THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

Novel Attack Detection in IoT Network Intrusion Detection Systems

by

Chiao-Hsi Joshua Wang

Submitted for the degree of Master of Data Science at
The University of Queensland in 2024

School of Electrical Engineering and Computer Science

Executive Summary

Internet of Things (IoT) refers to physical devices that can wirelessly connect to the internet, encompassing everything from smartphones to household appliances. As of 2021, there were over 11.7 billion IoT devices globally (Duggal, 2023). While this rapid proliferation has transformed various industries, it has also introduced significant cybersecurity challenges. Between 2022 and 2023, malicious attacks on IoT devices increased by 400% worldwide. Although solutions are continuously developed to mitigate new cybersecurity risks, cybercriminals persist in finding and exploiting unprotected vulnerabilities (Knowles, 2023). As IoT devices become increasingly connected and complex, the need to identify and mitigate novel cyber threats has become critical.

This project aims to address the limitations of current state-of-the-art IoT network intrusion detection systems (IDS), which often struggle to adapt to the emergence of new types of cyberattacks designed to bypass existing defences (Lail, Garcia, & Olivo, 2023). To enhance detection, I developed a decision model that leverages the predictions of three sub-models — two neural networks and a k-Nearest Neighbours (KNN) algorithm — to classify IoT traffic as benign, a known attack, or a novel attack. The first neural network is a multi-class classifier to determine whether a traffic packet of data is benign or one of several known attack types. The second is a binary classifier trained to distinguish between benign traffic and any form of attack. Lastly, the KNN model classifies packets based on their nearest neighbours, while also providing a measure of similarity to known attacks by comparing differences between traffic. By combining these sub-models, the decision model classifies packets through majority voting and uses distance metrics to identify novel attacks that deviate significantly from known patterns. To train the model to differentiate between different traffic types, I use the UQ IoT IDS dataset, which contains samples of benign traffic and 9 different types of attacks (He et al., 2022).

The resulting model achieved high accuracy and F1 scores, comparable to existing IDS solutions, with the added advantage of detecting novel cyberattacks. However, a notable challenge was the similarity between certain IoT attack packets, which occasionally led to the misclassification of novel attacks as known ones. Future improvements could focus on enhancing the model's capability to differentiate between similar attack types, and including adaptive learning mechanisms for the model to learn from newly identified novel attacks. The findings of this research provide valuable insights for both academic research and industrial applications in cybersecurity, contributing to a more secure and resilient IoT infrastructure.

Contents

List of Figures

List of Tables

1 Introduction

Internet of Things (IoT) encompasses a diverse range of devices, from smartphones to household appliances, that connect wirelessly to the internet. With over 11.7 billion IoT devices in use globally as of 2021 (Duggal, 2023), these technologies have transformed daily life, enabling remote control of home appliances, real-time health monitoring, and more. This vast interconnectivity brings unprecedented convenience, efficiency, and connectivity. However, the rapid adoption of IoT devices has also introduced significant cybersecurity risks. Between 2022 and 2023, malicious attacks on IoT networks surged by 400% (Knowles, 2023). Successful attacks can lead to data breaches, privacy violations, and critical service disruptions, impacting individuals and organisations alike, especially in sensitive sectors such as healthcare and defence.

Network Intrusion Detection Systems (NIDS) are essential for safeguarding IoT networks by identifying and mitigating potential security threats. However, the evolving nature of IoT ecosystems poses unique challenges for traditional NIDS. These systems are typically effective at detecting known attack types but may struggle to identify novel threats (Lail et al., 2023). As cybercriminals continually innovate to bypass existing security mechanisms, there is a critical need for adaptive intrusion detection models capable of recognising previously unseen cyberattacks in IoT environments.

This project addresses this gap by proposing a novel approach that integrates a binary classifier, a multi-class classifier (both neural networks), and a K-Nearest Neighbours (KNN) model to detect emerging threats in IoT networks. The binary and multi-class classifiers are designed to identify known attack types, while the KNN model measures distance differences to aid in recognising new anomalous patterns that could indicate novel threats. This multi-model approach leverages the strengths of each component to enhance detection accuracy against evolving threats.

In this report, I will outline the motivations, goals, and challenges driving this research, and review current methods used in the field of IoT security. I will then describe the data sources and methodologies applied in this project, followed by an analysis of the results. Finally, I will conclude with a discussion of the project's limitations and potential future directions for improving upon this work.

2 Project Background

In this section, I will delve into the background of the project, starting with the methodologies of existing solutions and highlighting the aspects of these solutions that motivated this project. Finally, I will provide an overview of the objectives that this project aimed to achieve.

2.1 Existing Solutions

In an IoT network, various devices are interconnected, with the ability to send and receive data among themselves. To enable this communication, data is transferred in "traffic packets", which store information such as the sender and receiver's addresses, packet length, and the actual data being transmitted (Cloudflare, n.d.-b). Cyber-criminals can exploit IoT networks by intercepting packets during communication or sending malicious packets to devices, leading to disruptions and other harmful consequences (Cloudflare, n.d.-b).

To mitigate these risks, Network Intrusion Detection Systems (NIDS) are employed. NIDS monitor traffic packets sent to and from IoT devices and can take action when a suspicious packet is detected, such as blocking it or alerting an administrator (Paloalto Networks, n.d.). NIDS use various approaches for detecting suspicious traffic, which can be categorised into traditional pattern-based systems and machine learning methods (Helixstorm, 2022). Machine learning approaches can be further divided into supervised and unsupervised algorithms, differing in how the models are trained to recognise attacks (Lail et al., 2023). Below, I explore each of these NIDS types in detail.

2.1.1 Pattern-Based Detection

Signature-based and anomaly-based detection are two primary methods for monitoring IoT network traffic using pattern recognition. Both approaches compare new traffic against expected traffic behaviour to identify deviations that may indicate an attack. However, they differ in how they identify attack packets.

Signature-based detection involves comparing traffic data against a database of known attack patterns and flagging traffic as malicious when a match is found (Helixstorm, 2022). This method is relatively straightforward but comes with limitations, such as requiring frequent manual updates to the database to remain effective. This reliance on pre-known patterns means it cannot defend against novel attacks not yet catalogued (Paloalto Networks, n.d.).

Anomaly-based detection, on the other hand, does not need a database of known attack patterns, which gives it an advantage over signature-based detection (Helixstorm, 2022). It works by flagging any packet that deviates from established "normal" traffic patterns as potentially

malicious. While this allows for the detection of new, previously unknown attacks, it requires a baseline understanding of what constitutes normal traffic. Since traffic patterns can vary significantly across different networks, this method needs an initial training period to learn the normal patterns. Additionally, anomaly-based methods may still misclassify malicious packets that mimic normal traffic patterns (Paloalto Networks, n.d.). To address these challenges, machine learning algorithms can be utilised, as explored in the following sections.

2.1.2 Supervised Machine Learning Algorithms

Machine learning (ML) algorithms, capable of learning from existing data and making decisions, have increasingly been implemented in NIDS to enhance cyber-attack detection. ML models offer advantages over traditional pattern-based detection by recognising complex patterns in large datasets. Research shows that ML models can significantly improve the accuracy and efficiency of threat detection (Dong & Wang, 2016). These models can be categorised into supervised and unsupervised types, which differ in how they learn patterns.

Supervised machine learning models require pre-existing labelled data, where each data point is associated with a known outcome. These models learn to recognise patterns between input data and their corresponding labels, allowing them to make predictions on new data. In NIDS, supervised ML models can be trained using labelled traffic packets, marked as either benign or a particular type of attack.

Various supervised ML algorithms, such as random forests, support vector machines, Gaussian Naive Bayes, and logistic regression, have been explored in NIDS research. Studies have shown that models like random forests can achieve high accuracy rates of up to 97% (Lail et al., 2023). However, simpler models, such as Gaussian Naive Bayes and logistic regression, have been found to perform less effectively (Lail et al., 2023). To enhance the performance of supervised ML models, more complex architectures can be considered.

One such example is the multilayer perceptron (MLP). MLPs are artificial neural networks, consisting of various layers of interconnected nodes. The model starts with a single input layer, followed by one or more hidden layers, and concludes with an output layer to generate predictions. An example of a MLP is illustrated in Figure 1. In a basic MLP, each node in a layer is connected to every node in the following layer, with each connection assigned a weight. During training, the MLP adjusts these weights to minimise the difference between predicted and actual outputs, enabling it to learn complex functions and make accurate predictions on new data. An MLP with four hidden layers and 100 hidden units (the number of nodes per layer) has been shown to achieve average accuracies of 99% for intrusion detection tasks (Kim, Shin, Jo, & Kim, 2017).

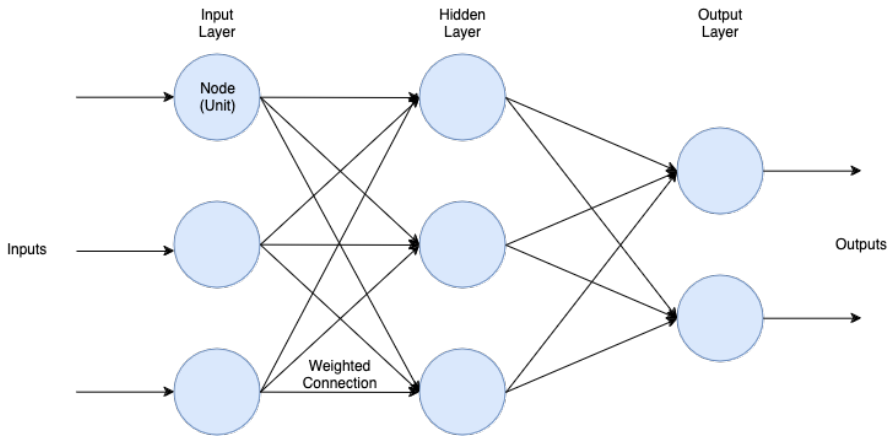


Figure 1: Illustration of a Multilayer Perceptron

2.1.3 Unsupervised Machine Learning Algorithms

Unlike supervised methods, unsupervised machine learning algorithms do not require labelled data to learn patterns for attack detection. One effective method explored in research is the autoencoder, illustrated in Figure 2. The autoencoder consists of two main components: an encoder and a decoder. These components are composed of layers similar to those found in MLPs.

The encoder compresses input data into a latent space representation, which summarises the input, while the decoder reconstructs the original input from the latent space. To use autoencoders for identifying malicious traffic, the model is trained only on benign traffic, enabling it to learn to reconstruct normal traffic accurately. When an anomalous packet is presented as input, it is expected that the model will struggle to reconstruct it, resulting in a higher reconstruction error, which indicates a potential attack.

