Development of a Deployable Network Intrusion Detection System Utilising Machine Learning Techniques

Younis Hubsey, Billy Usher, Katie Vandrill, Zhiling Wang, Art Wieczorkowski

Summary

Cyberattacks are predicted to increase in volume and heighten their impact on businesses and other organisations in the coming years. Organisations handling and storing sensitive information, such as those in the healthcare and financial industries, are more at risk due to being high-value targets. This project investigated the efficacy of multiple machine learning models in distinguishing between normal network traffic and cyberattacks, implementing them in an intrusion detection system (IDS). The IDS was hosted on a simulated network, modelled after a hospital LAN. A variety of supervised and unsupervised models were trained on attacker behaviour data from the KDD Cup 1999 dataset. Performance evaluations of these models were conducted both inside and outside the simulation. These evaluations showed that the best-performing supervised and unsupervised models were XGBoost and K-means clustering on the KDD Cup 1999 test dataset. Although only the third-highest performing model on the test dataset, LightGBM was found to be the best performing model when implemented in the simulation, with a classification accuracy of 74%. Thus, LightGBM was determined to be the most effective intrusion detection algorithm at classifying attacks from normal network traffic for this case study.

Supervised by Dr Rasheed Hussain

School of Electrical, Electronic, and Mechanical Engineering

University of Bristol

2025

Declaration

Y.H. [illegible] KVandrill Zhiling Wang [illegible]

Work Allocation

We confirm that the information on this page accurately describes our individual contributions to the project.

Sections	**Members' contributions**				
	Younis Hubsey YH	Billy Usher BU	Katie Vandrill KV	Zhiling Wang ZW	Art Wieczorkowski AW
Summary		▲	●		
1	■	■	●		
2			●	▲	
3	●	■	■		
4	●				■
5		▲	▲	▲	●
6		●			
7	■	■	■		●
8	■				■
9	■			●	■
10	■	●			
Formatting & Editing			●		

Key:

● **This member contributed the majority/all of the work in this section**

■ **This member has partially/split contribution of the work in this section**

▲ **This member has provided research/assistance for the work in this section**

Y. H. [illegible] K.Vandrill Zhiling Wang Art

Contents

1 Introduction (YH, BU, KV)

1.1 Background

In 2024 alone, UK businesses experienced 7.78 million cyberattacks [1], with 50% of UK businesses and 30% of UK charities reporting some form of cyber breach or attack. The average cost of a single cyberattack was £10,830 for medium and large businesses [2]. These figures reflect the large cost of cyberattacks and vulnerability of organisations, especially as information technology is such an integral part of any organisation. IT has increasingly integrated with businesses to become a foundational requirement for any growing and successful business. Relevant business information is not kept on paper anymore but is digitally stored in databases or cloud platforms. After the COVID-19 lockdown, many businesses embraced a hybrid working system, thereby becoming more reliant on technology and enabling online working systems. If a business suffers a cyberattack which compromises its network, it can cripple a business and stop it functioning. This compounds the urgent need to secure a business's network.

Network security systems play a vital role in securing organisations. These systems are designed to protect networks and stored data from unauthorised access or theft [3] and aim to prevent various threats: malicious software (viruses, worms, Trojans and spyware), network intruders, data loss, etc. Businesses use various systems; this paper will focus on the design of an Intrusion Detection System (IDS).

An IDS is a tool that monitors network traffic and devices for malicious activity [4]. These systems provide an early warning of potential attacks and are completely automated. Additionally, analysis can be performed on the amount and types of attacks the IDS records on the system; this information can provide insight into missing security features. However, an IDS does not prevent attacks, meaning a reaction plan must be implemented. Also, it cannot read encrypted packets, so intruders can use these to access the network [5].

As part of an organisation's security, having an IDS is one of the fourteen principles issued by the National Cyber Security Centre's (NCSC) Cyber Assessment Framework (CAF). The CAF is an assessment that examines how organisations handle cyber risks that could affect essential functions within the organisation. There are four high-level objectives (A-D), which the principles fall into, that outline the management of risks, protection from risks, detection of events, and minimisation of event impact. An IDS falls under principle C1: security monitoring under section C, detection [6]. This means an IDS is a key principle to include in an organisation's network security framework.

However, research states that IDSs work differently depending on the algorithms involved in the design and the network on which the IDS is implemented. Depending on the combination of these factors, the IDS may not be that effective at detecting 'attack' traffic [7].

This report focuses on the design of an IDS tailored for a hospital LAN system. NHS hospitals handle large amounts of sensitive information: personal (patient name, address, gender, etc), medical (prescriptions, treatments, conditions, etc), and employee information (profession, bank details, etc). As a result, hospitals are a prime target for cyberattacks. One attack that recently took place in June 2024 was on Synnovis, a blood testing partner of several NHS hospitals, which severely disrupted services at these hospitals. Blood tests could not be performed, resulting in disruptions to many patients' treatments, including surgeries and cancer treatments [8, 9]. Additional cybercriminals claimed to have published the stolen internal data, leaving NHS patients worried about their confidentiality [10]. This attack demonstrated the severe consequences of damage and disruption to a critical service. Thus, NHS hospitals are vulnerable systems without robust cybersecurity.

Considering these factors, designing an IDS for this network type is vital. The goal of this project is to design an IDS that can operate effectively within the constraints of this case network.

1.2 *Project Aim and Objectives*

The aim of the project is:

To design an intrusion detection system using machine learning models, trained on attacker behaviour data, to effectively differentiate between normal and attack traffic within a network simulation.

To achieve this aim, several objectives must be accomplished first:

- Design and simulate a network topology using appropriate hardware, software or services.
- Find and prepare a large, labelled dataset of normal network traffic and attacks.
- Use the dataset to train and test multiple machine learning models which classify network traffic as normal or as a specific type of attack.
- Implement an intrusion detection system (IDS) which captures network traffic and uses a trained model to classify it.
- Install the IDS into the simulated network and evaluate its performance against simulated traffic, including attacks.

The project progress timeline is shown in Figure 1:

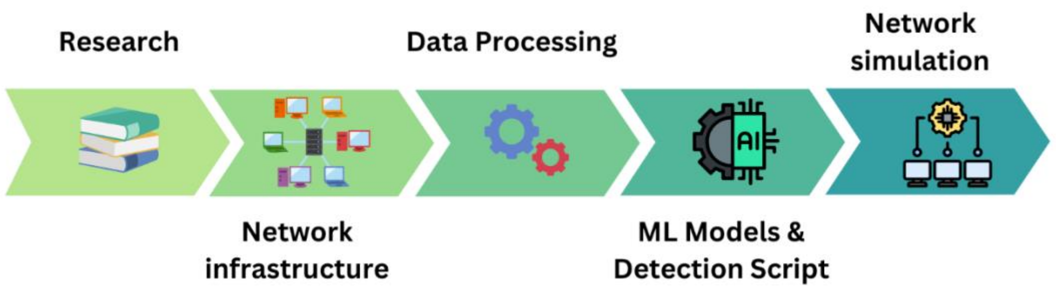


Figure 1: A Basic Project Timeline

1.3 *Methodology*

The structure of this project was based on the CRoss-Industry Standard Process for Data Mining (CRISP-DM) framework. CRISP-DM is an industry-standard model intended to streamline and guide data science projects [11]. Following this framework, the project consisted of 6 phases, detailed below.

1) **Project aims and objectives** – Adapted from CRISP-DM's "business understanding" phase, this phase included determining the aims and objectives described in Section 1.2 and forming the initial project plan.
2) **Data understanding** – This phase involved the acquirement, description and exploration of the data with which to train the machine learning models: the KDD Cup 1999 dataset. This dataset consists of 9 weeks' worth of simulated network traffic data on a simulated U.S Air Force LAN that was attacked with multiple different types of cyberattacks [12]. These four types of attacks were performed on the network and transport layers of the OSI/ISO model [13].
3) **Data preparation** – Once the data had been acquired, it was pre-processed, and features were selected.
4) **Modelling** – This phase involved the selection and training of machine learning models to be used by the IDS. In order to compare the effectiveness of signature-based and anomaly-based detection methods, both supervised models (XGBoost, random forest, LightGBM and

neural network) and unsupervised models (k-means clustering, isolation forest and Gaussian mixture model) were created.

5) **Evaluation** – After training and hyperparameter tuning, the efficacy of each model was evaluated using appropriate performance metrics, and the best-performing model was identified.
6) **Deployment** – Finally, the trained models were incorporated into an intrusion detection system, which was deployed in a simulated network. Here, further evaluation of the models took place using simulated traffic rather than a pre-prepared test set.

This study hypothesised that supervised models would better detect cyberattacks and do so with a higher percentage of accuracy in the simulation than unsupervised ones. Furthermore, the best-performing model on the KDD test data and in the simulation was hypothesised to be the XGBoost.

2 Network Infrastructure (KV)

For this case study, research needed to be conducted in order to gather information that helped establish a design for the simulated network.

2.1 Literature Review

For security reasons, the NHS keeps the information about its hospital network infrastructure private. However, there is some information available about how the network is structured.

Through research, Cisco was found to have designed a medical-grade network framework to encourage a set of proven best practices [14]. Figure 2 shows this ideal framework for hospitals to take on:

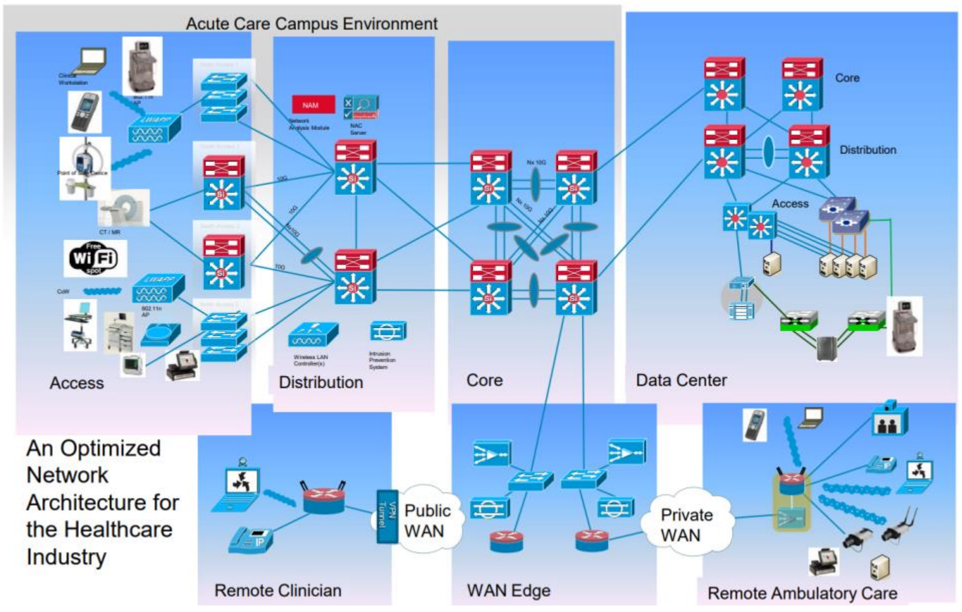


Figure 2: Cisco medical-grade network overall campus architecture [14, page 13]

This framework is designed to be applied to real-world applications; however, considering the computational limitations present in this project, such a network would be too complex to design and run in simulation. This prompted the design of a simplified network infrastructure for the purposes of the project, as described in Section 2.2.

In addition, research showed that the NHS offers Wi-Fi services split into three networks: staff, guests and public [15]. The corporate network is for NHS staff with access to patient records systems, and the staff devices must follow specific security and data protection guidelines. This network allows staff to access and update electronic patient records and connect mobile devices to deliver care. The guest network is used for business visitors or clinic staff devices that don't meet the security specifications for the corporate network. The public Wi-Fi is for members of the public, for example, patients, and allows access to personal health records and support networks [16]. This research concluded that public and staff Wi-Fi need to be included in the network infrastructure design.

Further research showed that Cisco produced a guide for the NHS on creating a network infrastructure that integrates electronic patient records (EPR) systems while applying the three main practices of agility, security, and resilience [17]. EPRs are an important part of a hospital network. This guide suggests how network segmentation can be applied to a hospital network, including segmenting medical devices and the public network into separate subnetworks. Isolating medical devices, such as heart monitors and MRI machines, results in better protection against cyberattacks. This, in turn, protects patient health and safety and protects against data breaches. A subnetwork for the public and guest W-fi means preventing unauthorised access and reducing the risk of cyber-attacks on the parts of the hospital network storing EPRs [17]. This network segregation was considered when designing the network infrastructure in the simulation.

Moreover, EPRs are becoming increasingly important due to the technological developments in the NHS. It allows for patient information to be shared safely across care settings, allowing an efficient delivery of care [18]. Thus, the network infrastructure needs to include storing and accessing EPRs.

The best topology to use for this case is the star topology. This topology is a network setup where each device, node, is connected to a central hub, such as a router or switch. The hub controls the transmission of data. When a node wants to transmit data to another node, it transmits the data to the hub first. The hub checks the destination address and sends the data to the node with that address, where the node accepts it. For a hospital network, this topology allows for easy access to data records, such as EPRS, for medical devices. Additionally, this topology allows for easy management and scalability of the network and the isolation of faulty devices [19].

