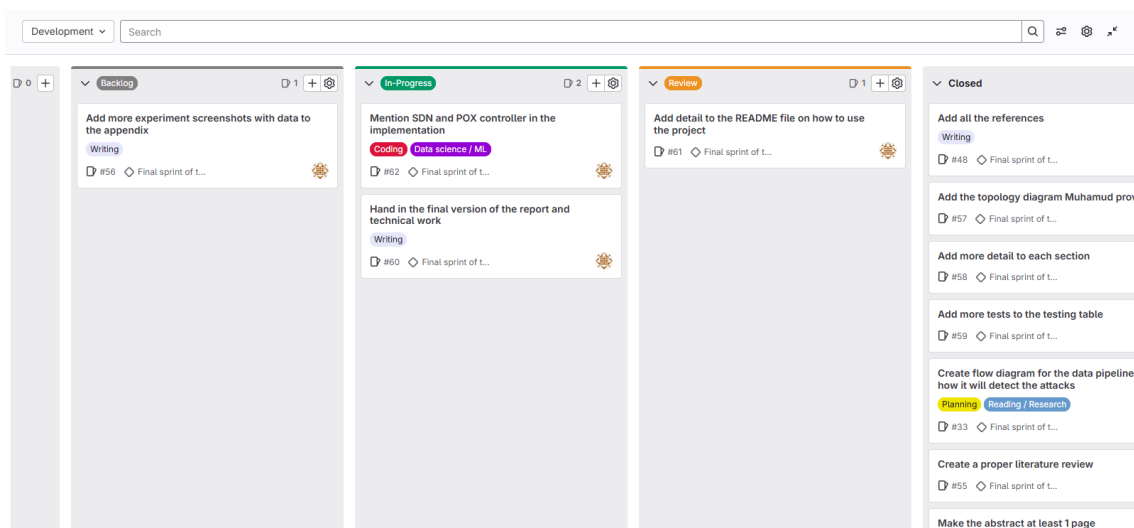


Figure B.9: GitLab Kanban dashboard