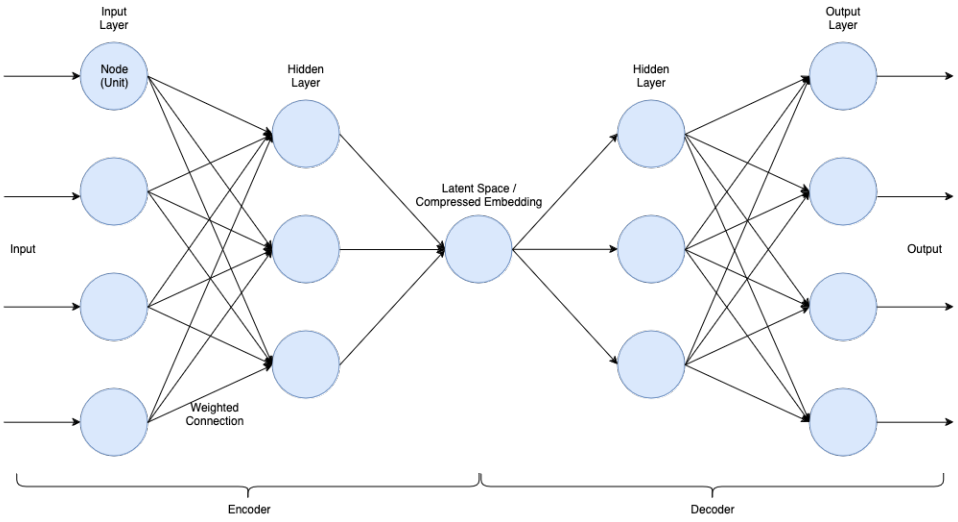


Figure 2: Illustration of an Autoencoder

This concept is a fundamental part of the state-of-the-art Kitsune model for NIDS, which employs multiple autoencoders trained to reconstruct benign traffic samples only. When Kitsune reconstructs data with significant error, it signals an anomaly in IoT traffic (Mirsky, Doitshman, Elovici, & Shabtai, 2018). This approach requires only benign traffic for training, simplifying the process compared to obtaining labelled data for all potential traffic types for training a supervised machine learning model.

2.2 Project Motivations

Despite extensive research in IoT Intrusion Detection Systems (IDS), many existing methods still face challenges that limit their overall effectiveness. The widespread use of IoT devices and the increasing frequency of cyber-attacks contribute to a growing variety of data that Network Intrusion Detection Systems (NIDS) must analyse and classify. This trend is expected to continue as IoT adoption grows and attack methods become more sophisticated (Knowles, 2023). Additionally, as detection techniques improve, attackers adapt by discovering new vulnerabilities and developing advanced attack methods. This underscores the need for NIDS that can adapt to evolving attack strategies over time.

Traditional NIDS techniques, such as signature-based detection, rely on known databases for comparison, which limits their effectiveness in detecting new and more complex attack patterns. Machine learning-based techniques have also become increasingly popular due to their efficiency and accuracy; however, they also present certain limitations (Dong & Wang, 2016).

While machine learning models for NIDS offer an advantage over methods that rely solely on known patterns, simpler machine learning models often suffer from low accuracy. In some cases, these simpler models can yield low performance metrics, despite more complex models in the same category being able to achieve better classification accuracy (Lail et al., 2023). Therefore, more advanced deep learning models, such as MLPs, are often needed to improve detection results when dealing with more complex data.

Although MLPs have shown promise in IoT intrusion detection research, they share a common drawback with other machine learning algorithms: the need for supervised training (Kim et al., 2017). Supervised learning depends on manually labelled data, and this requirement hinders the ability of these models to adapt to new data in real-time, as it is impractical to label incoming data in a continuous traffic stream. Given the dynamic nature of IoT environments, supervised training approaches can quickly become outdated and struggle to detect new types of attacks effectively as well. The increasing complexity of data and the demand for adaptive solutions make supervised methods less suitable for maintaining high detection accuracy in IoT networks, which often face diverse and novel attack types.

The Kitsune model, which has been previously discussed, takes a different approach by using autoencoders for anomaly detection based on reconstruction errors (Mirsky et al., 2018). Kitsune's unsupervised learning approach allows it to adapt continuously and requires only examples of benign traffic to operate. However, a major limitation is that it assumes malicious traffic will always be harder to reconstruct, which may not always hold true. With advancements in areas like adversarial machine learning, future cyber-attacks could be designed to mimic normal traffic patterns, deceiving models like Kitsune. Moreover, Kitsune is limited to anomaly detection and cannot classify the specific type of attack it identifies.

These limitations in current solutions motivated the development of a model capable of identifying specific attack types in network traffic while also being capable of detecting novel attacks. To achieve this, specific objectives were established, which are detailed in the next section.

2.3 Project Objectives

To address the limitations of current NIDS models, this project aims to develop a machine learning model with the following key objectives:

- Binary Classification: Accurately classify IoT network traffic as either an attack or benign.
- Multi-Class Classification: Classify IoT network traffic as benign or identify its specific type of attack.
- Novel Attack Detection: Effectively recognise and classify previously unseen types of attacks.

Achieving these objectives will allow the proposed model to incorporate the strengths of existing solutions while addressing their weaknesses. Before delving into the architectural details and components used to fulfil these objectives, some preliminary knowledge required for the rest of the report is outlined in the next section.

3 Project Preliminaries

In this section, we will cover some fundamental concepts that will be used throughout the report. This includes an overview of neural networks and their application in classification tasks, K-Nearest Neighbours (KNN) models, and evaluation metrics like accuracy and the F1 score.

3.1 Neural Networks

Neural networks are a type of machine learning model that consist of layers of interconnected nodes that can learn to recognise patterns in data through a training process (IBM, n.d.-a). A specific example of a neural network used in this project is the Multi-Layer Perceptron (MLP).

Each layer in a neural network performs a transformation on the input data. The first layer is the input layer, which receives the raw data features. This is followed by one or more hidden layers that extract and learn features from the input using weighted connections. Finally, there is an output layer, which makes the final predictions (IBM, n.d.-a).

Neural networks are supervised machine learning algorithms, meaning they depend on pre-labelled data. Using this pre-labelled data, neural networks can be trained to perform predictions. The training process aims to minimise the difference between the network's predictions and the actual labels, as measured by a loss function. This involves adjusting the weights of the connections between neurons using an optimisation algorithm, which updates the weights to decrease the loss function. Neural networks can be used for various classification tasks, including binary classification (e.g., determining whether a packet of traffic is an attack or not) and multi-class classification (e.g., identifying the type of an IoT threat) (IBM, n.d.-a).

One major advantage of neural networks is their ability to learn complex, non-linear relationships in data. However, they require a significant amount of data and computational resources. This project will utilise two neural networks as components of the proposed model, in addition to a K-Nearest Neighbours model, which is described in the next section.

3.2 K-Nearest Neighbours

K-Nearest Neighbours (KNN) is a machine learning algorithm that does not explicitly learn a decision function during training but rather memorises the training instances (IBM, n.d.-b). The KNN algorithm identifies the k closest training examples to a given input data point based on a distance metric, such as the Euclidean distance that is used in this project. The Euclidean distance takes two feature vectors x and y, and for each of the x_i and y_i components of the two feature vectors, calculates the following:

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

In classification tasks, using the Euclidean distance, the algorithm will classify the inputted data as the class that was most common among its k nearest neighbours. The choice of k greatly influences the model's performance: a small k value means each data point only considers a small number of its closest neighbours which can lead to predictions being affected by noise, while a large k value may smooth out the decision boundary too much, making it less flexible (IBM, n.d.-b).

KNN is a non-parametric model, where it does not learn the underlying distribution of data the model is being trained on, making it well-suited for problems where the data distribution is unknown or complex. This characteristic is particularly useful in problems such as anomaly detection, where outlier data points can be identified based on their distance from other points (IBM, n.d.-b).

3.3 Metrics

Evaluation metrics are crucial for assessing the performance of machine learning models. They provide insights into how well the model is making predictions, and different metrics are suitable for different kinds of tasks. Since this project focuses on how well a model can classify traffic packets, accuracy and F1 score will be used as the primary metrics to measure the model's ability to correctly differentiate between different traffic types.

3.3.1 Accuracy

The accuracy metric is the ratio of correctly predicted instances to the total number of data points being considered. Mathematically, it is expressed as:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

While accuracy is easy to understand and interpret, it can be misleading when dealing with imbalanced datasets. For example, if a model is trained to detect rare network attacks, a high accuracy might be achieved by always predicting the more common "normal" traffic. In such cases, other metrics like the F1 score can be more informative (Evidently AI, n.d.).

3.3.2 F1 Score

The F1 score is a more comprehensive metric that accounts for both precision and recall. It is particularly useful for imbalanced datasets, which is relevant when exploring the UQ IoT IDS dataset used in this project. In the context of traffic packet classification, precision refers to the proportion of correctly identified packets of a certain traffic type among all packets predicted as that type by the model. Recall, on the other hand, measures the proportion of packets of that traffic type that the model correctly identifies out of all actual instances of that type (Kundu, n.d.). The F1 score is the harmonic mean of precision and recall, given by:

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

The harmonic mean emphasises low values more than the arithmetic mean, ensuring that the F1 score is high only when both precision and recall are high. This makes the F1 score a balanced metric that provides a method of gauging a model's performance on imbalanced data (Kundu, n.d.).

Additionally, since this project uses a diverse dataset of 10 different possible types of traffic (including benign and 9 attack types), a separate F1 score will be calculated for each traffic class. To evaluate the overall performance of the model, the individual F1 scores for each class can be averaged to obtain a single value. However, various strategies can be employed for this averaging, depending on the importance of each class (Kundu, n.d.).

One method is the weighted F1 score, where the F1 score for the ith class is multiplied by the proportion of entire dataset that the class takes up (w_i). This is formulated as:

$$\text{Weighted F1 Score} = \sum_{i=1}^{n} w_i \cdot \text{F1 Score}_i$$

The benefit of the weighted F1 score is that it reflects the model's performance according to the class distribution in the dataset. If the dataset is representative of a real-world scenario, stronger performance on more frequently represented classes is often more important than classes which occur less commonly (Kundu, n.d.).