This review concluded with four points to keep in mind when developing the network infrastructure design:

- Keep the design simple.
- There needs to be three points of Wi-Fi access on the network: staff, guest and public.
- The network must be segregated into at least three sections: staff, public, and medical devices.
- The system must have a way to allow staff to access and store EPRs.
- A star topology is the chosen topology to use in the network design

2.2 Network Infrastructure Design

From this research, the initial network infrastructure for simulation was designed. This design is shown in Figure 3:

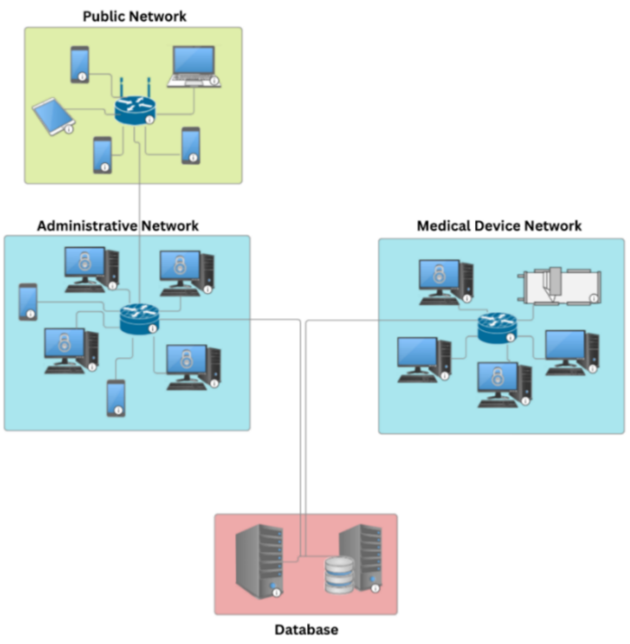


Figure 3: Initial network design for simulation

There are three subnetworks: A public network where the general public and guests can connect and access the internet; An administrative network where hospital staff can access patient and hospital information on secure devices and can connect their own secure devices to this network; A medical device network where medical and laboratory equipment are connected, and testing information is collected. The administrative and medical networks are connected to a server and a database server. Each subnetwork follows a star topology design, where the devices on each subnetwork are connected to a router, as it is easy to manage and scalable, meaning the IDS can easily adapt to any hospital infrastructure. The purpose of the database server is to store information on EPRs, and the server manages the applications used in the hospital, such as staff communication systems and applications for viewing and updating EPRs.

However, there are still flaws with this initial design. The router of the public subnetwork is connected to the router of the administrator system; this comes with security risks, as an attacker can potentially connect to the administrator network by connecting to the public router, allowing access to sensitive information. Additionally, it can be simplified further by needing only the database server, as hospitals may have different ways of running applications on the network. It may just run from a database server. This database server is still needed because hospitals need a way to store EPRs, even if some hospitals have different methods of storing this information, such as using cloud storage. However, keeping it simple will allow the IDS to be adapted to protect real-life networks.

The final design that influenced the simulation network design is shown in Figure 4.

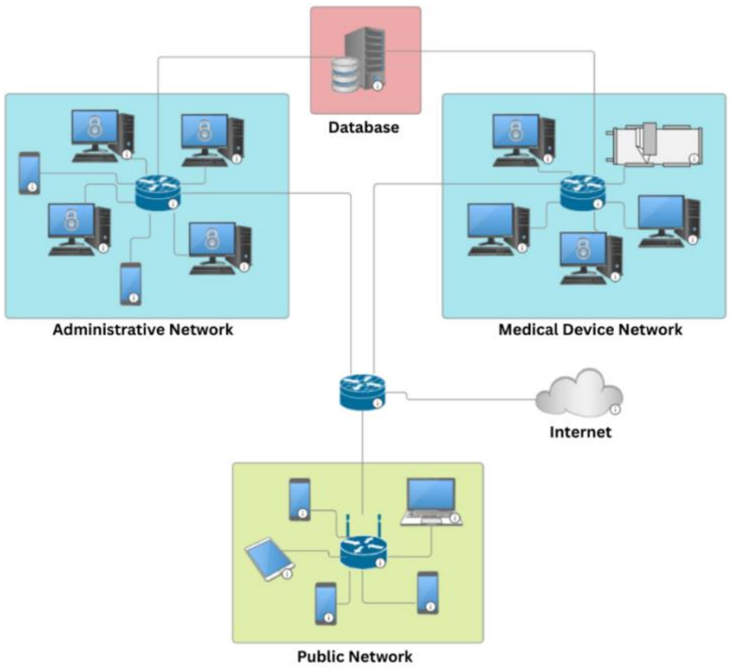


Figure 4: Final network infrastructure design used to influence the simulated network design

3 Dataset (YH, BU, KV)

3.1 Data Understanding

The machine learning models employed by the IDS were trained using a 10% subset of the KDD Cup 1999 (KDD'99) dataset, which was initially used for the Third International Knowledge Discovery and Data Mining Tools Competition in 1999. The task was to build a classifier capable of distinguishing "good" (normal) connections over a LAN from "bad" connections (i.e. intrusions/attacks) . The unique challenge for the project described in this report was to incorporate such a classifier into a real-time system and to evaluate its performance in a novel simulated environment.

The KDD'99 dataset is a version of the dataset acquired by MIT Lincoln Labs' 1998 DARPA Intrusion Detection Evaluation Program, consisting of 9 weeks of traffic captured in a simulated U.S. Air Force LAN. The raw TCP dump data has been processed into "connection records", which form the datapoints of the dataset. A connection is defined as "a sequence of TCP packets starting and ending at some well-defined times, between which data flows to and from a source IP address to a target IP address under some well-defined protocol". Each connection contains 41 features, split into three main categories: basic features, content features, and time-based features [11].

Basic features are individual TCP connections derived from packet headers without inspecting the payload. Meaning these features are ideal for training algorithms on detecting large attacks, such as DoS. Content features are used to access the packet payload. Finally, time-based features are used to capture properties that mature over a two-second timeframe. Depending on the data types, these features are further categorised into being discrete or continuous [20]. This distinction is important for encoding, which is explained further in section 3.2.1.

The KDD Cup 1999 is the most widely cited and used dataset for IDS development and machine learning areas [21], making it the most appropriate benchmark for comparing the effectiveness of different machine learning models in this case study. Although created in 1999, making it a 26-year-old dataset that could be seen as outdated, the KDD Cup is unlike any common IDS datasets [21].

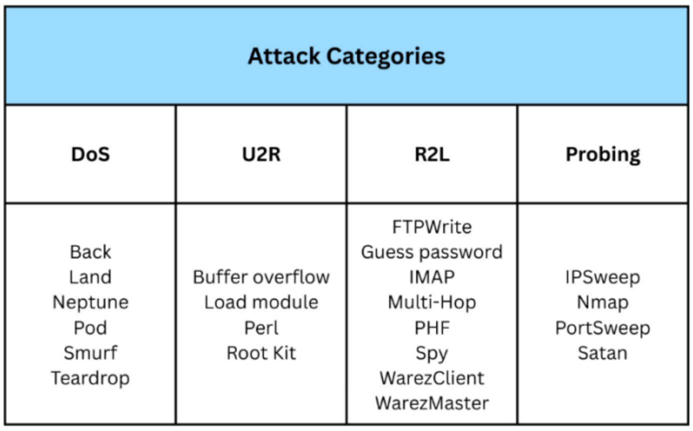
The training dataset used in this project contains 494,021 records, with 97,278(19.7%) labelled as normal network traffic records. Meanwhile, the test dataset, 'Corrected KDD', contained 311,029 records, with 60593 normal records. The remaining records in each dataset represented the attacks [20]. There were 22 attack types within the attack data, which fell into one of four categories: DoS, Probing, R2L, and U2R.

3.1.1 Attack Types

The KDD attacks have been categorised into four different attack types:

- Denial of Service (DoS): This attack aims to overload a network so the performance is lowered or part/all of the network is taken down. A successful DoS attack on a network will take time and money to repair the damage [22, 23].
- User-to-Root (U2R): When a non-privileged user, an intruder on the network, obtains local administrative user (root) access via a specific computer or a system [11, 23].
- Remote-to-Local (R2L): This attack involves a remote attack attempting to gain local access to a network by sending packets to a target's computer [23]. An attack is successful once the attacker is on the system locally and can launch an attack, such as stealing sensitive information.
- Probing: a reconnaissance attack where an attacker gathers information on the targeted network or computer system before launching an attack [11, 23]. This is still considered an attack as they gain unauthorised access to the network with the intent to cause harm or extract sensitive information.

Table 1 shows the distribution of attacks between these four categories:



Attack Categories			
DoS	**U2R**	**R2L**	**Probing**
Back Land Neptune Pod Smurf Teardrop	Buffer overflow Load module Perl Root Kit	FTPWrite Guess password IMAP Multi-Hop PHF Spy WarezClient WarezMaster	IPSweep Nmap PortSweep Satan

Table 1: The 22 KDD'99 attack types sorted into their categories

3.1.2 Exploratory Data Analysis (EDA)

An exploratory analysis was conducted on the data to understand the dataset better. This explored: class imbalance, feature distribution, missing values and outliers. This involved plotting charts to visualise class distribution and plotting histograms of each feature. Kernel density estimation plots were also overlaid in red on the histogram plots, which allowed for the visualisation of kurtosis and skewness. The KDD Cup 1999 dataset had zero missing values as it had already been processed. However, it did contain duplicate records, which were removed. The dataset was found to include 22 attack types. Upon further inspection of the distribution of these attack types, it was found that there was an extreme distribution imbalance between the attack types. Attacks such as Spy and Perl have fewer than five samples, while attacks such as Neptune and Smurf have over 100,000. This imbalance was plotted on a bar chart shown in Figure 5.

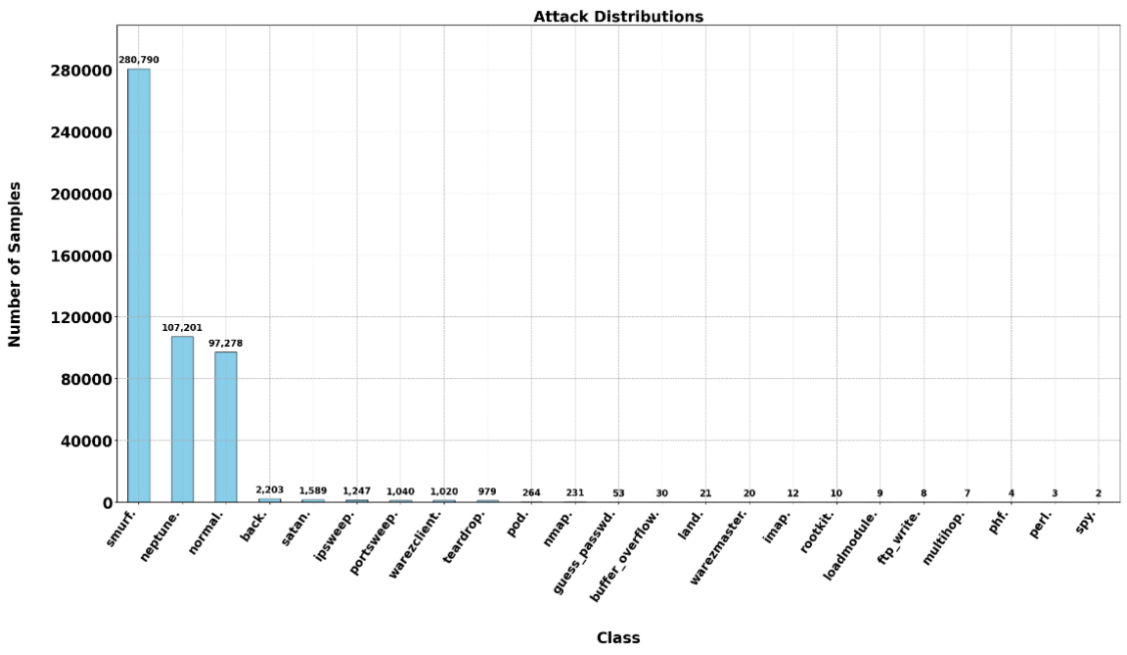


Figure 5: Visualisation of class imbalance between the 22 attack types

From Figure 5, the attack types that hold the majority of samples from the dataset are Smurf, Neptune, and normal. To mitigate this imbalance, all 22 attacks were categorised into five attack groups depending on the type represented by the attack. These five categories were: Normal, U2R, R2L, Probing, and DoS. The updated class distribution is shown in Figure 6 below.