However, for a NIDS, all traffic types should be considered as equally important, regardless of their frequency in the dataset. This is because we aim to create a NIDS that is effective at detecting all types of traffic, rather than just the most common types. Therefore, instead of weighted F1 scores, the macro F1 score will be used for this project. This is formulated as:

$$\text{Macro F1 Score} = \frac{1}{n}\sum_{i=1}^{n} \text{F1 Score}_i$$

This approach calculates the F1 score for each class independently and then averages them, ensuring that every possible traffic type contributes equally to the final performance measure. To understand why the F1 score is required and how the model will learn to differentiate between traffic packets, we will explore the dataset for the project in the next section.

4 Dataset

In this section, we provide an in-depth overview of the dataset used in this project, along with the rationale and methods for sampling, feature extraction, and preprocessing. We discuss the characteristics of the data and explain the necessary steps taken to prepare it for model training and evaluation.

4.1 Dataset Overview

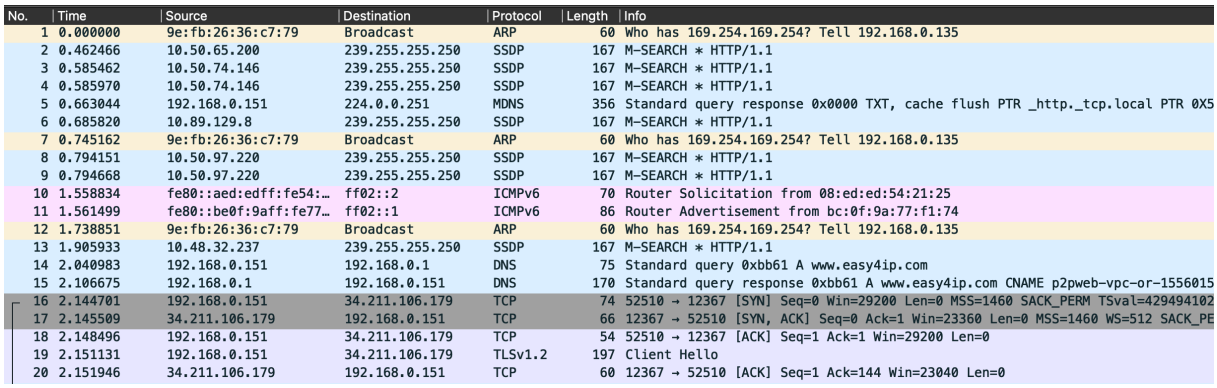
This project leverages the UQ IoT IDS dataset to help the model understand how to classify types of traffic packets (He et al., 2022). The dataset contains a total of 41,404,983 packets, each labelled as either benign or belonging to one of nine attack types. A more detailed explanation of each attack type is available in the Appendix (see Table 11).

To ensure the model's robustness across diverse conditions, the traffic packets in this dataset were collected from eight different IoT devices, capturing variations of the same attacks across devices. This approach helps to provide a more comprehensive view of how attacks can manifest in real-world scenarios, and enhances the model's ability to generalise. Table 1 presents a summary of the traffic types across these devices, detailing the distribution of benign packets and packets corresponding to each attack type.

Traffic Type	Camera	Google Nest Mini	Lenovo Bulb	Raspberry Pi	Smart Clock	Smartphone 1	Smartphone 2	Smart TV	All devices	Traffic Totals
ACK Flooding	1,355,787	1,023,660	187,623	1,324,669	897,760	1,220,486	1,022,158	1,877,857	-	**8,910,000**
ARP Spoofing	621	4,275	759	220	2,659	159	4,275	2,334	-	**15,302**
Port Scanning	4,213	4,223	4,401	4,050	4,389	4,206	10,189	5,953	-	**41,624**
Service Detection	7,533	19,233	5,002	4,949	28,922	4,455	14,865	7,685	-	**92,644**
SYN Flooding	797,217	965,978	187,820	661,368	598,337	1,265,998	347,909	696,542	-	**5,521,169**
UDP Flooding	1,719,733	1,270,097	297,774	460,805	561,931	1,478,419	1,627,320	1,145,485	-	**8,561,564**
Telnet Brute Force	N/A	N/A	N/A	128,653	N/A	N/A	N/A	N/A	-	**128,653**
HTTP Flooding	N/A	N/A	N/A	683,408	N/A	N/A	N/A	N/A	-	**683,408**
Host Discovery	-	-	-	-	-	-	-	-	25,775	**25,775**
Benign Traffic	-	-	-	-	-	-	-	-	17,422,556	**17,422,556**
Device Totals	3,885,104	3,287,466	683,379	3,268,541	2,093,998	3,973,723	3,026,716	3,735,856	17,450,200	**41,404,983**

Table 1: Traffic Samples in the UQ IoT IDS Dataset

To illustrate the structure of the packets, Figure 3 presents a sample of 20 ARP spoofing packets collected from a camera device. Each packet is represented by seven fields: number, time, source, destination, protocol, length, and information. This structure is consistent across all traffic types. Table 2 provides definitions for each of these fields.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	9e:fb:26:36:c7:79	Broadcast	ARP	60	Who has 169.254.169.254? Tell 192.168.0.135
2	0.462466	10.50.65.200	239.255.255.250	SSDP	167	M-SEARCH * HTTP/1.1
3	0.585462	10.50.74.146	239.255.255.250	SSDP	167	M-SEARCH * HTTP/1.1
4	0.585970	10.50.74.146	239.255.255.250	SSDP	167	M-SEARCH * HTTP/1.1
5	0.663044	192.168.0.151	224.0.0.251	MDNS	356	Standard query response 0x0000 TXT, cache flush PTR _http._tcp.local PTR 0X5
6	0.685820	10.89.129.8	239.255.255.250	SSDP	167	M-SEARCH * HTTP/1.1
7	0.745162	9e:fb:26:36:c7:79	Broadcast	ARP	60	Who has 169.254.169.254? Tell 192.168.0.135
8	0.794151	10.50.97.220	239.255.255.250	SSDP	167	M-SEARCH * HTTP/1.1
9	0.794668	10.50.97.220	239.255.255.250	SSDP	167	M-SEARCH * HTTP/1.1
10	1.558834	fe80::aed:edff:fe54:…	ff02::2	ICMPv6	70	Router Solicitation from 08:ed:ed:54:21:25
11	1.561499	fe80::be0f:9aff:fe77…	ff02::1	ICMPv6	86	Router Advertisement from bc:0f:9a:77:f1:74
12	1.738851	9e:fb:26:36:c7:79	Broadcast	ARP	60	Who has 169.254.169.254? Tell 192.168.0.135
13	1.905933	10.48.32.237	239.255.255.250	SSDP	167	M-SEARCH * HTTP/1.1
14	2.040983	192.168.0.151	192.168.0.1	DNS	75	Standard query 0xbb61 A www.easy4ip.com
15	2.106675	192.168.0.1	192.168.0.151	DNS	170	Standard query response 0xbb61 A www.easy4ip.com CNAME p2pweb-vpc-or-1556015
16	2.144701	192.168.0.151	34.211.106.179	TCP	74	52510 → 12367 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=429494102
17	2.145509	34.211.106.179	192.168.0.151	TCP	66	12367 → 52510 [SYN, ACK] Seq=0 Ack=1 Win=23360 Len=0 MSS=1460 WS=512 SACK_PE
18	2.148496	192.168.0.151	34.211.106.179	TCP	54	52510 → 12367 [ACK] Seq=1 Ack=1 Win=29200 Len=0
19	2.151131	192.168.0.151	34.211.106.179	TLSv1.2	197	Client Hello
20	2.151946	34.211.106.179	192.168.0.151	TCP	60	12367 → 52510 [ACK] Seq=1 Ack=144 Win=23040 Len=0

Figure 3: Example of Traffic Packet Data in UQ IoT IDS Dataset

Field	**Data Type**	**Description**
No.	Numerical (Integer)	Order in which packets were transmitted
Time	Numerical (Float)	Time when the packet is sent
Source	Categorical (String)	Device where the packet originated from
Destination	Categorical (String)	Device the packet is being sent to
Protocol	Categorical (String)	How the packet is being sent
Length	Numerical (Integer)	Size of the packet
Info	Categorical (String)	Information contained in the packet

Table 2: Description of Packet Fields in the UQ IoT IDS Dataset

As seen above, the seven features representing each raw packet provides a limited amount of information for the model to make decisions. With the dataset structure established, we now move to examine the specific data processing steps to prepare the data for model training, starting with feature extraction.

4.2 Feature Extraction

Feature extraction is a crucial step in data preprocessing, as the fields in the raw packet data listed in Table 2 alone may not provide enough contextual information for accurate model predictions. For instance, the raw packet data does not account for the timing between packets sent to or from specific devices. This omission is significant, as certain attacks, like ACK, UDP, or SYN flooding, rely on overloading a target device with numerous non-essential packets, thereby disrupting its ability to process important traffic. Without additional features, the model would lack insight into the temporal patterns between packets, which is essential for identifying these types of flooding attacks.

To address these limitations, we used the Kitsune feature extractor, which is a part of the state-of-the-art Kitsune model for intrusion detection. This extractor generates summary statistics that capture the flow of traffic to and from specific source and destination devices over multiple time frames. It processes raw packet data (e.g., as shown in Figure 3) and considers previous packets within time windows of 100 milliseconds, 500 milliseconds, 1.5 seconds, 10 seconds, and 1 minute, to compute 100 key statistics for each packet (Mirsky et al., 2018). By using these enriched features as inputs to the model, it gains valuable context regarding prior packet flows, allowing it to assess whether a current packet is part of a pattern indicative of an attack.

In addition to the summary statistics provided by the Kitsune feature extractor, certain aspects of the original packet information were still valuable for traffic classification. Specifically, the length and protocol of each raw traffic packet were retained as additional features in the dataset. Since packet length is an integer, it was directly usable without further modification. However, the protocol field required transformation, because it is represented as a categorical string. Machine learning models operate on numerical data, so we needed to encode the possible protocols into numerical values.

A straightforward sequential encoding (e.g., assigning 1, 2, 3, etc.) was inappropriate because it would imply an ordinal relationship between protocols that does not exist (e.g., suggesting that a higher protocol number is "greater" than a lower one). To avoid this, we applied one-hot encoding, creating a separate feature for each protocol type. Each packet was then represented by a binary vector where the feature corresponding to its protocol was set to 1, while all others remained 0. This encoding preserved the categorical nature of the protocols, allowing the model to interpret each protocol independently (Geeks for Geeks, 2024).

Overall, the dataset preparation involved a careful selection of both engineered and original features to enable effective traffic classification. Each packet is ultimately represented by a total of 299 features (100 Kitsune features, 198 one-hot encoded protocol features, 1 length feature), as well as a single label indicating the type of traffic the packet belonged to (0 for benign, 1 to 9 for attack traffic types). By leveraging Kitsune's feature extractor and supplementing it with essential raw packet information, it was possible to ensure that the model had access to both temporal patterns and packet-specific attributes necessary for distinguishing between benign and malicious traffic accurately.

4.3 Data Sampling

With the necessary features for the model to clearly differentiate between packets now extracted, we move onto sampling the data. By observing Table 1, it is evident that the dataset contains more examples of specific types of traffic compared to others. For example, the number of

flooding attacks (ACK Flooding, SYN Flooding, UDP Flooding and HTTP Flooding) greatly outnumber the number of samples present for other types of attacks. Similarly, there is a total of 17,422,556 samples of benign traffic, which greatly outnumbers the available data for any of the attacks. This discrepancy is more clearly visualised in Figure 4.

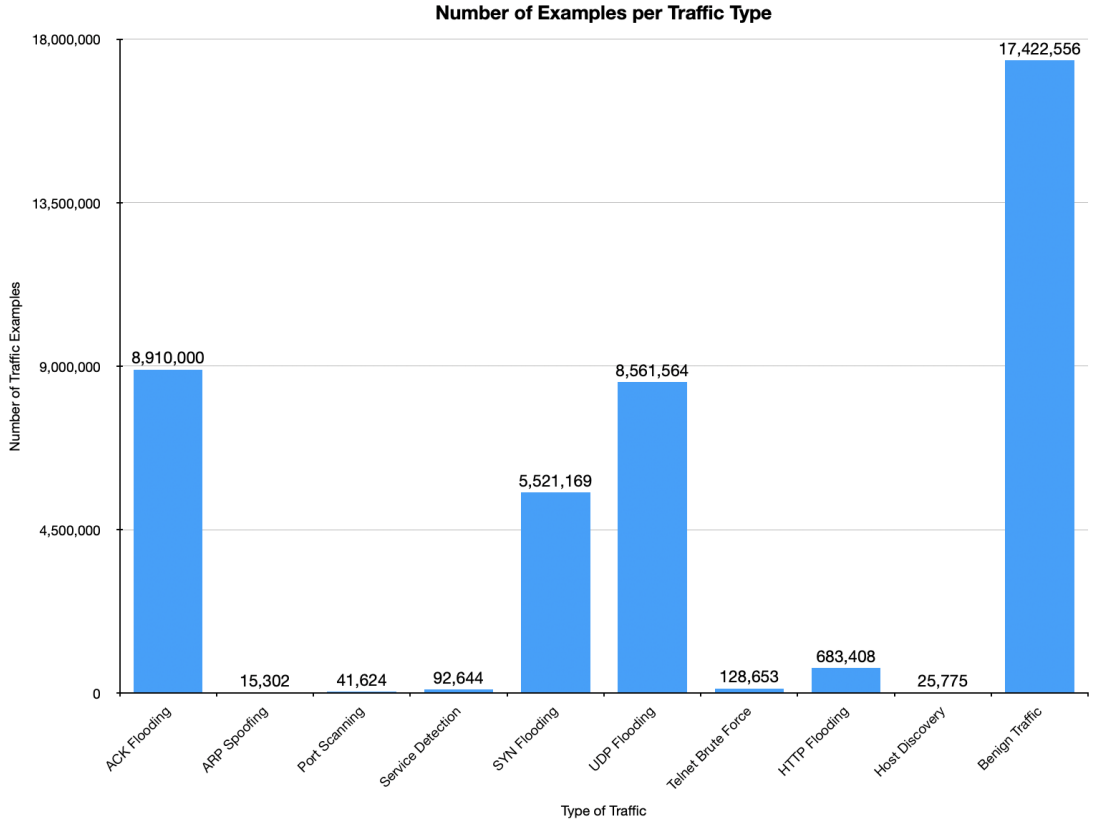


Figure 4: Number of Samples per Traffic Type in the UQ IoT IDS Dataset

Since the number of examples for each traffic type differ greatly, there is a class imbalance problem. If a model were to be trained on this dataset with imbalanced classes, it could detrimentally affect the performance of the model to generalise and detect all other types of attacks. Specifically, the model could achieve high overall accuracy by performing well on the majority classes — those with a larger number of samples — while neglecting the minority classes, which are less represented. This could lead to a situation where the model's performance on the minority classes is poor, yet the overall accuracy remains artificially high due to the abundance of samples from the majority classes (Santiago, 2023). Thus, addressing the class imbalance is critical for ensuring robust model performance across all traffic types.

Moreover, the dataset must be split into separate training, validation and testing sets, ensuring no overlap of data points between these subsets. This separation is crucial to accurately evaluate the model's performance, as it allows the model to be trained on a subset of the available data while being tested on unseen data. Doing this will prevent data leakage, ensuring that the

evaluation metrics reflect the model's true ability to perform prediction in a practical setting (Brownlee, 2020).

To solve the class imbalance problem, a strategy of oversampling the minority classes and undersampling the majority classes was used (Santiago, 2023). For the minority classes, all available data was used, dividing it into training, validation, and testing sets in the following proportions: 70% for training, 15% for validation, and 15% for testing. In contrast, for the majority classes, given the vast number of packets, I randomly sampled 5% of the available data for each of the training, validation, and testing sets whilst ensuring no repetitions. Furthermore, sampling was evenly distributed across the eight devices present in the dataset. By randomly allocating traffic packets to each of the three training, validation and testing sets, it was also possible to mitigate the potential for the model to learn to classify packets through their order of collection.

After performing the sampling procedures, the number of samples within the training set is summarised in Table 3, while the validation and testing set counts can be found in Table 4. Observing the total counts for each traffic type, we can find that the subsets are now considerably more balanced. By splitting the data into three separate subsets with no repeated samples between the subsets, we now also have a way to reliably train the model and evaluate its prediction performance.

Traffic Type	Camera	Google Nest Mini	Lenovo Bulb	Raspberry Pi	Smart Clock	Smartphone 1	Smartphone 2	Smart TV	All devices	Traffic Totals
ACK Flooding	67789	51183	9381	66233	44888	61024	51108	93893	0	**445499**
ARP Spoofing	435	2992	531	154	1861	111	2992	1634	0	**10710**
Port Scanning	2949	2956	3081	2835	3072	2944	7132	4167	0	**29136**
Service Detection	5273	13463	3501	3464	20245	3118	10406	5380	0	**64850**
SYN Flooding	39861	48299	9391	33068	29917	63300	17395	34827	0	**276058**
UDP Flooding	85987	63505	14889	23040	28097	73921	81366	57274	0	**428079**
Telnet Brute Force	0	0	0	90057	0	0	0	0	0	**90057**
HTTP Flooding	0	0	0	34170	0	0	0	0	0	**34170**
Host Discovery	0	0	0	0	0	0	0	0	18042	**18042**
Benign Traffic	0	0	0	0	0	0	0	0	871108	**871108**
Device Totals	202294	182398	40774	253021	128080	204418	170399	197175	889150	**2267709**

Table 3: Sample Counts in the Training Set

Traffic Type	Camera	Google Nest Mini	Lenovo Bulb	Raspberry Pi	Smart Clock	Smartphone 1	Smartphone 2	Smart TV	All devices	Traffic Totals
ACK Flooding	67789	51183	9381	66233	44888	61024	51108	93893	0	**445499**
ARP Spoofing	93	641	114	33	399	24	641	350	0	**2295**
Port Scanning	632	633	660	608	658	631	1528	893	0	**6243**
Service Detection	1130	2885	750	742	4338	668	2230	1153	0	**13896**
SYN Flooding	39861	48299	9391	33068	29917	63300	17395	34827	0	**276058**
UDP Flooding	85987	63505	14889	23040	28097	73921	81366	57274	0	**428079**
Telnet Brute Force	0	0	0	19298	0	0	0	0	0	**19298**
HTTP Flooding	0	0	0	34170	0	0	0	0	0	**34170**
Host Discovery	0	0	0	0	0	0	0	0	3866	**3866**
Benign Traffic	0	0	0	0	0	0	0	0	871108	**871108**
Device Totals	195492	167146	35185	177192	108297	199568	154268	188390	874974	**2100512**

Table 4: Sample Counts in the Validation and Testing Set

4.4 Data Preprocessing

With essential features extracted from the raw packet data and balanced train, validation, and test splits established, data preprocessing was necessary to prepare the dataset for effective model training. These preprocessing steps ensured that features were appropriately scaled and formatted for the model.

The first step was min-max scaling on the numerical Kitsune features, transforming each feature to a range between 0 and 1. Standardising feature values in this way ensures each feature contributes equally to the model's learning process, preventing any particular feature's scale from inadvertently dominating the model (Bhandari, 2024).

To achieve this, the minimum and maximum values for each feature were derived from the training data alone. To scale each feature in the training set, the formula used was:

$$\frac{X - x_{min}}{x_{max} - x_{min}}$$

where X represents the value being transformed, x_{min} is the minimum value in the train set, and x_{max} is the maximum value in the train set. The validation and testing sets were scaled in a similar fashion, using x_{min} and x_{max} from the training set to prevent data leakage, thereby ensuring that unknown data points remained consistent with the scale seen during training.

Following scaling, Principal Component Analysis (PCA) was applied to reduce dataset dimensionality (number of features). PCA transforms the original features into a set of uncorrelated principal components, capturing the main variance in the data while discarding less informative dimensions (Parte, 2020). This approach is particularly valuable since each traffic packet has 299 extracted features after my data preprocessing steps, which increases computational complexity and could introduce noise into the model.

Additionally, the k-nearest neighbours model used in the project methodology relies on feature distances. This makes dimensionality reduction even more crucial to mitigate issues arising from the "curse of dimensionality". This is a phenomenon where, as data becomes more high-dimensional, models struggle to identify meaningful patterns due to increased noise and distance between data points (Parte, 2020).

As with min-max scaling, PCA was first fitted to the training data, then applied consistently to the validation and test data to avoid leakage. The PCA model was specified to retain at least 95% of the variance in the dataset, and this condensed the 299 features for each traffic packet to just 12 features representing each packet.

Together, these preprocessing steps — min-max scaling and PCA — prepared and optimised

the dataset for model training. However, instead of retaining all classes in each of the training, validation and testing datasets to train and evaluate the model, the datasets had to be prepared in a way to simulate novel attacks to help achieve a key objective of this project.

4.5 Simulation of Novel Attacks

A primary objective of this project is to develop a model capable of identifying novel attacks that it was not specifically trained on. To effectively evaluate the model's ability to classify unknown threats, the dataset must be processed in a way that simulates this scenario.