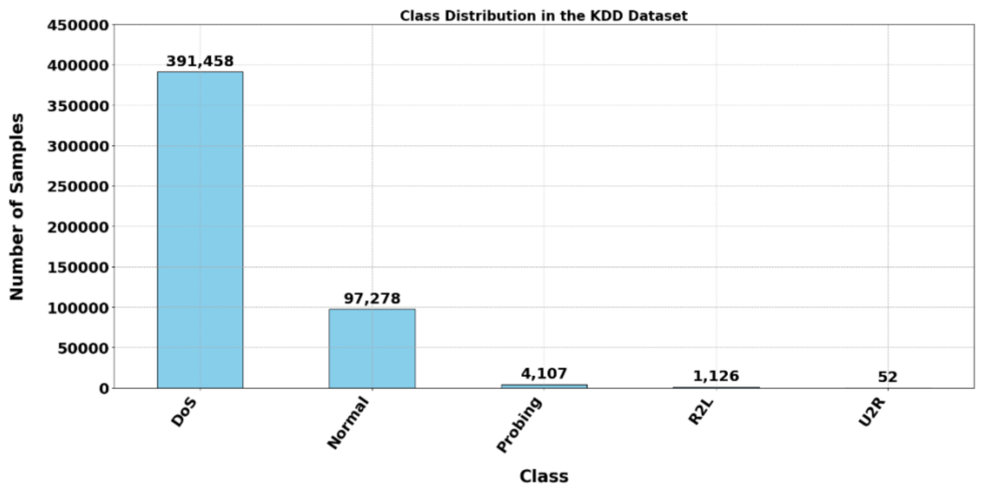


Figure 6: Class distribution between the five categories

The class imbalance has slightly improved, as shown in Figure 6. However, U2R is still too low and has an extreme lack of samples, which can result in multiple problems. Oversampling techniques, such as SMOTE and random oversampling, cannot work effectively, as they will not have enough samples to perform oversampling effectively. Models will usually treat classes with such low samples like U2R as just noise and at times these samples will not be classified, this can harm the performance of models. Therefore, U2R was removed from the dataset due to having too few samples. However, despite U2R being removed, there was still a significant class imbalance between the remaining classes, indicating the need for a sampling technique. The sampling technique used is explained in the sampling section 3.2.4.

After addressing the class imbalance, the next stage was to explore the distributions of the features within the dataset. This was done using skewness and kurtosis, which aid in determining if and what scaling is needed. Skewness measures the lack of symmetry within a distribution [24]. A distribution is positively skewed when there is a long tail to the left, which indicates positive skewness. In contrast, a distribution is negatively skewed when there is a long tail to the right, which indicates negative skewness. Meanwhile, Kurtosis measures the combined weight of a distribution's tail relative to the mean. It measures the tailedness of the data relative to a normal distribution [24]. The KDE plot can take different forms depending on the value of the kurtosis; the majority of the features within the KDD Cup dataset have a high kurtosis and a leptokurtic distribution, which shows a sharp peak, steep gradient and heavy tails [24]. Out of all the KDD features, 'wrong_fragment' best represents this distribution, as shown in Figure 7 below.

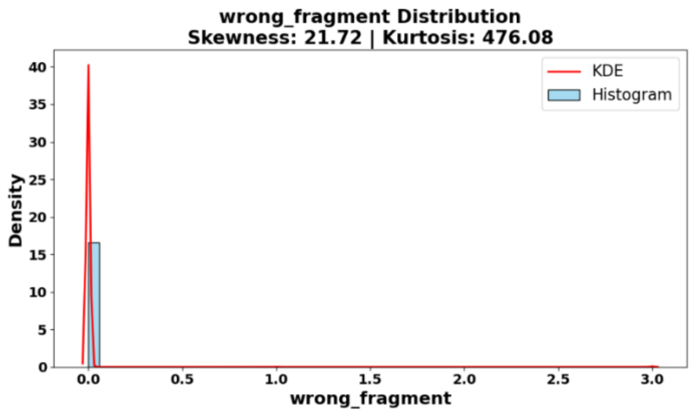


Figure 7: Distribution of 'Wrong_fragment' feature

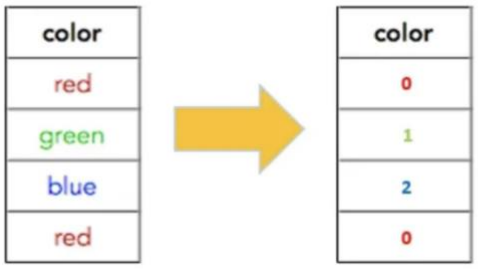
As illustrated in Figure 7, there is high kurtosis of 476.08 and heavy skewness of 21.72; this is supported by the KDE plot's sharp peak, steep gradient and long fat tail to the right, indicating high kurtosis and skewness. This type of non-normal distribution can negatively impact machine learning models, specifically Neural networks, as gradients are used to learn and improve predictions. Having steep gradients like this can negatively affect the learning of the neural network model used in this project, the MLP classifier. Therefore, it was decided that this data could benefit from scaling. The appropriate scaler used is discussed further in the data preparation phase in section 3.2.3.

3.2 Data Preparation

3.2.1 Encoding

the next stage was to encode the features to prepare them for feature selection and to be used in machine learning models. Appropriate encoding is necessary because machine learning models can only effectively work with numerical values [25]. Within this dataset, multiple categorical features must be encoded; these are "protocol_type", "service", and "flag". For this project, both label encoding and one-hot encoding are used. One-hot encoding is used for the input variables, while label encoding is used for the target variable.

Label encoding is a technique that converts categorical values into numerical values to be digested by machine learning models. This method assigns each unique category to a unique integer, effectively converting categorical data into numerical values [26]. Label encoding is appropriate for target variables and categorical data with an ordinal ranking. Applying this encoding to data that does not have an ordinal ranking, such as "protocol_type", can create a hierarchy that does not exist, which can mislead machine learning models. In Figure 8, a visual example of label encoding is shown.

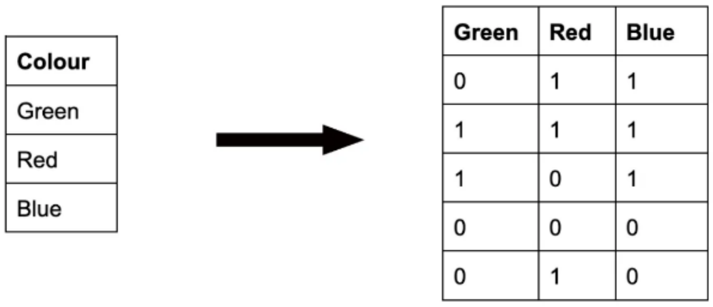


color
red
green
blue
red

color
0
1
2
0

Figure 8: Visualisation of Label encoding [27]

One-hot encoding is the process of converting categorical data into numerical data to be fed into machine learning algorithms. Each categorical value is converted into a categorical column, and a binary value of 0 or 1 is assigned to these columns [28]. The advantage of one-hot encoding is that categorical data does not create any ordinal relationships that can mislead the machine learning models. This is perfect for the categorical features within this dataset, such as protocol_type. However, using one-hot encoding can increase the dimensionality of the data, as each unique category in a feature has a column created for it. This can lead to the curse of dimensionality, where the increase in new features can negatively impact the models' performance and computational efficiency as the amount of data that needs to be generalised increases exponentially [29]. This can also make it challenging to perform feature selection. An example of this encoding is visualised below in Figure 9.



Colour
Green
Red
Blue

Green	Red	Blue
0	1	1
1	1	1
1	0	1
0	0	0
0	1	0

Figure 9: An example of One-Hot-Encoding [30]

Considering the functionality of both encoders, it was decided that during feature selection and visualisation, all features and the target variable were label encoded; however, when building the machine learning algorithms, the input categorical variables were One-hot-encoded, while the target variable was label encoded. This is because the one-hot encoding reduced interpretability during feature selection, especially for high-cardinality features in the KDDCUP99 dataset, such as ‘service’. One-hot encoding this feature caused the number of columns to increase significantly, complicating the univariate and bivariate feature selection processes. Therefore, separate encoding methods were used for each stage. Label and one-hot encoding were used to build the machine learning models, whiles only label encoding was used for visualisation and feature selection. The encoding caused the attack categories to be assigned numeric labels; therefore, throughout this project, the attacks will be named as normal(class 0), DoS (class 1), Probing(class 2), R2L(class 3), and U2R(class 4). Now that categorical features have been encoded, the data is ready for bivariate and univariate feature selection.

3.2.2 Feature selection

A structured feature selection process ensured that the machine learning models were trained on the most informative and non-redundant features. This process consisted of a combination of univariate and bivariate analysis. Mutual Information (MI), a univariate feature selection method, calculates the amount of information one variable holds about another variable [31]. Each feature in the dataset was measured by how much it contributes to predicting the output feature. Features with a high MI score, close to 1, are informative, while features close to 0 are less informative and do not contribute much to predicting the target variable. MI was selected over other feature selection methods for the tree-based models, such as feature importance, as this method would need to be computed for each tree model, increasing computation time; therefore, to streamline the feature selection for the tree-based models, MI was used.

The other method used was the Pearson correlation coefficient (R), a bivariate feature selection method that calculates the linear relationships, specifically correlation, between two numerical features [32].

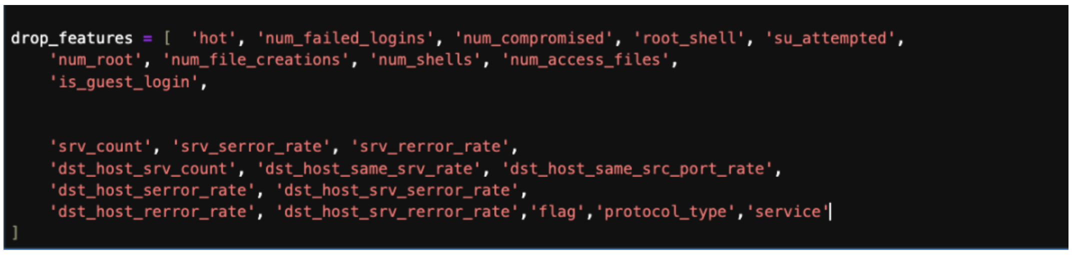
A correlation matrix heatmap was utilised to visualise the calculation of Pearson's R. Redundant features were then identified through this matrix. The Correlation matrix heatmap of all 41 features is shown in Figure A (found in the Appendix section of this report).

Figure A shows that the colour intensity and value of the correlation coefficient (R) reveal the correlation between two features. Dark red indicates a strong positive correlation, while dark blue indicates a strong negative correlation. A high positive correlation between two features is indicated by a value closer to 1.00, while a negative correlation is indicated by a value closer to -1.00. No correlation is colour close to white on the heatmap and represents a value close to 0.

The univariate feature selection MI and bivariate feature selection correlation matrix were combined to create a robust feature selection method. These two were used together because the MI will not provide information on correlated features; therefore, only using MI can cause uncorrelated features, which are non-redundant, to be accidentally removed, while redundant ones are kept. While the correlation matrix does not provide information on how important certain features are towards the target variable, using only a correlation matrix can cause informative features to be unintentionally removed. Categorical features such as 'protocol_type', 'flag' and 'service' were excluded from the correlation matrix and MI scoring; these features were labelled for preprocessing purposes. However, the encoded values of these features do not represent ordinal relationships. Therefore, misleading interpretations can be formed from these features included in the feature selection processes.

For this project, a threshold of 0.7 was found to be the most optimal for the correlation matrix; this means any features with an R score of higher than 0.7 are flagged as highly correlated. For the MI, a threshold of 0.005 was the most optimal; features with an MI score smaller than 0.005 were flagged as uninformative and could be removed.

If two features were highly correlated, the two MI scores were compared. The feature with the lower MI score was removed. This was done iteratively, eventually leading to this list of features being dropped from the database. This is visualised in Figure 10.

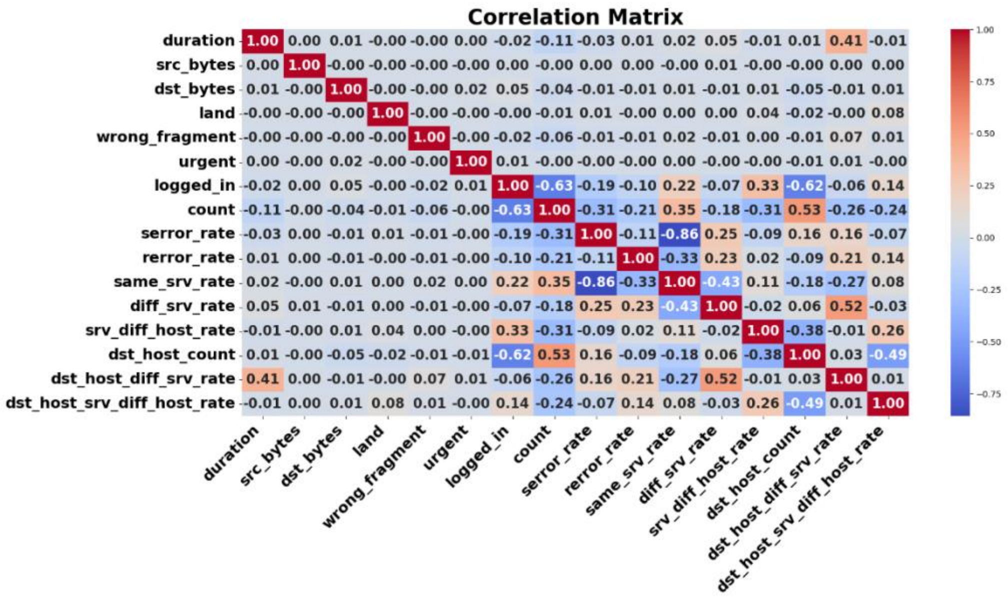


```
drop_features = [  'hot', 'num_failed_logins', 'num_compromised', 'root_shell', 'su_attempted',
    'num_root', 'num_file_creations', 'num_shells', 'num_access_files',
    'is_guest_login',


    'srv_count', 'srv_serror_rate', 'srv_rerror_rate',
    'dst_host_srv_count', 'dst_host_same_srv_rate', 'dst_host_same_src_port_rate',
    'dst_host_serror_rate', 'dst_host_srv_serror_rate',
    'dst_host_rerror_rate', 'dst_host_srv_rerror_rate','flag','protocol_type','service'
]
```

Figure 10: List of features dropped from the dataset

As shown in Figure 10 the content features were majority features dropped from the kddcup99 dataset however this is understandable as they often contain sparse or useless information, prior studies such as prof Danijela D review of kddcup99, NSL-kdd and kyoto2006 [33] also came to a similar conclusion as improved datasets of the kddcup99 such as kyoto2006 found majority of content features also to be redundant and removed. Other features, such as srv_count, were also removed as they were redundant and had a lower MI score compared to their correlated partner. This was the final resulting correlation matrix, showing the most optimal features, as shown below in Figure 11.