To achieve this, a strategy of systematically excluding one attack type from the training dataset during the model training phase was used. For each attack type, the specific attack was excluded from both the training and validation sets before any preprocessing steps (min-max scaling and PCA) were applied. This approach allowed the model to be trained and validated using only known traffic packets, comprising benign instances and eight types of previously encountered attacks.

However, the omitted "novel" attack type was retained in the testing dataset. This design enables us to evaluate the model's performance in detecting attacks it was not trained on. Importantly, leaving out the novel class before the preprocessing steps ensures that the presence of the novel attack does not influence the scaling parameters or the PCA transformation derived from the "known" attacks. By maintaining the integrity of the training process, we can accurately assess how well the model generalises to new, unseen threats.

With the dataset now sampled and preprocessed, we can begin training the model. The model architecture used to achieve the objectives for the project will be detailed in the following section.

5 Methodology

This section outlines the model architecture and the process of fine-tuning hyperparameters to optimise performance of the model. We describe the components of the model, explain how they are designed to meet the project's goals, and discuss the strategies used for training and evaluation.

5.1 Model Architecture

The model architecture was developed using three distinct sub-model components, each serving a specific purpose to address the primary goals of the project. Two of these sub-components are neural networks, chosen for their proven efficacy in NIDS classification tasks (Kim et al., 2017). Additionally, the third component was a KNN model was utilised for its strength in anomaly detection (IBM, n.d.-b). All three sub-model components are supervised machine learning algorithms. The reason for using supervised training processes is because an objective for the project is that the proposed model has the ability to predict the exact type of traffic detected, and this is only possible using supervised training. The following subsections will describe the role of each sub-model, the architectural details, and the manner in which they work together to classify network traffic packets as benign, a known attack, or a novel attack.

5.1.1 Binary Classification Sub-Model

The binary classification sub-model is a neural network specifically designed to ascertain whether a given network traffic packet is benign or represents either a known or novel attack. This classification is critical, as one of the primary objectives of the project is to establish a reliable mechanism for distinguishing between benign and malicious traffic.

The architecture of the binary classifier is illustrated in Figure 5. The model accepts a feature vector of 12 elements, which encapsulates the information extracted from each traffic packet during the preprocessing stage.

The structure of this neural network follows the principles of a Multi-Layer Perceptron (MLP) and consists of four fully connected layers, with 512, 128, 64, and 16 nodes in each layer, respectively. Every node in one layer is connected to all nodes in the subsequent layer, with weights assigned to these connections.

To enhance the model's learning capabilities, the ReLU (Rectified Linear Unit) activation function is applied after each hidden layer. This choice of activation function facilitates the introduction of non-linearity into the model, allowing it to learn complex patterns within the data (Whitfield, 2024). The output layer contains a single node with a sigmoid activation function

to produce a probability score, indicating the likelihood that a given packet is an attack. Predictions closer to 0 indicate benign traffic, while values closer to 1 indicate an attack. Since the original labels from the data range from 0 (benign) to 8 (different attack types), all attack labels are consolidated to a single label of 1 to simplify the binary classification.

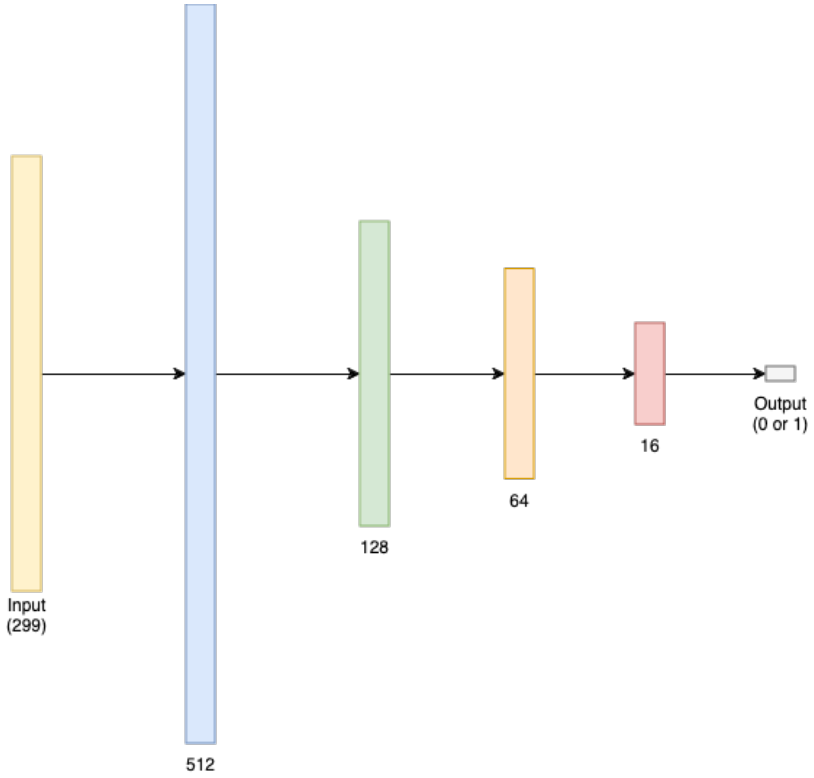


Figure 5: Binary Classifier Sub-Model Architecture

During training of the binary classifier, the Binary Cross-Entropy (BCE) with Logits Loss function was used to quantify the error between the predicted and actual labels, and is formulated as:

$$\text{BCE with Logits Loss} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\sigma(\hat{y}_i)) + (1-y_i)\log(1-\sigma(\hat{y}_i))]$$

where N is the number of samples, y_i is the true label for the ith sample (either 0 or 1), $\hat{y}_i$ is the predicted raw score (logit) for the ith sample by the classifier, and $\sigma(\hat{y}_i)$ is the sigmoid function applied to $\hat{y}_i$ and scales the value to be between 0 and 1 (Saxena, 2024). Minimising this loss during training, the model learns to adjust its weights to improve its ability to detect IoT cyber-attacks.

By employing this binary classification sub-model, the overall framework gains the ability to effectively identify benign traffic versus malicious traffic. This helps to achieve the first of the

three project objectives.

5.1.2 Multi-Class Classification Sub-Model

The multi-class classification sub-model is designed to differentiate between specific types of traffic packets, classifying them as benign or one of several known attack types. This sub-model employs a neural network architecture similar to the binary classifier but with a key difference: the output layer.

As illustrated in Figure 6, the model architecture consists of four fully connected layers with 512, 128, 64, and 16 nodes, respectively. Similar to the binary classifier, each node in one layer is fully connected to every node in the next, with associated weights that are adjusted during training. However, the output layer of the multi-class classifier has as many nodes as there are known traffic classes in the dataset, with one node corresponding to each specific class, ensuring that the model can make predictions across all possible categories.

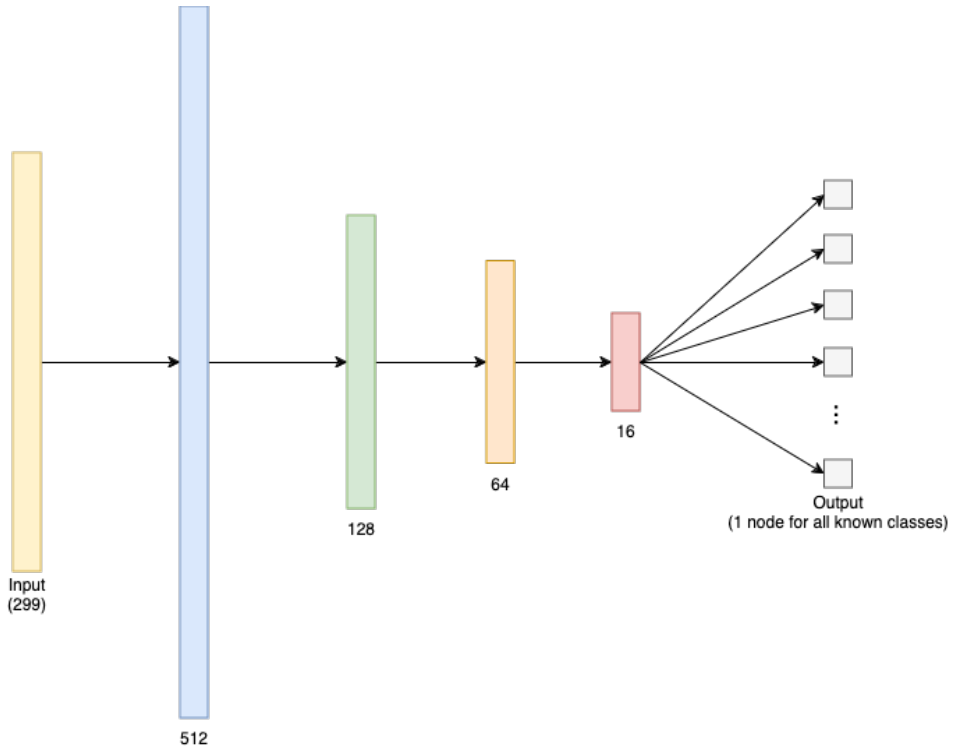


Figure 6: Multi-Class Classifier Sub-Model Architecture

To train the model, the 12 features of each traffic packet from the data preprocessing steps are passed through the network, and the output layer produces a probability distribution for all classes. The model assigns the class with the highest probability as the predicted label for a given packet.

The loss function used for this sub-model is Cross-Entropy Loss, which is well-suited for multi-class classification tasks, and is formulated as:

$$\text{Cross-Entropy Loss} = -\sum_{i=1}^{C} y_i \log(\hat{p}_i)$$

where C is the number of classes, y_i is the true label (1 for the correct class, 0 otherwise), and $\hat{p}_i$ is the predicted probability for class i. Cross-Entropy Loss calculates the difference between the predicted probabilities for each possible label and the true class labels from the dataset, encouraging the model to maximise the probabilities of the correct class (Pykes, 2024). By minimising this loss during training, the model learns to correctly assign probabilities that reflect the likelihood of each class, improving overall classification accuracy, and achieving another objective of the project.

5.1.3 K-Nearest Neighbours (KNN) Sub-Model

The K-Nearest Neighbours (KNN) sub-model was incorporated to evaluate the similarity between a given network traffic packet and known packets, serving to identify novel attacks and classify packets based on proximity to known classes. The use of KNN in this project provides an intuitive, distance-based similarity measure that complements the aforementioned neural network sub-models.

For this project, the model calculates the distance between a given packet and all other packets in the training set using the Euclidean distance metric, as detailed in the Project Preliminary section. The packet is then classified based on the majority class among its k closest neighbours.

To determine the optimal value of k (number of closest neighbours to consider), different KNN models with different k hyperparameters were fitted to the training set. Each of the models were then evaluated against the validation set to understand how the model performed when considering different numbers of closest neighbours. These results are plotted in Figure 7. After testing various values of k, it was found that a k value of 3 provided the best balance for model performance. Thus, k was set to 3 for all predictions, ensuring that the model considers a diverse yet manageable set of neighbours when making classification decisions.

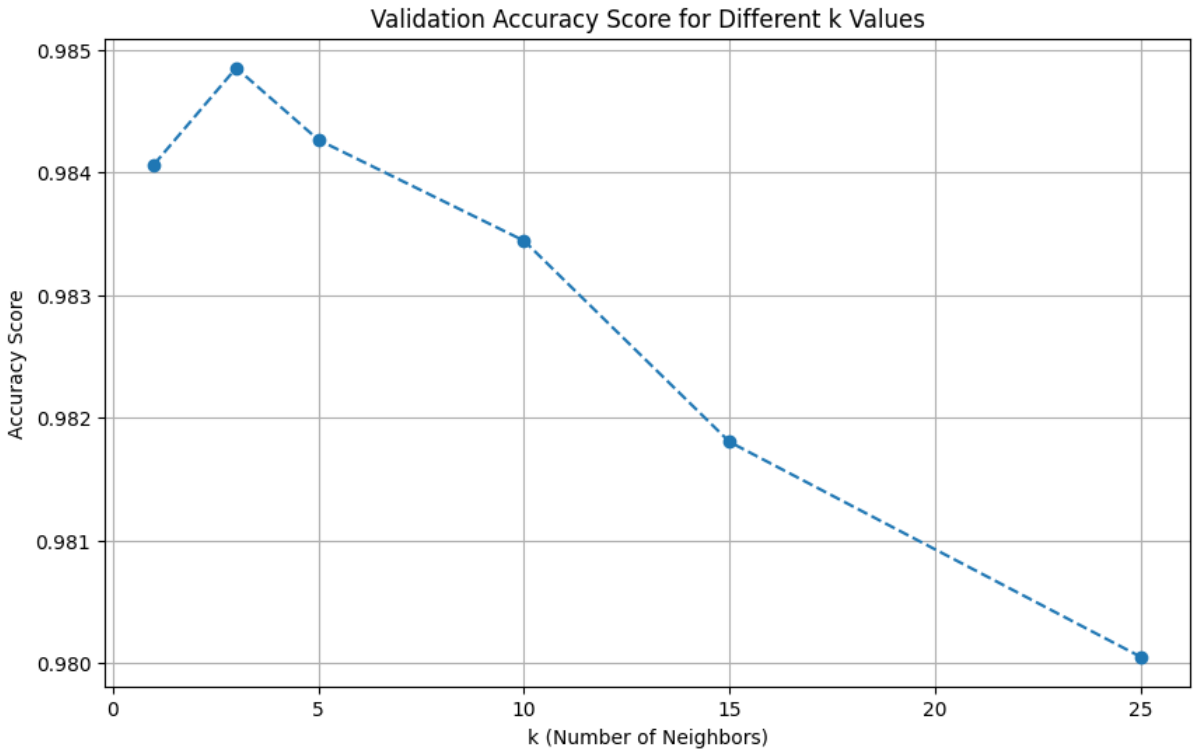


Figure 7: Accuracy Score of KNN Model with Various k Hyperparameters

Additionally, the KNN model provides a method of identifying whether a traffic packet is a novel attack through the use of the distance metric. This is achieved by calculating the average distance between all data points of the same class with their neighbours. Then, for new traffic packets to be classified by the model, the KNN model will compute the average distance between the packet and its k closest neighbours. If the distance between the new data point and its predicted class is significantly greater than the average distance of known data points in that class, the packet is flagged as potentially novel. This suggests that the new packet may represent a novel attack type, as it deviates notably from all other known traffic types.

By integrating KNN into the overall model architecture, the system gains an additional layer of interpretability, allowing it to make decisions based on the direct similarity of packets to known data points. This is particularly beneficial in situations where understanding how closely a packet resembles known classes is crucial for accurate classification and anomaly detection.

5.1.4 Voting Ensemble

Using the three sub-models capable of addressing the primary objectives of the project, they are combined into an ensemble to classify traffic packets as benign, a known attack, or a novel attack. This ensemble approach aggregates the predictions from each sub-model, leveraging their outputs to determine the final classification, as illustrated in Figure 8.

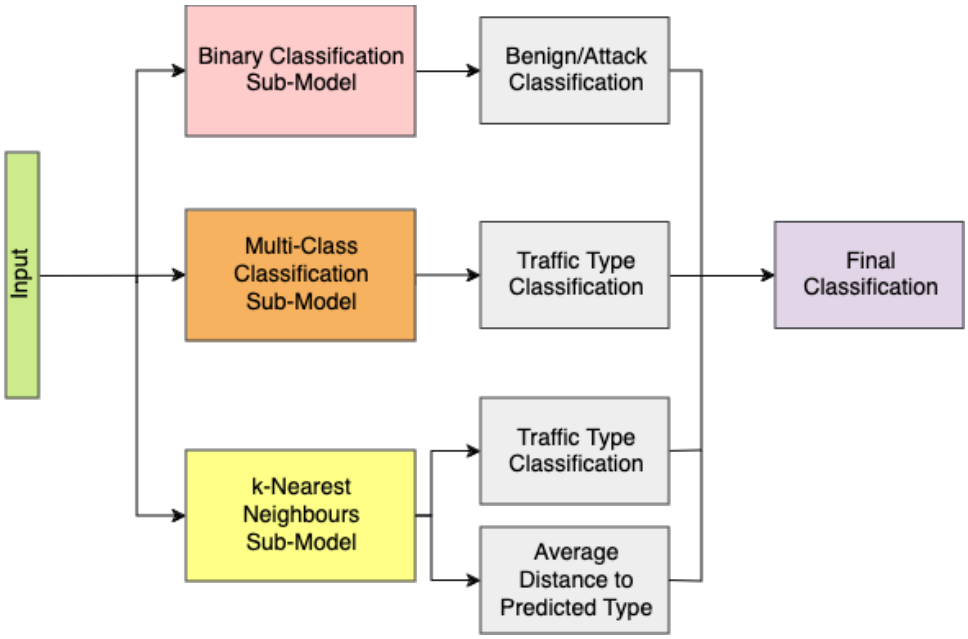


Figure 8: Full Model Architecture

In the voting ensemble, the input features of a packet are passed to all three sub-models simultaneously. The binary and multi-class classifiers, trained independently, make their predictions: the binary classifier indicates if the packet is benign or an attack, while the multi-class classifier predicts if it matches any known traffic type (benign or attack). Since these two neural networks cannot identify novel traffic, the KNN sub-model is incorporated to support tie-breaking and provide the average distance to the predicted class as a metric for novel attack detection.

The final classification decision is based on a voting algorithm outlined in Algorithm 1. When a majority of the sub-models agree that a packet is benign, the ensemble predicts it as such. If the models disagree, the algorithm evaluates whether the packet is a known or novel attack. Specifically, in line 8, the `avg_knn_distance` (the average distance between the packet and its 3 nearest neighbours) is compared to the `avg_known_class_distance` (the average distance between points in the predicted class). Additionally, my model incorporates a distance threshold, which can be adjusted using the hyperparameter α, allowing for controlled flexibility in what is accepted as a "known" attack. If the `avg_knn_distance` exceeds the threshold, the packet is classified as a novel attack. Otherwise, it is labelled as a known attack.

By using this algorithm, the model is able to effectively combine the predictions of all three sub-models. Then, using these predictions, it has the ability to identify between benign traffic, known attacks, and novel attacks.

Algorithm 1: Labelling Data Based on Sub-Model Predictions

Input: Average distance for KNN-predicted traffic type `avg_known_class_distance`, Distance threshold hyperparameter α

```
for each instance i to predict do
    Retrieve multi_label, binary_label, and knn_label;
    if (multi_label = 0 and binary_label = 0 and knn_label = 0) or
    (multi_label = 0 and binary_label = 0 and knn_label > 0) or
    (multi_label > 0 and binary_label = 0 and knn_label = 0) then
        Predict as benign;
    else
        if avg_knn_distance > avg_known_class_distance × α then
            Predict as novel attack;
        else
            Predict as knn_label (known attack);
        end
    end
end
return final_labels;
```

5.2 Distance Threshold Hyperparameter Tuning

As discussed in the previous section and in Algorithm 1, my model includes a distance threshold hyperparameter that can be adjusted. This threshold is applied as a multiplier to the average distance between data points within a given class. Higher α values allow more variability, enabling traffic packets to be classified as known even if they are somewhat dissimilar to existing data points. Conversely, smaller α values impose stricter constraints, requiring new data points to have a distance to its neighbours similar to the average class distance to be considered part of that class.

To choose the ideal hyperparameter for the model, we can evaluate the model's performance using a range of different α values, and use the value that yields the strongest results across all cases. To do this, we consider each of the "unknown" attack combinations (where each possible attack is left out of the training set but still kept in the validation set). The entire model is trained to recognise all but the simulated "novel" attack, before being evaluated on the validation set to gauge the model's ability to identify novel attacks from known attacks. Since the project aims to develop a model that yields the strongest performance across all possible traffic types, the α value that gives the highest F1 score across all known classes and the novel class is chosen.