	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	logged_in	count	serror_rate	rerror_rate	same_srv_rate	diff_srv_rate	srv_diff_host_rate	dst_host_count	dst_host_diff_srv_rate	dst_host_srv_diff_host_rate
duration	1.00	0.00	0.01	-0.00	-0.00	0.00	-0.02	-0.11	-0.03	0.01	0.02	0.05	-0.01	0.01	0.41	-0.01
src_bytes	0.00	1.00	-0.00	-0.00	-0.00	-0.00	0.00	-0.00	0.00	0.00	-0.00	0.01	-0.00	-0.00	0.00	0.00
dst_bytes	0.01	-0.00	1.00	-0.00	-0.00	0.02	0.05	-0.04	-0.01	-0.01	0.01	-0.01	0.01	-0.05	-0.01	0.01
land	-0.00	-0.00	-0.00	1.00	-0.00	-0.00	-0.00	-0.01	0.01	-0.00	0.00	0.00	0.04	-0.02	-0.00	0.08
wrong_fragment	-0.00	-0.00	-0.00	-0.00	1.00	-0.00	-0.02	-0.06	-0.01	-0.01	0.02	-0.01	0.00	-0.01	0.07	0.01
urgent	0.00	-0.00	0.02	-0.00	-0.00	1.00	0.01	-0.00	-0.00	-0.00	0.00	-0.00	-0.00	-0.01	0.01	-0.00
logged_in	-0.02	0.00	0.05	-0.00	-0.02	0.01	1.00	-0.63	-0.19	-0.10	0.22	-0.07	0.33	-0.62	-0.06	0.14
count	-0.11	-0.00	-0.04	-0.01	-0.06	-0.00	-0.63	1.00	-0.31	-0.21	0.35	-0.18	-0.31	0.53	-0.26	-0.24
serror_rate	-0.03	0.00	-0.01	0.01	-0.01	-0.00	-0.19	-0.31	1.00	-0.11	-0.86	0.25	-0.09	0.16	0.16	-0.07
rerror_rate	0.01	0.00	-0.01	-0.00	-0.01	-0.00	-0.10	-0.21	-0.11	1.00	-0.33	0.23	0.02	-0.09	0.21	0.14
same_srv_rate	0.02	-0.00	0.01	0.00	0.02	0.00	0.22	0.35	-0.86	-0.33	1.00	-0.43	0.11	-0.18	-0.27	0.08
diff_srv_rate	0.05	0.01	-0.01	0.00	-0.01	-0.00	-0.07	-0.18	0.25	0.23	-0.43	1.00	-0.02	0.06	0.52	-0.03
srv_diff_host_rate	-0.01	-0.00	0.01	0.04	0.00	-0.00	0.33	-0.31	-0.09	0.02	0.11	-0.02	1.00	-0.38	-0.01	0.26
dst_host_count	0.01	-0.00	-0.05	-0.02	-0.01	-0.01	-0.62	0.53	0.16	-0.09	-0.18	0.06	-0.38	1.00	0.03	-0.49
dst_host_diff_srv_rate	0.41	0.00	-0.01	-0.00	0.07	0.01	-0.06	-0.26	0.16	0.21	-0.27	0.52	-0.01	0.03	1.00	0.01
dst_host_srv_diff_host_rate	-0.01	0.00	0.01	0.08	0.01	-0.00	0.14	-0.24	-0.07	0.14	0.08	-0.03	0.26	-0.49	0.01	1.00

Figure 11: Correlation Matrix after feature selection

'same_srv_rate' and 'serror_rate' were kept as substitutes for other collated features, despite having a correlation of -0.86. Therefore, removing either can cause important information to be lost. This was confirmed later in the project during training of the models, performance worsened when one of these features was removed. This process was used on both the supervised tree-based models and the unsupervised models due to its agnostic nature, being able to work effectively for a range of models.

For the Neural network model used in this project, feature selection was done via permutation feature importance. This feature selection method is used to measure the importance of each feature in a specific model. Features are randomly shuffled to see how each feature affects the model's performance [34]. Permutation feature importance was explicitly used for the neural network model, as, unlike the tree-based models, algorithms such as neural networks can be too complex to interpret using univariate and bivariate feature selection methods. The results from the permutation feature importance are shown in Figure 12 below.

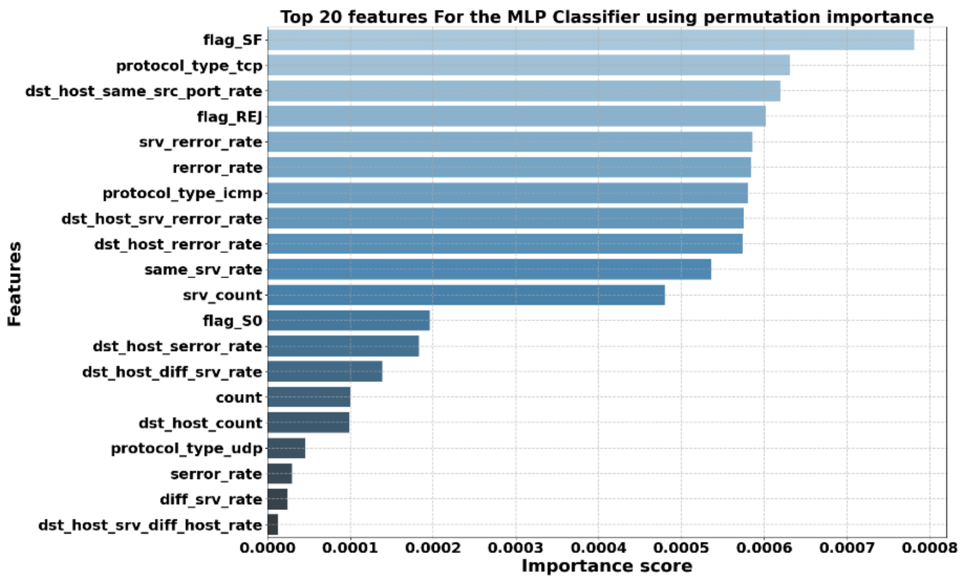


Figure 12: Chart visualising Neural network permutations

Similar to the previous findings, most of the features that can be discarded are the content features from Table 2 in the KDDCup99 schema [12]. Overall, similar features are kept between the two selection methods; however, there are slight differences.

For the unsupervised models, a variance threshold was also used as hyperparameters for feature selection. Variance threshold is a feature selection method that removes low variance features [35]. However, the same features were dropped by the MI and correlation matrix combination for the supervised models were also dropped by the variance threshold, this is because the variance threshold was kept at 0.00 for all 3 models since the univariate and bivariate feature selection process explained above was already applied to the unsupervised models.

3.2.3 Scaling

As discussed in the data exploration stage, for this project, scaling is necessary for models such as the MLP classifier. A robust scaler and a min/max scaler were preferred over a standard scaler as the standard scaler assumes that features are normally distributed [36]. However, the majority of the features do not follow a normal distribution. Robust scaler was then preferred over min/max scaler due to the extreme kurtosis and skewness values of the features within the dataset. An example would be “num_compromised”, which had a skewness value of 417.53 and a kurtosis value of 188119.4, this type of extreme skewness and kurtosis was seen throughout the dataset across majority of the features. The Robust scaler is better at handling extreme values like this as it removes the median and scales the data according to the quantile range [37]. Compared to a min/max scaler that fixes all features in a distribution between (0,1), which can cause important information about certain features to be lost.

However, after testing, the min/max scaler resulted in better performance in the models compared to the robust scaler. Therefore, the min/max scaler was chosen as the scaler to be used.

The min/max scaler transforms features by scaling each feature to a given range, usually between 0 and 1 [38]. Min/max scaler doesn’t reduce the effect of outliers, but it linearly scales them down into a fixed range, where the largest occurring data point corresponds to the maximum value and the smallest one to the minimum value.

The min/max scaler is calculated as:

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}},$$

where X is the feature value and X_{min} is the smallest occurring data point and X_{max} is the largest occurring data point.

3.2.4 Sampling

As shown previously in Figure 6, the KDD Cup 1999 was highly imbalanced which can negatively affect the performance of machine learning models abilities to classify minority classes. A class imbalance of this size can cause a majority class to overlap on the feature space of other minority classes, making separating and classifying the different classes much more difficult [39].

Tackling this imbalance is therefore crucial for this project, especially for rare attacks such as R2L. Failure to detect these rare but critical attacks in a hospital network can prove to be fatal, as people's lives and personal information would be at risk. An effective sampling technique was required to minimise the class imbalance within the dataset. SMOTE-tomek was finally selected over other sampling techniques such as random oversampling, standard SMOTE and other variants of SMOTE.

Random oversampling is a sampling technique that increases the number of samples for minority classes to balance class distribution by duplicating the existing samples. However, this can lead to overfitting as the model learns to memorise the same duplicated data. Therefore, random oversampling was not chosen, and methods that tackle this limitation, such as the different variants of SMOTE, were investigated.

SMOTE is an oversampling technique that creates synthetic samples for smaller classes in order to balance the dataset. Although SMOTE improves upon the random oversampling by creating synthetic samples instead of duplicating existing samples, it does not address the issue of overlapping classes and noisy samples, which are commonly present in network intrusion datasets like KDDCUP99. For this reason, variants of SMOTE that also remove overlapping samples were sought. The KDD Cup has a small number of positive instances (80% normal, 20% attacks). As Gustavo found, SMOTE-EEN or SMOTE-Tomek are the most appropriate sampling techniques for a similar situation.[40]

SMOTE-EEN is a variant of SMOTE that introduces an additional step called Edited Nearest Neighbours (EEN). This step addresses the limitations of SMOTE by removing noisy or overlapping samples. Similarly, SMOTE-Tomek combines SMOTE's ability to create synthetic samples with Tomek links' ability to remove borderline or noisy samples, thereby reducing overlapping datapoints. Tomek links identify pairs of datapoints from different classes that are close in proximity in the feature space; these pairs are removed to create a more defined boundary between classes. A visual representation of Tomek links is shown below in Figure 13.

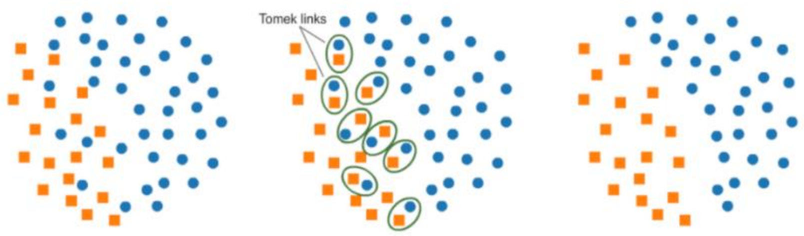


Figure 13: Visual representation of tomek links [41]

For this project, SMOTE-Tomek was preferred over SMOTE-EEN. This is because SMOTE-EEN is much more aggressive with its removal of samples, which can result in too many samples being removed and more information that is needed by the model being lost, compared to SMOTE-Tomek, which is more lenient with misplaced samples or borderline samples.

The ‘sampling_strategy’ parameter within SMOTE-Tomek determines which samples will be oversampled with synthetic samples. For this project, a custom sampling strategy was implemented for the tree-based models, while ‘Not Majority’ was implemented for the Neural Network. The most optimal synthetic samples to add to each class was 40,000 for probing(class 2) and 391,400 samples for R2L(class 3).

To confirm SMOTE-Tomek's effects on the imbalanced dataset, a t-distributed Stochastic Neighbour **Embedding** (t-SNE) was utilised. t-SNE is a non-linear dimensionality reduction technique that maps high-dimensional data into a 2d or 3d space while preserving local structures[42]. It is appropriate for visualising clusters and overlapping [42], which is prevalent within this dataset. t-SNE was preferred over other dimensionality reduction techniques, such as PCA(full name), to visualise the class imbalance, as t-SNE provides a more meaningful visual assessment of the effects of SMOTE-Tomek on the class imbalance within the dataset. Unlike PCA, t-SNE preserves the relationships between data points in a lower-dimensional space [42]. Additionally, PCA cannot handle non-linearity, whereas t-SNE is specialised to handle non-linear datasets, such as the KDD Cup.