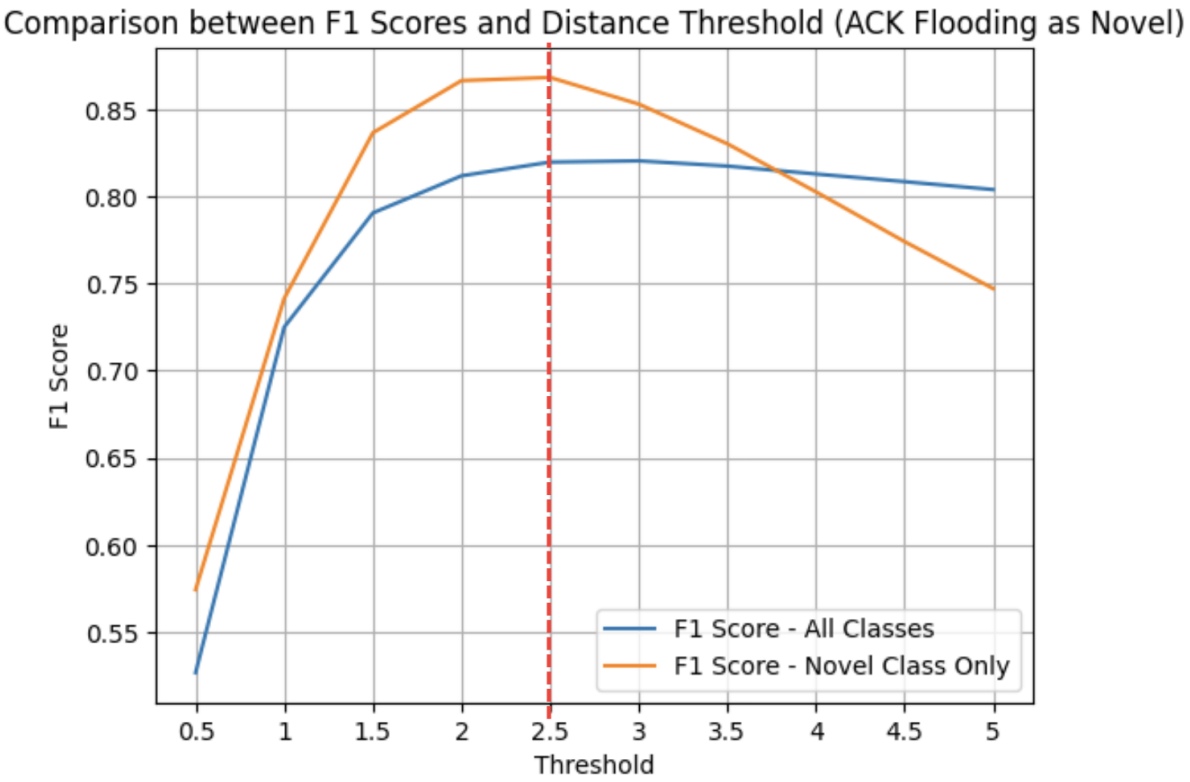


Figure 9: F1 Scores Across Different Threshold Values (ACK Flooding as Novel)

Figure 9 demonstrates how F1 scores fluctuate with different α values, using ACK Flooding as the example novel attack type. The figure shows F1 scores for both known and novel class detection, providing insight into how the threshold impacts overall model performance. In this case, an α value of 2.5 was found to be ideal, optimising performance on both known and novel classes. The choice of 2.5 is logical since a factor of 2 covers the complete range of distances within the known class, while an additional 0.5 factor accommodates slight deviations that still align with known class characteristics. Notably, this trend and the optimal α value of 2.5 were consistent across all tested novel attack scenarios.

With this tuning complete, the decision model was prepared for further evaluation. The following section discusses the performance results achieved using this refined model.

6 Analysis of Results

Using the testing dataset sampled during the data preprocessing step of the project, we judge the model's performance and whether it is able to accomplish the three goals set out for the project. To do this, each data point in the testing set is passed to the model, and each sub-model makes their respective predictions and distance calculations. Then, the decision algorithm is used to classify these data points.

Since multiple versions of the dataset were created to have separate versions with different "novel" attacks, the model needs to be retrained on each version to ensure it does not have prior exposure to the "novel" attack beforehand during training. Similarly, evaluation of the model's results needs to be performed across all versions of the testing, where each attack type is considered "novel" once.

Analysing the results of the models, we find that some traffic types were easily distinguishable from others, whilst some others could be easily confused with other attacks. Observing Table 5 below, we can see the performance of the model across all traffic types, with the ACK Flooding attack highlighted in red being considered the novel attack. From this, it is evident that the model performed well with high F1 scores across the majority of traffic types. Additionally, it was also able to identify the novel ACK Flooding attack quite well, showcasing that the model was successful at detecting attacks that it had never seen before. However, attacks such as ARP Spoofing and Host Discovery had relatively low F1 scores due to the model confusing these packets with other types of attack traffic.

Traffic Type	**Precision**	**Recall**	**F1**
Benign	0.99	0.98	0.99
ACK Flooding	0.88	0.85	0.87
ARP Spoofing	0.49	0.53	0.51
Port Scanning	0.88	0.82	0.85
Service Detection	0.91	0.78	0.84
SYN Flooding	0.81	0.94	0.87
UDP Flooding	1.00	0.95	0.98
HTTP Flooding	1.00	0.94	0.97
Telnet Brute Force	0.83	0.77	0.80
Host Discovery	0.71	0.42	0.53

Table 5: Results on Testing Set with ACK Flooding as Novel

A similar pattern in results can be seen below in Table 6, which considers UDP Flooding as the novel attack. The majority of classification F1 results are quite high, but performance on traffic such as ARP Spoofing and Host Discovery are still quite poor and these same attacks are still being mistaken for other existing types of attacks. Additionally, comparing Table 5 to Table 6, it can also be observed that there are slight changes in performance when specific attacks are chosen as novel. For example, ARP Spoofing experiences a drop of 7% in F1 score from the set with ACK Flooding as novel to the set with UDP Flooding as novel. This is likely a byproduct of the features being shifted during the data pre-processing steps. When a specific type of attack is simulated as novel, its values are no longer considered during min-max scaling and PCA, which can affect other data points' values after preprocessing.

Traffic Type	**Precision**	**Recall**	**F1**
Benign	0.98	0.98	0.98
ACK Flooding	0.98	0.92	0.95
ARP Spoofing	0.55	0.37	0.44
Port Scanning	0.89	0.82	0.85
Service Detection	0.91	0.77	0.83
SYN Flooding	0.97	0.91	0.94
UDP Flooding	0.89	0.99	0.93
HTTP Flooding	1.00	0.94	0.97
Telnet Brute Force	0.80	0.76	0.78
Host Discovery	0.72	0.40	0.51

Table 6: Results on Testing Set with UDP Flooding as Novel

To further delve into the results of the model, we now consider ARP Spoofing as the novel attack to understand if the model can identify this traffic type as novel even if it struggles to classify it after being directly trained on it. As can be seen in Tables 5 and 6, the model struggled to identify ARP Spoofing packets even when it was not a "novel" attack, and struggled to separate the packets from other traffic types. The results of testing with ARP Spoofing as "novel" are displayed in Table 7. With a F1 score of 0.02, this is an extremely poor performance, especially when compared to the model's previously strong novel attack detection capabilities on attacks such as ACK and UDP Flooding.

Traffic Type	Precision	Recall	F1
Benign	0.99	0.98	0.99
ACK Flooding	0.98	0.93	0.96
ARP Spoofing	0.01	0.28	0.02
Port Scanning	0.83	0.83	0.83
Service Detection	0.89	0.79	0.84
SYN Flooding	0.98	0.92	0.95
UDP Flooding	1.00	0.96	0.98
HTTP Flooding	1.00	0.94	0.97
Telnet Brute Force	0.82	0.77	0.79
Host Discovery	0.69	0.44	0.53

Table 7: Results on Testing Set with ARP Spoofing as Novel

The Port Scanning attack is another case of an attack that the model struggles to identify accurately as novel, and Table 8 illustrates this. For this traffic type, the model is able to perform well when it is directly trained on it, which can be seen in Tables 5, 6 and 7. However, the model performs very poorly at detecting the ”novel” Port Scanning attack at a F1 of 0.02, and confuses it with the ”known” Service Detection attacks. This is likely due to the attacks being carried out using similar patterns, and so when the traffic is ”novel” but seems similar enough to a ”known” attack, the model classifies it as ”known”.

Traffic Type	Precision	Recall	F1
Benign	0.99	0.98	0.99
ACK Flooding	0.98	0.93	0.96
ARP Spoofing	0.52	0.45	0.48
Port Scanning	0.01	0.12	0.02
Service Detection	0.70	0.83	0.76
SYN Flooding	0.98	0.92	0.95
UDP Flooding	1.00	0.96	0.98
HTTP Flooding	1.00	0.94	0.97
Telnet Brute Force	0.83	0.77	0.80
Host Discovery	0.71	0.44	0.55

Table 8: Results on Testing Set with Port Scanning as Novel

However, despite the model confusing certain attack types with other attacks, the results from this project are still very strong. The model is capable of producing predictions to classify traffic

packets as benign, known attacks or novel attacks, and for the majority of traffic packets this is performed at a very high accuracy. Additionally, for the attack packets that do get confused with each other, this may not be a significant drawback as long as the packets are still classified as a type of attack so that the model is able to protect IoT networks from attacks in general.

Although the model achieves strong standalone results, these results should also be compared against existing solutions to evaluate whether the model offers any accuracy advantages. To do this, the model's performance in two areas is evaluated: multi-class classification, and anomalous traffic detection. The same UQ IoT IDS dataset that has been used throughout this project is used to directly compare results between different models.

For multi-class classification, the proposed model is compared to a random forest and convolutional neural network to understand how well it performs at differentiating between traffic packets compared to existing solutions. The results are listed below in Table 9. From these results, we can see that my proposed model achieves an accuracy of 95.01% and F1 score of 79.65%. This outperforms the convolutional neural network significantly, but underperforms the random forest which achieved an accuracy of 98.66% and F1 score of 88.30%. This can be explained by the random forest having the ability to make more fine-grained decisions using packets' features. But, despite my model performing worse than the random forest, it still has the added ability of being able to detect novel attacks which the random forest cannot do.

Model	**Accuracy**	**F1**
Proposed Model	95.01%	79.65%
Random Forest	98.86%	88.30%
Convolutional Neural Network	89.25%	64.87%

Table 9: Comparison of Results for Multi-Class Classification

To evaluate the model's ability to perform anomaly detection, it will be compared against the state-of-the-art Kitsune model. The reason why the Kitsune model has to be compared separately rather than comparing it directly with random forests and convolutional neural networks is that, during training, Kitsune is only exposed to benign traffic packets. This means it can detect attack traffic of any kind but cannot perform multi-class classification and specify the exact class of attack that was detected. Therefore, the predictions from my proposed model are also transformed to be 0 (benign) or 1 (attack of any kind) to match the functionality of Kitsune.

Model	Accuracy	F1
Proposed Model	98.83%	98.79%
Kitsune	94.99%	92.21%

Table 10: Comparison of Results for Anomaly Detection

Comparing the results between my model and Kitsune at detecting attacks from Table 10, we can see that my model outperforms Kitsune in both accuracy and F1. This means that my model is able to identify attacks more accurately than the state-of-the-art Kitsune model. Additionally, my model also has the added advantage of specifying the exact type of attack and when a new attack type is detected.

With these results, the three objectives of this project have been met as I have successfully developed a model capable of identifying the exact type of traffic packet that is encountered, whilst also being able to be applied to previously unknown attacks. Additionally, the performance of my model is also quite strong. However, there are still aspects of the project that can be improved, and this will be detailed in the following section of the report.