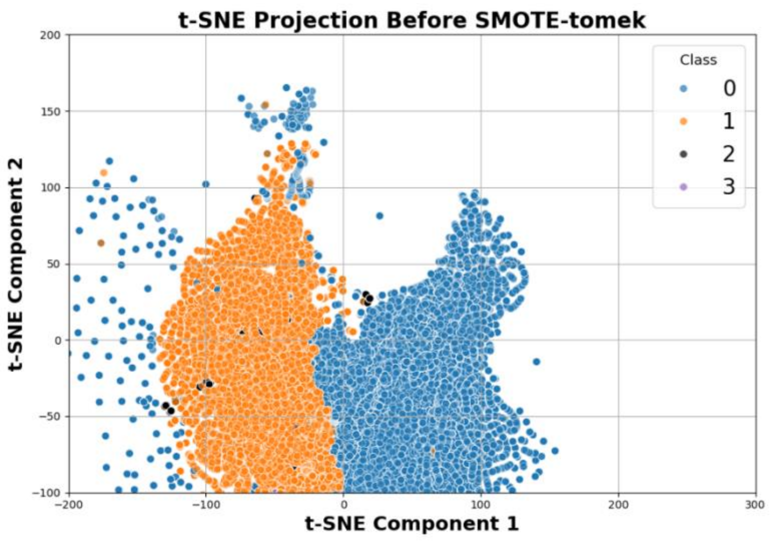


Figure 14: t-SNE plot of classes

Figure 14 shows the t-SNE plot of the class imbalance of the KDD Cup dataset; as evident, classes 0 and 1 have defined borders and dominate the plot; this is understandable as classes 0 and 1 are the majority, therefore these two classes will overlap and cover the minority classes. The lack of visibility of classes 2,3, and 4 within the plot highlights the severity of the class imbalance.

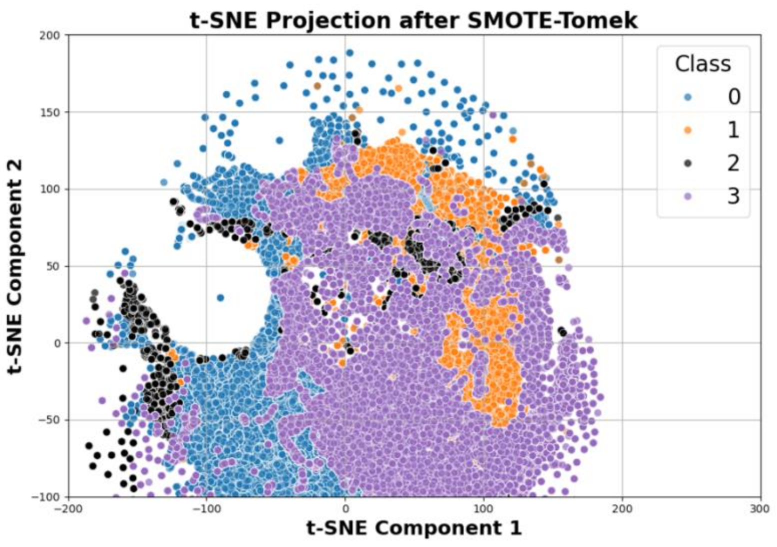


Figure 15: t-SNE plot of classes after SMOTE-Tomek with customer sampling strategy applied

Figure 15 shows that the classes are much more evenly mixed, and all classes are now visible within the distribution space with less overlap. This change will improve the detection and classification of each class by the machine learning models, as each class now has a better chance of being classified by the models.

For the Neural network model, the 'Not Majority' strategy resulted in the best performance. Neural networks are very sensitive towards class imbalance; therefore, a more aggressive sampling strategy was needed to balance the dataset more effectively. Figure 16 shows the t-SNE plot after the 'not majority' sampling strategy is applied. As shown in Figure 16, most classes can now be compared to Figure 14.

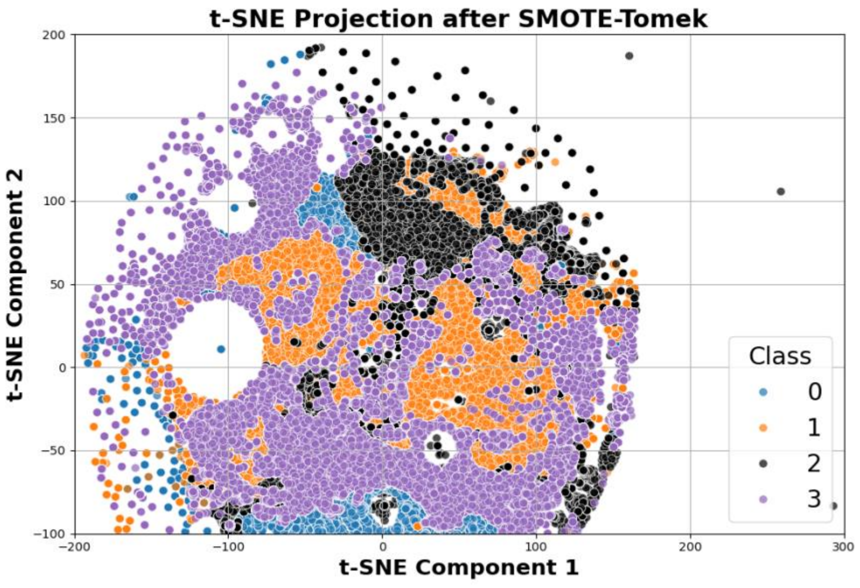


Figure 16: t-SNE plot after 'not majority' sampling was applied

4 Intrusion Detection Algorithms (YH, AW)

4.1 Introduction

At this stage, the supervised models utilised for the classification of attacks within the KDD Cup 1999 dataset and the unsupervised models deployed for anomaly detection were introduced. Four supervised models and three unsupervised models were selected based on their proven performance within intrusion detection.

4.1.1 Supervised Introduction

Supervised learning Models are machine learning based algorithms that use labelled data to understand the relationships between input and output data [43]. The main aim of supervised learning is to create a model that can learn the data given to it so that it can predict the correct outputs on real-world data [43]. The model is trained on data to learn patterns, which it then uses to predict new, unseen data.

4.1.2 Unsupervised Introduction

Unsupervised Learning (UL) models are machine learning based models trained on unlabelled data. Rather than relying on explicit guidance from the dataset, unsupervised models explore inherent patterns, structures, and relationships from the data points' features. Afterwards, the model may further process and restructure the dataset to display its predictions.

Regarding Unsupervised Network Intrusion Detection Systems (UNIDS), they make up for the shortcomings of traditional Network Intrusion Detection Systems (NIDS), which utilise supervised approaches [44]. However, challenges unique to UNIDS leave them as an active area of research and not widely used in practical NIDS. One shortcoming with supervised IDS is the reliance on a database of known attack signatures; it requires human experts to maintain and continuously update [45]. A key threat that UNIDS are not vulnerable to are zero-day attacks. Zero-day attacks exploit traditional IDS, with their existence being not yet known; there are no defences or known signatures for these attacks [46]. Unsupervised models only observe patterns with received data records, instead of analysing signatures.

One challenge with UL models is how performance is evaluated [47]. Labelled data is required to determine the model's outcome; however, it is not feasible in practice, experts must analyse results manually for external evaluation. Additionally, without labelled data, UL models must make assumptions about the data when making predictions [47]. UL models assume normal network traffic is frequent and has similar features. Abnormal behaviour is assumed to occur rarely and have standout network features. Using a small amount of training and simulation data, unsupervised models can only classify data points in a binary fashion. Once the UL algorithm has learned the inherent relationships/structures with the dataset, it can only determine whether the record indicates normal or anomalous traffic.

4.2 Supervised Models

4.2.1 Random Forest:

Random Forest classifier is a supervised ensemble machine learning model that combines multiple weak learners, such as decision trees. This model uses bootstrap aggregating(bagging), an ensemble method that trains multiple decision trees in parallel and aggregates the predictions using a majority vote [48]. The bagging of multiple decision trees increases the performance, leading to better results than a single decision tree. However, a disadvantage is that it creates similar split decisions across trees, which can reduce the diversity of the models in the ensemble. Random Forest tackles this limitation by adding diversity via randomness; at every split, a Random subset of features is used [49]. This randomness prevents overfitting and allows the random forest to handle noisy and high-dimensional data, such as the KDD Cup dataset, much better.

Random Forest utilises the CART algorithm to build individual decision trees. At each node of a decision tree within the forest, the CART algorithm calculates the node with the lowest Gini impurity score. Gini impurity is measured to evaluate how pure a node is, if a node primarily consists of one class, for example 90% Dos class and 10% normal class compared to a node that is 60% Dos and 40% normal the former is said to be purer and have a lower Gini impurity score. A lower Gini impurity score indicated that a node is more "pure" [50].

The Gini impurity is calculated as:

$$G = \sum_{i=1}^{C} p(i) \cdot \left(1 - p(i)\right),$$

where $p(i)$ *is the proportion of the samples of* class i at node t and C is the number of classes

The split that leads to the greatest reduction in Gini impurity is chosen. In the context of this project, the split that makes the best distinction between normal traffic and attacks like Dos, Probing, etc.

This process is repeated for all the trees in the random forest. Once all trees have given a prediction, a voting method for classification tasks aggregates the results from the trees in the random forest and selects the most common outcome among them.

Random Forest was selected for this project for a multitude of reasons. It is highly scalable and appropriate for large datasets like the KDD Cup. Random Forest classifier is also flexible; it can handle unscaled data well, and as shown in the EDA section above, this dataset has very uneven distribution; therefore, utilising a model that can handle such uneven distribution is essential. The KDD Cup dataset is also non-linear; random forest models are known to handle non-linear data well. Random Forests generally produce high accuracy compared to other models, as the aggregation of multiple decision trees reduces the chance of overfitting compared to a single decision tree [51], making it a good contender for producing an IDS with few false positive and negative detections.

4.2.2 Extreme gradient boosting (XGBoost)

XGBoost, also known as Extreme Gradient Boosting, is an improved variant of the gradient boosting algorithm [52]. Similar to what was seen in the random forest XGBoost also combines decision trees however unlike random forests it utilises boosting. Boosting is where the decision trees learn sequentially unlike random forest where they learn in parallel. In XGBoost each tree is dependent on the outcome of the previous tree and each new tree learns and corrects the mistakes of the previous tree. This sequential learning process causes the performance to improve each iteration. This differs from the Random Forest, where the trees work in parallel and are independent.

XGBoost evaluates each prediction made by the decision trees by using a loss function, which calculates the difference between the prediction and actual values. After each iteration, the loss function is adjusted to minimise this loss. This loss function helps the model learn from its errors making it better at predictions as it learns from its mistakes. Regularisation is also used within XGBoost to reduce overfitting by penalising trees that are too complex.

XGBoost was selected for this project because it is a high-performance model generally designed for great scalability and efficiency. It is robust against overfitting and it is great handling large imbalanced datasets, such as the KDD Cup dataset.

4.2.3 LightGBM

LightGBM is another tree-based model that was used for this project. LightGBM is very similar to XGBoost as it is a type of gradient boosting machine [1]. However, LightGBM is more memory efficient and generally faster than XGBoost.

Unlike XGBoost, Light-GBM uses Gradient-Based-One-Side-Sampling (GOSS) and Exclusive Feature Bundling (EFB), two techniques that make LightGBM faster than XGBoost. GOSS ignores gradient calculations that are not very informative and do not have a significant effect on the model's performance which in turn reduces training time as the model has less gradient calculations to go through whiles XGBoost would calculate every gradient calculations including the non-informative ones. EFB works by combining mutually exclusive features into one group; grouping these features reduces the number of splits needed to process the data for example if there are 10 features, using EFB these can be grouped into two groups with 5 mutually exclusive features in each group, this cuts the splits needed from 10 to 2 whiles xgboost would go through the full 10 splits. This reduces the number of features the model needs to process, speeding up the process faster than XGBoost while achieving similar performance [53].

LightGBM was selected due to its speed and memory efficiency compared to XGBoost and Random Forest. This models use of EFB and GOSS suggests that it is suitable for the dataset used in this project.

4.2.4 MLP Classifier (Neural network)

Neural networks, specifically MLP Classifier - a variant of neural networks, can effectively classify attacks within the KDD Cup dataset as they are designed to recognise complex patterns in data. The models consist of an input layer, a perceptron at the beginning of a neural network that receives the data. One or more hidden layers which process the data, and an output layer which is where the prediction is released from [54]. Neural networks use forward and backwards propagation to learn.

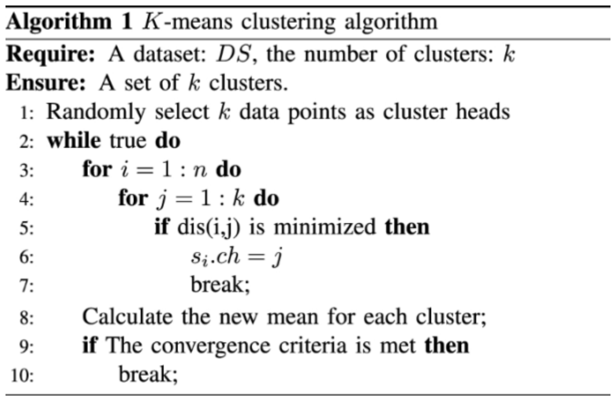
Forward propagation is where data is passed into the input layer; a linear transformation is then performed on the input data. This transformed data is then passed into a non-linear transformation to introduce non-linearity which allows the model to learn complex patterns in the data, this is done using an activation function like RELU or sigmoid [55]. Once this data has been processed through the layers, it is passed onto the output layer where a prediction is output. This prediction made by the neural network is then compared to the actual result. The difference between the two is calculated using the loss function, specifically the cross-entropy loss function for this project as classification is being used. Backwards propagation is then used to minimise the error,this process starts at the output layer and works backwards calculating how much each neuron contributed to the error and updates the weights as seemed appropriate.