7 Recommendations

Although this project has achieved strong results, there are still areas for improvements that can further enhance the model's performance. This section of the report will identify some of the potential areas that future work can explore.

One major area for improvement is feature engineering. As observed in the results section of the report, some types of attacks are easily confused with others. A likely cause of this is that the features obtained through the Kitsune feature extractor and the addition of the packet length and protocol is not enough for the model to clearly differentiate between some types of traffic packets. To solve this issue, further efforts can be made towards identifying other features that can be selected so that different attack traffic packets are not confused with one another.

Another significant enhancement to the proposed model is incorporating a mechanism that allows the model to adapt and learn from new data points and novel classes. This can be achieved by firstly performing clustering on detected novel attacks to remove noise and ensure detected attacks are truly novel, before fine-tuning each of the sub-model components. To perform fine-tuning, the two neural network based models would need to be trained on a mixture of novel and known attacks to ensure they learn to classify novel attacks whilst maintaining the ability to classify known attacks. The KNN model would also need to be updated with the new novel attacks, signifying that these attacks are now "known" while still allowing the detection of future novel attacks. Since all three components of my model are supervised machine learning algorithms and are dependent on labelled data, the model's predictions would be utilised as the labels for fine-tuning. A difficulty that may arise when implementing this feature is that neural networks require a substantial amount of data, and the model will also need to be exposed to a significant amount of novel attack data to be able to accurately learn from its own predictions.

Furthermore, evaluating the model across additional datasets is another avenue worth exploring. This project only utilised the UQ IoT IDS dataset, which limits our understanding of the model's generalisation capabilities. By testing the proposed model across a variety of IoT datasets, we can assess its ability to generalise, an essential feature given the rapidly evolving nature of IoT networks in practical applications.

In summary, enhancing feature engineering, implementing adaptive learning mechanisms, and expanding the evaluation to include diverse datasets are recommendations that could significantly improve the model's performance and applicability in real-world settings. Nonetheless, even without these additional features, this project has been a notable success in achieving its primary objectives.

8 Conclusion

In conclusion, this project enhances existing cybersecurity frameworks for IoT ecosystems by developing a decision model aimed at improving IoT network intrusion detection systems (IDS). The model specifically addresses the challenge of identifying both known and novel cyberattacks. By integrating the strengths of multi-class classification, binary classification, and k-Nearest Neighbours (KNN), the proposed model achieved high accuracy and F1 scores, comparable to state-of-the-art solutions, while also demonstrating the capability to detect previously unseen cyber threats. This represents a significant advancement in overcoming one of the major limitations of current IDS approaches.

Despite these successes, challenges remain, particularly with regard to attack packets that share similar features, which occasionally led to misclassifications. These challenges highlight the inherent complexity of IoT environments, where diverse device types and varying data traffic patterns complicate intrusion detection. Future work should focus on refining the model's ability to differentiate between closely related attack types. Specifically, further exploration of feature engineering processes to identify distinctive characteristics of traffic packets could substantially enhance performance. Additionally, incorporating adaptive learning mechanisms that enable the model to retrain itself in real-time as new attacks are identified would further increase its utility. These advancements would support the development of more dynamic and responsive cybersecurity solutions, capable of evolving alongside emerging threats.

References

Bhandari, A. (2024, October 9). *Feature Scaling: Engineering, Normalization, and Standardization.* https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/.

Brownlee, J. (2020, August 14). *What is the Difference Between Test and Validation Datasets?* https://machinelearningmastery.com/difference-test-validation-datasets/.

Cloudflare. (n.d.-a). *What is a DDoS attack?* https://www.cloudflare.com/learning/ddos/what-is-an-ack-flood/.

Cloudflare. (n.d.-b). *What is a packet?* https://www.cloudflare.com/learning/network-layer/what-is-a-packet/.

Dong, B., & Wang, X. (2016). Comparison deep learning method to traditional methods using for network intrusion detection. In *2016 8th IEEE International Conference on Communication Software and Networks (ICCSN).* Retrieved from https://doi.org/10.1109/iccsn.2016.7586590 doi: 10.1109/iccsn.2016.7586590

Duggal, N. (2023, December 5). *What are IOT devices? Definition, types, and 5 most popular ones for 2024.* https://www.simplilearn.com/iot-devices-article. (Simplilearn)

Evidently AI. (n.d.). *Accuracy vs. precision vs. recall in machine learning.* https://www.evidentlyai.com/classification-metrics/accuracy-precision-recall/.

Fortinet. (n.d.). *What Is a Port Scan? How to Prevent Port Scan Attacks?* https://www.fortinet.com/resources/cyberglossary/what-is-port-scan/.

Geeks for Geeks. (2024, November 2). *One Hot Encoding in Machine Learning.* https://www.geeksforgeeks.org/ml-one-hot-encoding/.

He, K., Kim, D., Zhang, Z., Ge, M., Lam, U., & Yu, J. (2022). *UQ IoT IDS dataset 2021.* (Dataset). The University of Queensland. Retrieved from http://dx.doi.org/10.48610/17b44bb doi: 10.48610/17b44bb

Helixstorm. (2022, August 18). *Understanding the 5 types of intrusion detection systems.* https://www.helixstorm.com/blog/types-of-intrusion-detection-systems/.

IBM. (n.d.-a). *How do neural networks work?* https://www.ibm.com/topics/neural-networks/.

IBM. (n.d.-b). *What is the KNN algorithm?* https://www.ibm.com/topics/knn/.

Kim, J., Shin, N., Jo, S. Y., & Kim, S. H. (2017). Method of intrusion detection using Deep Neural Network. In *2017 IEEE International Conference on Big Data and Smart Computing (BigComp).* IEEE. Retrieved from https://doi.org/10.1109/

bigcomp.2017.7881684 doi: 10.1109/bigcomp.2017.7881684

Knowles, C. (2023, October 26). Manufacturing sector hit hardest by 400% rise in IoT malware attacks. *SecurityBrief Australia*. https://securitybrief.com.au/story/manufacturing-sector-hit-hardest-by-400-rise-in-iot-malware-attacks.

Kundu, R. (n.d.). *Accuracy vs. precision vs. recall in machine learning.* https://www.v7labs.com/blog/f1-score-guide/.

Lail, M. A., Garcia, A., & Olivo, S. (2023). Machine learning for network intrusion detection — a comparative study. *Future Internet*, *15*(7), 243. Retrieved from https://doi.org/10.3390/fi15070243 doi: 10.3390/fi15070243

Lenaerts-Bergmans, B. (2024, May 19). *Address Resolution Protocol (ARP) Spoofing: What It Is and How to Prevent an ARP Attack.* https://www.crowdstrike.com/en-us/cybersecurity-101/social-engineering/arp-spoofing/.

Mirsky, Y., Doitshman, T., Elovici, Y., & Shabtai, A. (2018). Kitsune: An ensemble of autoencoders for online network intrusion detection. In *Proceedings 2018 Network and Distributed System Security Symposium.* Internet Society. Retrieved from https://doi.org/10.14722/ndss.2018.23204 doi: 10.14722/ndss.2018.23204

NMAP. (n.d.). *Host Discovery.* https://nmap.org/book/man-host-discovery.html/.

Paloalto Networks. (n.d.). *What is an Intrusion Detection System?* https://www.paloaltonetworks.com.au/cyberpedia/what-is-an-intrusion-detection-system-ids/.

Parte, K. (2020, November 3). *Dimensionality Reduction: Principal Component Analysis.* https://medium.com/analytics-vidhya/dimensionality-reduction-principal-component-analysis-d1402b58feb1.

Pykes, K. (2024, August 10). *Cross-Entropy Loss Function in Machine Learning.* https://www.datacamp.com/tutorial/the-cross-entropy-loss-function-in-machine-learning/.

Santiago, D. (2023, June 5). *Balancing Imbalanced Data: Undersampling and Oversampling Techniques in Python.* https://medium.com/@daniele.santiago/balancing-imbalanced-data-undersampling-and-oversampling-techniques-in-python-7c5378282290#:~:text=In%20general%2C%20under%2Dsampling%20involves,its%20representation%20in%20the%20dataset.

Saxena, S. (2024, October 10). *Binary Cross Entropy/Log Loss for Binary Classification.* https://www.analyticsvidhya.com/blog/2021/03/binary-cross-entropy-log-loss-for-binary-classification/.

Stahie, S. (n.d.). *New IoT Botnet Finds Open Telnet Ports and Brute-Forces Entry and Installation.* https://www.bitdefender.com/en-au/blog/hotforsecurity/new-iot-botnet-finds-open-telnet-ports-brute-forces-entry-installation/.

Whitfield, B. (2024, February 26). *An Introduction to the ReLU Activation Function.* https://builtin.com/machine-learning/relu-activation-function.

A Appendix A: Description of Attacks in Dataset

Attack Type	Description
ACK Flooding	A type of Distributed Denial of Service (DDoS) attack where a device is flooded with ACK packets, overwhelming it with acknowledgement messages, leading to excessive resource usage and potentially making the server unresponsive (Cloudflare, n.d.-a).
ARP Spoofing	An attack where falsified Address Resolution Protocol (ARP) messages are used to associate an attacker's address to a host's address, allowing the attacker to intercept, modify, or disrupt network traffic (Lenaerts-Bergmans, 2024).
Port Scanning	A technique where attackers check for open ports on a network device to identify available services and vulnerabilities, often as a precursor to more intrusive attacks (Fortinet, n.d.).
Service Detection	Similar to port scanning, this technique involves identifying the services running on open ports, allowing attackers to understand the network environment and potential vulnerabilities of each service (Fortinet, n.d.).
SYN Flooding	A DDoS attack that repeatedly sends a large number of SYN packets to a device to request an initial connection, without actually performing any connections, which consumes resources and can slow down or crash the device (Cloudflare, n.d.-a).
UDP Flooding	A DDoS attack where the attacker sends many UDP packets to random ports on a target, forcing it to respond with ICMP "Destination Unreachable" packets, which overloads the target's network capacity and exhausts system resources (Cloudflare, n.d.-a).
Telnet Brute Force	An attack where attackers attempt to access devices by trying different username and password combinations over the Telnet protocol (Stahie, n.d.).
HTTP Flooding	A type of DDoS attack where attackers send a massive number of HTTP requests, overwhelming it with traffic and making it unavailable to respond to legitimate users (Cloudflare, n.d.-a).
Host Discovery	A technique used to identify active devices on a network by sending probes to detect which IP addresses are active and reachable (NMAP, n.d.).

Table 11: Description of Possible Attacks in the UQ IoT IDS Dataset