These four algorithms were selected as they possess characteristics that align with the nature of the KDD Cup 99 dataset, which is large, complex and non-linear. Models such as MLP Classifier and tree-based models such as random forest are also among the most applied algorithms for KDDCup99-based IDS research [56]. Also, generally perform well in classification tasks, which is needed for effective signature-based detection.

4.3 Unsupervised Models

4.3.1 K-Means Clustering

K-Means clustering is a UL algorithm that groups data into K distinct clusters based on the similarity of data points. By definition, a data point in a cluster is more similar to points in the same cluster than to those in others. The algorithm then evaluates properties of clusters to make a prediction for each data point. K-Means is effective with network traffic data and the KDD Cup dataset because of the data's inherent structure. Normal traffic is frequent and shares similar features, allowing for K-Means to place most normal traffic into a single cluster. By analysing datapoints' distances to their cluster centre, as well as the size of their cluster, the K-Means algorithm can effectively isolate anomalous network traffic.



```
Algorithm 1 $K$-means clustering algorithm
Require: A dataset: $DS$, the number of clusters: $k$
Ensure: A set of $k$ clusters.
 1: Randomly select $k$ data points as cluster heads
 2: while true do
 3:     for $i = 1 : n$ do
 4:         for $j = 1 : k$ do
 5:             if dis(i,j) is minimized then
 6:                 $s_i.ch = j$
 7:                 break;
 8:     Calculate the new mean for each cluster;
 9:     if The convergence criteria is met then
10:         break;
```

Figure 17: K-Means Algorithm Pseudocode, [57]

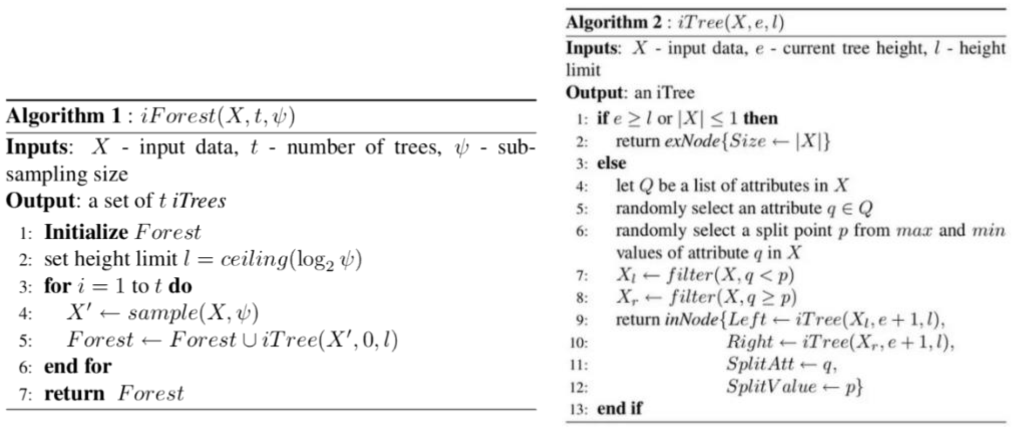
Shown in Figure 17, the K-Means algorithm first selects K random data points to be cluster centres. The algorithm then iteratively assigns a cluster to each data point, based on the closest cluster centre. The iterative process recalculates the clusters' centres as more data points join. The process stops if convergence occurs (minimal change) or iteration reaches a maximum count.

4.3.2 Isolation Forests

The isolation Forest model (iForest) is a UL technique that groups data so that outliers are easily identified by isolating them. An algorithm recursively partitions the data set into an iForest, a set of

Isolation Trees (iTrees), a graph type with edges and vertices. The algorithm gets path lengths for datapoints; by the number of edges traversed from the root node of its tree. For network IDS, the model assumes that anomalous traffic has shorter path lengths, due to having more sporadic features [58].

For creating a NIDS that uses the KDD Cup 1999 dataset for training its ML models, having the iForest model recursively partition the data gives it linear time complexity, making the fastest out of the selected intrusion detection algorithms. With network intrusions being 'few and different' compared to regular network traffic, iForest models can effectively isolate and identify anomalous network behaviours.



Algorithm 1 : $iForest(X, t, \psi)$

Inputs: X - input data, t - number of trees, ψ - sub-sampling size

Output: a set of t *iTrees*

```
1: Initialize $Forest$
2: set height limit $l = ceiling(\log_2 \psi)$
3: for $i = 1$ to $t$ do
4:     $X' \leftarrow sample(X, \psi)$
5:     $Forest \leftarrow Forest \cup iTree(X', 0, l)$
6: end for
7: return $Forest$
```

Algorithm 2 : $iTree(X, e, l)$

Inputs: X - input data, e - current tree height, l - height limit

Output: an iTree

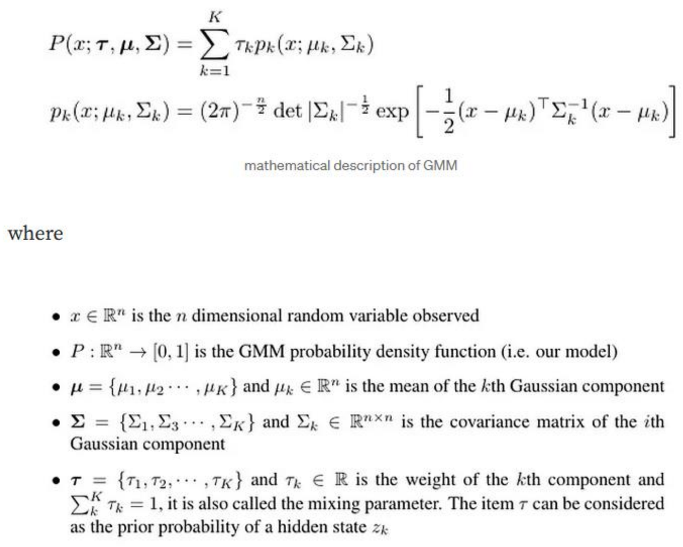
```
1: if $e \geq l$ or $|X| \leq 1$ then
2:     return exNode{$Size \leftarrow |X|$}
3: else
4:     let $Q$ be a list of attributes in $X$
5:     randomly select an attribute $q \in Q$
6:     randomly select a split point $p$ from $max$ and $min$
       values of attribute $q$ in $X$
7:     $X_l \leftarrow filter(X, q < p)$
8:     $X_r \leftarrow filter(X, q \geq p)$
9:     return inNode{$Left \leftarrow iTree(X_l, e + 1, l)$,
10:                  $Right \leftarrow iTree(X_r, e + 1, l)$,
11:                  $SplitAtt \leftarrow q$,
12:                  $SplitValue \leftarrow p$}
13: end if
```

Figure 18: iForest Basic Algorithm Pseudocode (Left) and iTree algorithm pseudocode (Right), [59]

Figure 18 shows how the simple iForest algorithm takes data and returns a set of iTrees (an iForest), which represents inherent patterns/structures of the dataset. Another algorithm instantiates iTrees by taking a sample data point and an iTree, it then selects a random feature and a random value within its range for the split point. A conditional statement determines how to branch the sample as a vertex, by evaluating the value of the randomly selected feature.

4.3.3 Gaussian Mixture Model

A Gaussian Mixture Model (GMM) is a probabilistic model that assumes all data points in the dataset come from a mixture of K Gaussian distributions. The model is a form of clustering, as the model segregates the dataset into K groups, also known as Gaussian components. GMMs groups data points based on their similarity. An Expectation-Maximisation (EM) algorithm estimates the composition of each Gaussian component, and a process can then assign data points to Gaussian components. GMMs perform well with NIDS for similar reasons to K-Means clustering. A key difference is that GMMs calculate the likelihood value for each data point, rather than making discrete assignments. The algorithm creates a probability distribution from the network traffic. Data points that have a low likelihood score are likely to be anomalous.



$$P(x;\boldsymbol{\tau},\boldsymbol{\mu},\boldsymbol{\Sigma}) = \sum_{k=1}^{K} \tau_k p_k(x;\mu_k,\Sigma_k)$$

$$p_k(x;\mu_k,\Sigma_k) = (2\pi)^{-\frac{n}{2}} \det|\Sigma_k|^{-\frac{1}{2}} \exp\left[-\frac{1}{2}(x-\mu_k)^\top \Sigma_k^{-1}(x-\mu_k)\right]$$

mathematical description of GMM

where

- $x \in \mathbb{R}^n$ is the n dimensional random variable observed
- $P : \mathbb{R}^n \to [0,1]$ is the GMM probability density function (i.e. our model)
- $\boldsymbol{\mu} = \{\mu_1, \mu_2 \cdots, \mu_K\}$ and $\mu_k \in \mathbb{R}^n$ is the mean of the kth Gaussian component
- $\boldsymbol{\Sigma} = \{\Sigma_1, \Sigma_3 \cdots, \Sigma_K\}$ and $\Sigma_k \in \mathbb{R}^{n\times n}$ is the covariance matrix of the ith Gaussian component
- $\boldsymbol{\tau} = \{\tau_1, \tau_2, \cdots, \tau_K\}$ and $\tau_k \in \mathbb{R}$ is the weight of the kth component and $\sum_k^K \tau_k = 1$, it is also called the mixing parameter. The item τ can be considered as the prior probability of a hidden state z_k

Figure 19: GMM Mathematical Representation, [60]

Compared to the previous two models, GMMS are more mathematically and computationally complex; they do not scale with high-dimensionality and categorical data. Figure 19 displays the mathematical representation of GMMs rather than pseudocode. The primary goal of the GMM algorithm is to find the parameters of the Gaussian components of the K Gaussian distributions that the data is assumed to be made of. These parameters are the Mean, the centre of each component. The Covariance, the shape and orientation, describe how datapoints are scattered about the mean. The Weight (or Mixing Probabilities), which explains how much that Gaussian distribution contributes to the overall mixture (set of Gaussian Distributions), describes the size and importance of each cluster.

4.4 Optimisation

At this stage, optimisation of the models was needed as a model's hyperparameters can influence its performance and can be why models do not overfit and underfit. The optimisation technique used was GridsearchCV, a widely used hyperparameter tuning method. GridsearchCV does an exhaustive search over a list of parameters. it goes through every combination possible until it finds the most optimal set of parameters that produce the best results.[61]

Cross-validation within the GridsearchCV is used to evaluate model performance for model selection and to tune hyperparameters. Within k-fold cross-validation, the training set is split into k folds, and each fold is used as a validation set for models trained on all the other folds. K is the number of folds within the cross-validation. An example of k-fold CV is shown in Figure 20.

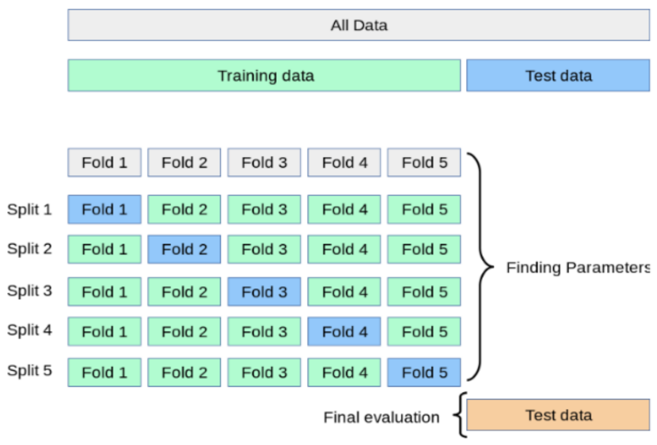


Figure 20: Example process of k-fold cross-validation where k = 5 [62]

For this project, GridsearchCV utilised stratified k-fold as a hyperparameter. Stratified k-fold cross-validation is a cross-validation variant that ensures each fold preserves the same percentage of samples from each class[63]. The models are less biased towards any class since each fold has similar distributions. This is crucial in highly imbalanced datasets such as the KDD Cup [64]. GridsearchCV was preferred over RandomsearchCV due to RandomsearchCV having less of a guarantee of returning the most optimal parameters, even though GridsearchCV has a higher computation time. Similarly, stratified-k-fold was chosen over standard k-fold cross validation as the standard k-fold cross validation can result in imbalanced class distributions across the folds, especially for already imbalanced datasets like the one used in this project. A stratified k-fold ensures that every fold has a representative class distribution.

Once the optimisation method was established, each model was tuned to a list of parameters estimated to be the optimal parameters for each model.

4.4.1 Supervised Optimisation

Table 2 below presents the optimal hyperparameters for each supervised model. It must be noted that “Random_state = 42” and parallelisation such as “n_jobs=-1” [65], which forces all core processors to be used by grid search, were used throughout for reproducibility and faster computation.

HYPERPARAMETERS	XGBOOST	RANDOMFOREST	LIGHTGBM	MLP CLASSIFIER
max_depth	**1**	**8**	**6**	X
n_estimators	**230**	**1200**	**300**	X
subsample	**0.7**	X	X	X
colsample_bytree/max_features	**0.7**	**0.7**	X	X
reg_lambda/lambda_L2	**10**	X	**2**	X
reg_alpha/alpha(MLP)	**0.1**	X	X	**0.1**
bootstrap	X	**True**	X	X
min_samples_leaf/min_child_samples	X	**1**	**5**	X
min_samples_split	X	**2**	X	X
Boosting_type	X	X	**Dart**	X
num_leaves	X	X	**20**	X
hidden_layer_sizes	X	X	X	**(128,64,32)**

activation	X	X	X	**RELU**
solver	X	X	X	**adam**
learning_rate_init	X	X	X	**0.01**
max_iter	X	X	X	**1000**
batch size	X	X	X	**128**
n_iter_no_change	X	X	X	**40**
early_stopping	X	X	X	**TRUE**
Learning_rate	X	X	**0.01**	**Adaptive**

Table 2: Optimal hyperparameters for the supervised models

The grid search found these models to be the most optimal hyperparameters for each supervised model.

4.4.2 Unsupervised Optimisation

For unsupervised models, these were the most optimal hyperparameters, as shown in Table 3 below.

HYPERPARAMETERS	K-means clustering	Isolation Forest	Gaussian mixture model
n_estimators	X	**100**	X
contamination	X	**0.21**	X
max_samples	X	**48**	X
max_features	X	**1.00**	X
variance threshold	**0.00**	**0.00**	**0.00**
PCA_components	**0.999**	**0.97**	**0.97**
K-clusters	**100**	X	X
max_iter	**100**	X	X
per_min_size	**99.5**	X	X
per_dist	**99.5**	X	X
w_distance	**0.01**	X	X
w_clusters	**0.85**	X	X
score_threshold	**0.64**	X	X
n_components	X	X	**18**

Table 3: Optimal hyperparameters for the unsupervised models

The hyperparameters shown in Table 3 were found to be the most optimal by the grid search. Hyperparameters, such as variance threshold, which is a feature selector that removes features under a certain threshold, and PCA components, which determines how many to retain, had similar or identical values throughout the three unsupervised models.

4.5 Results and Evaluation:

4.5.1 Evaluation metrics

At this stage, the models were evaluated, and their results were compared to conclude the best model for classifying intrusion detection on the KDD Cup dataset. The metrics used to come to this conclusion are presented.

Accuracy measures the total number of correctly classified cases divided by the number of cases. The formula for this metric is:

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

where:

- True Positives (TP) is the number of attack samples correctly identified by the model [66].
- True Negatives (TN) is the number of normal instances correctly predicted as normal [66].
- False Positives (FP) *is t*he number of normal samples classified as an attack [66].
- False Negative (FN) *is* the number of attack samples classified as normal [66].

Although accuracy can be a useful tool to measure the performance of the models, it can be misleading, especially in the case of extremely imbalanced datasets such as the KDD Cup dataset used in this project [67]. Therefore, more detailed metrics such as precision, recall and F1-score were used. Classification reports were also utilised to evaluate each class's precision, recall and F1 score within the dataset. This gives a more accurate assessment of the model's ability to classify attacks. The classification report returns scores ranging from 0.0 to 1.00 for each class on each metric in the report. An explanation of each metric and its equation is shown below.

Precision is calculated as:

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

Precision measures the accuracy of the model's positive predictions, showing if the model is classifying the correct class [68].

Recall is calculated as:

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

Recall measures how well the model performs in classifying the positive instances [68].

F1-score is calculated as:

$$F1\,Score = \frac{TP}{TP+\frac{1}{2}(FP+FN)},$$

F1 score is the balance between Recall and precision. It combines recall and precision into one value [68].

4.5.2 Supervised Results

The Macro average, which calculates how well a model performed on all classes overall, was used to compare the overall performance of each model. Below, the macro average results are shown in Table 4 for each model:

MODEL	ACCURACY	PRECISION	RECALL	F1-SCORE
XGBOOST	0.99	0.92	0.99	0.95
RANDOM FOREST	0.99	0.92	0.99	0.95
LIGHTGBM	0.99	0.92	0.99	0.95
MLP CLASSIFIER	0.93	0.86	0.71	0.74

Table 4: Macro-average results for supervised models

As shown in Table 4, the tree-based models achieved identical macro-average scores across all metrics: accuracy, precision, recall and f1-score. In comparison, the MLP classifier performed worse in all metrics, especially in precision, recall and F1 score. This is understandable as MLP classifiers are known to be hyper-sensitive towards hyperparameter tuning [69] and class imbalance compared to tree-based models, particularly ensemble tree-based models.

Although the tree-based models had identical macro-average scores as seen in table 4 , a deeper look into each model's classification reports revealed significant differences in each model's performance on the minority classes within the dataset. A look into these differences gives more information on which models can handle minority classes better, indicating which model would be most suitable for the network case study. After inspecting each individual classification report, it was identified that:

- The Random Forest had slightly better performance with recall in class 2 at 0.98, while XGBoost had a score of 0.96, a 0.02 difference
- The XGBOOST had a more consistent precision performance than Random Forest, staying at 0.84 for both minority classes 2 and 3, while Random Forest had 0.81 for class 2 but 0.87 for class 3.They both had 1.00 for class 0 and 1.

Overall, it was decided that XGBOOST was the best-performing model on this dataset, as both random forest and XGBoost had almost perfect recall, with only a 0.02 difference; however, there was more consistent performance for precision in classes 2 and 3 by XGBoost. The Classification report of the XGBoost is shown below in Table 5:

XGBOOST	PRECISION	RECALL	F1-SCORE	support
Class 0	1.00	1.00	1.00	223298
Class 1	1.00	0.96	0.98	60593
Class 2	0.84	1.00	0.91	2377
Class 3	0.84	0.99	0.91	5993
ACCURACY			0.99	292261
macro avg	0.92	0.99	0.95	292261
weighted avg	0.99	0.99	0.99	2922261

Table 5: Classification report of XGBoost

A confusion matrix was then deployed to examine the best performing model, XGboost's, performance more deeply. A confusion matrix gives an intricate view into how each label was classified, whether they were classified to the correct class or misclassified to another class. This can provide information on class overlap. Figure 21 shows the confusion matrix of the XGBoost.

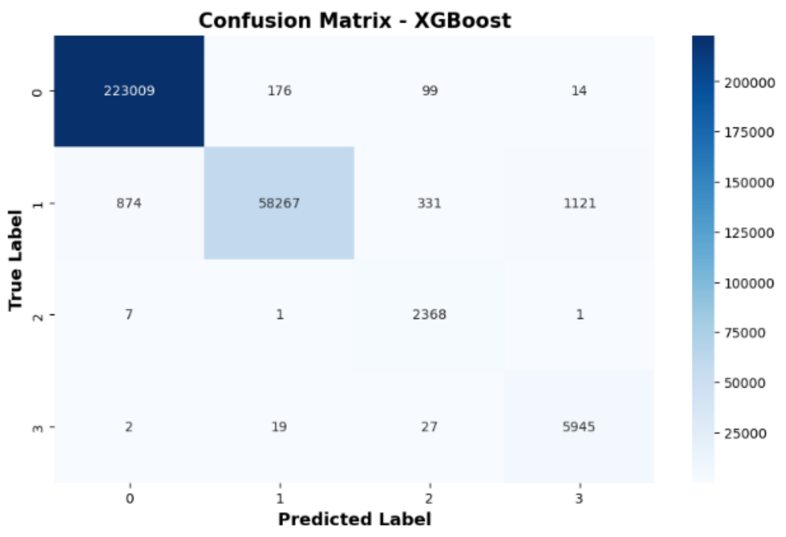


Figure 21: Confusion Matrix of XGBoost

The confusion matrix confirms the findings in the classification report. It shows that most classes are being classified correctly, suggesting the model can distinguish between classes very well. Minimal misclassification relative to the number of samples for each class is also observed.

Comparing the supervised model's performance to previous research, the authors of "A Robust Comparison of the KDDCup99 and NSL-KDD IoT Network Intrusion Detection Datasets Through

Various Machine Learning Algorithms” [66] got precision, recall and F1-score for the random forest of 0.9987, 0.9084 and 0.9514. The random forest in this project had better recall and similar F1-score performance, but worse precision performance at 0.92.

The authors of “Augmenting Network Intrusion Detection System using Extreme Gradient Boosting” [70] had an XGBoost performance of 0.93, 0.84, 0.88 and an MLP classifier performance of 0.85, 0.72, 0.78 for precision, recall and F1-score, respectively. The project's MLP classifier performs better in precision but worse in F1 score and recall compared to the MLP classifier in the research paper [70]. The XGBoost within this project performs better overall than.

For LightGBM, the authors of “An efficient Intrusion Detection Approach using Light Gradient Boosting” [71] got performance of 0.989, 0.974, 0.982 for precision, recall and F1-score. The LightGBM in the research paper has better precision and F1 score but lower recall compared to the LightGBM proposed in this project.

Although the models within this project do not have an overall better performance than previous research it must be noted that the authors of these research papers [66, 70, 71] all used binary classification with one [71] additionally using the NSL-kdd dataset, which is an improved version of the original KDD Cup 1999 used for this project. Multi-classification is a much more challenging task than binary classification; therefore, the performance of the models within this project, being similar and sometimes better in certain metrics, highlights the effectiveness of the feature selection, sampling and optimisation techniques used throughout this project.

4.5.3 Unsupervised Results

For the unsupervised models, the attack classes were aggregated into one single attack class to have a normal class(0) and an attack class(1). Furthermore, the Receiver Operating Characteristics - Area Under the Curve (ROC-AUC) scores were also evaluated for these models to show how well each model can make a distinction between normal(non-anomaly) and attack(anomalies) classes.

These are the results of the macro averages of the three unsupervised models shown in Table 6:

Models	Accuracy	Precision	Recall	F1-Score	AUC
GMM	0.75	0.69	0.74	0.70	0.79
K-Means	0.93	0.89	0.95	0.91	0.95
I-Forest	0.80	0.74	0.79	0.76	0.79

Table 6: Macro averages of unsupervised models

Table 6 illustrates that the best-performing unsupervised model was the K-means model, which performed significantly better in all metrics compared to I-Forest and GMM. For example, the F1-score of K-means is 0.91, compared to 0.76 and 0.70 for I-forest and GMM, respectively. This is due to the distinct boundary made by the classes being split between normal and attack; this type of distinct boundary is more ideal for the K-means model than GMM and I-Forest. The K-means clustering has an ROC-AUC score of 0.95, indicating excellent classification performance. In contrast, the I-Forest and GMM have ROC-AUC scores of 0.79, which indicates these two models perform relatively well.

A visual representation of the best-performing model's ability to distinguish between anomalies and non-anomalies is shown in Figure 22 as an ROC curve and Figure 23 as a confusion matrix:

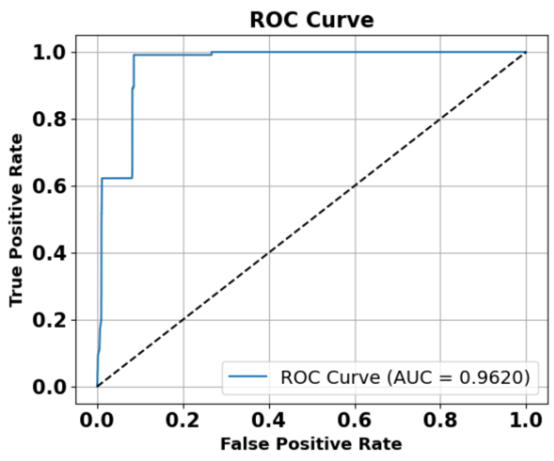


Figure 22: ROC Curve of K-means clustering algorithm

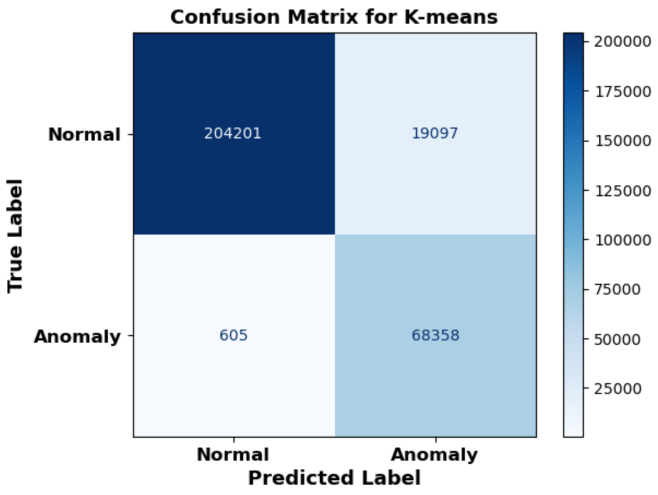


Figure 23: Confusion Matrix of K-means clustering algorithm

Comparing the unsupervised models to prior research, for the I-forest, the authors of "Evaluation of Unsupervised Anomaly Detection in Structured API Logs" [72] used the HTTP-CSIC 2010 dataset. It had a GMM performance of Precision: 0.66, Recall: 0.79, F1-score: 0.71, AUC: 0.83 and Accuracy: 0.74, while the GMM in this project had a similar performance of Precision: 0.69, Recall: 0.74, F1-score: 0.70, AUC: 0.79 and Accuracy: 0.75.

They also had a K-means performance of Precision: 0.56, Recall: 0.65, F1-score: 0.60, AUC: 0.77, Accuracy: 0.65. In contrast, the K-means within this project had much better performance at Precision: 0.89, Recall: 0.95, F1score: 0.91, AUC: 0.95, Accuracy: 0.93, suggesting that the techniques used in the feature selection and optimisation stages were effective.

For the I-forest model, they achieved a score of Precision: 0.67, Recall: 0.80, F1-score: 0.73, AUC: 0.83, and accuracy: 0.75. The I-forest model used in this project achieved scores: Precision: 0.74, Recall: 0.79, F1-Score: 0.76, AUC: 0.79, Accuracy: 0.80, similar to scores seen in the author's paper.

However, it must be noted that different Datasets were used as this project involved the KDDCUP99 dataset, which is more suited for network-based anomaly detection [72]. In contrast, the HTTPS-CSIC 2010 dataset used by the authors is more suited for web-based anomaly detection [72]. However, despite this, the scores between the models on each dataset are quite similar, indicating that these models have great generalisability.

5 Virtual Environment Set-up (AW)

5.1 Problem Overview

For the IDS to read network traffic and make predictions, a Local Area Network (LAN) is required. The technical and legal barriers to using a real NHS network make it unfeasible for this group project. Using physical devices to construct a home lab is also unfeasible due to pricing and requiring people to be physically present. Instead, the group focused on creating a Virtual Environment (VE) that can imitate a real NHS hospital LAN network environment. However, VEs also utilise hardware, although group member's laptops were deemed sufficient. Laptops were the better choice over the university's laboratory computers due to their portability, a beneficial ability to manage workloads and reduce risks associated with members occasionally being off campus.

Due to the poorer performance of an average laptop, it was critical to design the VE to use minimal CPU and RAM. A typical laptop has a ~3.0GHz multi-core CPU and ~8GB RAM. Having an existing dataset to train the ML algorithms - and then loading them onto the IDS means that the virtual environment does not need to be capable of producing massive amounts of traffic over extended periods.

Members researched Cloud-based approaches for the VE, using services such as Amazon Web Services (AWS). AWS provides support for hosting network infrastructure and the IDS. Amazon offers a variety of tools for AWS users, from Virtual Machine (VM) instances to evaluation software. However, research found that providers have strict terms of service for developing networks, particularly when performing cyberattack tests. Additionally, there were concerns about costs based on performance, such as limiting penetration testing.

Regarding simulation-based approaches for the VE, many resources are available that detail topologies, devices, and strategies for mimicking a real network environment. Cisco published a network architecture blueprint for the NHS [73]. The report listed standards and practices for implementing Information and Communications Technology and highlighted unique challenges and goals for such systems. Most network simulation software can run on average hardware and is safe to launch cyberattacks, with the LAN environment isolated from the host machine.

To summarise, fulfilling the project's aims and objectives requires a network environment for deploying the IDS. The VE must be able to represent common-place networking devices, such as switches and routers, and present a safe environment for cyberattacks. Members have specified that the VE uses low hardware to maximise work efficiency by running on typical PCS or laptops.

5.2 Required Tools Selection

5.2.1 Network Simulation Software

For hosting the VE, Network Simulation Software (NSS) can design, build, and test complex network topologies. Most NSSs enable users to implement real networking OS, such as Cisco IOS devices, Ubuntu servers, and Kali Linux-based attackers.

The group can use the NSS of choice to recreate the initial LAN design in a lab environment. The software can properly monitor devices, packet routing, and protocol usage for further understanding of the normal behaviour of the LAN [73]. At this point, adjustments to the original topology may be made to suit simulated behaviour and performance requirements better. With a sufficient base network infrastructure design created for the virtual environment (covered in section 2.2.3), the implementations of the attacker and IDS can begin.

The three best-received and most used network simulation software for creating a virtual environment of the initially created healthcare LAN design are:

- **Cisco Packet Tracker:** A cross-platform network simulation software tool developed by Cisco. It enables users to visualise, design and experiment with real network topologies and protocols in a virtual environment. Unfortunately, Packet Tracer uses only Cisco devices, restricting the usable OS. Additionally, the software is challenging to access as it is only available to Cisco certification students.
- **GNS3:** A free, open-source networking tool which supports the visualisation, design, and experimentation of real network topologies. Software images of real networking OSs are readily available, such as Cisco IOS and Juniper JunOS. Virtualisation support with VirtualBox or VMware allows virtual machines of any operating system to work in the simulated topologies.
- **Virtual Box / VMWare Workstation:** Both software are free hypervisors that enable the host computer to run and manage one or more virtual machines. The flexible programs enable each VM to run different operating systems, allowing for individual configuration. Hypervisors are not proper NSS, with no topology builder or visualiser, they enable the connection of multiple VMs to represent clients, servers, attackers, and routers.

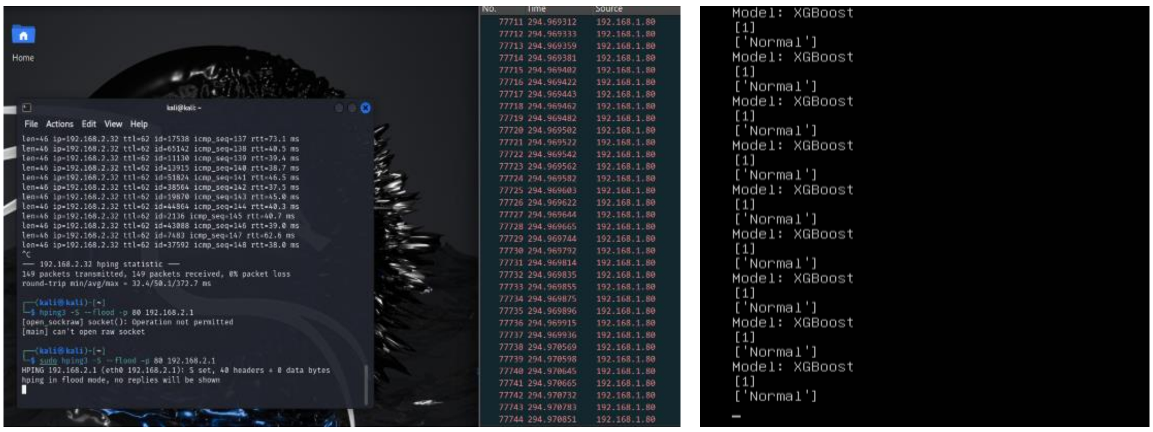
After researching and weighing various NSS for creating the virtual environment, the group decided to use GNS3. Without the restraint of only using Cisco IOS devices that come with CPT, as well as the ease of downloading software and required image files, GNS3 is ideal. GNS3's built-in support for hypervisors helps offload the inflexibility and heavier performance when using them alone. GNS3 can intuitively place the VMs from the hypervisors into the visualised topology, enabling VMs that are easy to understand and work with.

5.2.2 Attacker and IDS Virtual Machines

Implementing the simulation with GNS3, the attacker and IDS systems would be best hosted on individual VMs, which provide maximum performance, functionality, and individual configuration. The official website explains to use GNS3 with VMware Workstation Pro, over VirtualBox for hosting the simulation itself and any other VMs.

VMware gives extensive control for creating VMs, which are easy to import into a GNS3 project. GNS3 can then configure the VM devices to interact with the rest of the network topology. All operating systems, such as Windows, Mac, Ubuntu, Alpine, and Kali Linux, are feasible.

For the attacker VM, online resources point to Kali Linux [74]. It is a Debian-based Linux distribution designed specifically for penetration testing, ethical hacking, and security research. Kali comes packaged with hundreds of attack tools, including Nmap, Wireshark, and Hydra, covering all the attack categories from the KDD'99 dataset. For proper evaluation of the IDS, the VE must be able to generate both normal and attack traffic. Figure 24 shows a Kali Linux VM successfully connected to the VE and launching a DoS attack, and the XGBoost model loaded onto the VE.



```
len=46 ip=192.168.2.32 ttl=62 id=17538 icmp_seq=137 rtt=73.1 ms
len=46 ip=192.168.2.32 ttl=62 id=65142 icmp_seq=138 rtt=40.5 ms
len=46 ip=192.168.2.32 ttl=62 id=11130 icmp_seq=139 rtt=39.4 ms
len=46 ip=192.168.2.32 ttl=62 id=13915 icmp_seq=140 rtt=38.7 ms
len=46 ip=192.168.2.32 ttl=62 id=51824 icmp_seq=141 rtt=46.5 ms
len=46 ip=192.168.2.32 ttl=62 id=38564 icmp_seq=142 rtt=37.5 ms
len=46 ip=192.168.2.32 ttl=62 id=19870 icmp_seq=143 rtt=45.0 ms
len=46 ip=192.168.2.32 ttl=62 id=44864 icmp_seq=144 rtt=40.3 ms
len=46 ip=192.168.2.32 ttl=62 id=2136 icmp_seq=145 rtt=40.7 ms
len=46 ip=192.168.2.32 ttl=62 id=43888 icmp_seq=146 rtt=39.0 ms
len=46 ip=192.168.2.32 ttl=62 id=7483 icmp_seq=147 rtt=62.6 ms
len=46 ip=192.168.2.32 ttl=62 id=37592 icmp_seq=148 rtt=38.0 ms
^C
--- 192.168.2.32 hping statistic ---
149 packets transmitted, 149 packets received, 0% packet loss
round-trip min/avg/max = 32.4/50.1/372.7 ms

┌──(kali㉿kali)-[~]
└─$ hping3 -S --flood -p 80 192.168.2.1
[open_sockraw] socket(): Operation not permitted
[main] can't open raw socket

┌──(kali㉿kali)-[~]
└─$ sudo hping3 -S --flood -p 80 192.168.2.1
HPING 192.168.2.1 (eth0 192.168.2.1): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
```

```
No.   Time        Source
77711 294.969312  192.168.1.80
77712 294.969333  192.168.1.80
77713 294.969359  192.168.1.80
77714 294.969381  192.168.1.80
77715 294.969402  192.168.1.80
77716 294.969422  192.168.1.80
77717 294.969443  192.168.1.80
77718 294.969462  192.168.1.80
77719 294.969482  192.168.1.80
77720 294.969502  192.168.1.80
77721 294.969522  192.168.1.80
77722 294.969542  192.168.1.80
77723 294.969562  192.168.1.80
77724 294.969582  192.168.1.80
77725 294.969603  192.168.1.80
77726 294.969622  192.168.1.80
77727 294.969644  192.168.1.80
77728 294.969665  192.168.1.80
77729 294.969744  192.168.1.80
77730 294.969792  192.168.1.80
77731 294.969814  192.168.1.80
77732 294.969835  192.168.1.80
77733 294.969855  192.168.1.80
77734 294.969875  192.168.1.80
77735 294.969896  192.168.1.80
77736 294.969915  192.168.1.80
77737 294.969936  192.168.1.80
77738 294.970569  192.168.1.80
77739 294.970598  192.168.1.80
77740 294.970645  192.168.1.80
77741 294.970665  192.168.1.80
77742 294.970732  192.168.1.80
77743 294.970783  192.168.1.80
77744 294.970851  192.168.1.80
```

```
Model: XGBoost
[1]
['Normal']
Model: XGBoost
[1]
['Normal']
Model: XGBoost
[1]
['Normal']
Model: XGBoost
[1]
['Normal']
Model: XGBoost
[1]
['Normal']
Model: XGBoost
[1]
['Normal']
Model: XGBoost
[1]
['Normal']
Model: XGBoost
[1]
['Normal']
Model: XGBoost
[1]
['Normal']
_
```

Figure 24: The Ubuntu Server VM running the IDS script, detecting traffic in the virtual environment with the model 'XGBoost' loaded.

A VM is ideal for hosting the IDS, for easily deploying scripts made by group members, and for having full control over it. However, a drawback is that constant modification is required, such as simulation parameters and hardware usage. After reviewing all potential distributions, the group finalised on utilising an Ubuntu server VM to host the IDS. At first, an ultra-lightweight Linux distribution (Alpine/Linux Mint) was the best choice; however, it required too much time and work to deploy the scripts error-free. The minimal distributions lack most required packages and do not provide support for newer Python versions that scripts utilise.

The Ubuntu server has no graphical interface by default, meaning it uses fewer resources, particularly CPU and RAM. Its lightweight makes it more stable for long sessions of network traffic analysis. It comes preinstalled with Python3 and includes easy package management tools. The extensive documentation for Ubuntu makes managing errors and adding extra tools straightforward. Figure 24 displays the text-based environment of the Ubuntu Server and how the IDS program can display predictions in the virtual environment.

5.2.3 Other Software and GNS3 Devices

The core component of creating a realistic network environment comes with the routers used. Fortunately, GNS3 utilizes Cisco IOS images for real router emulation, which enables realistic implementation of subnets, routing protocols and port connections. The final virtual environment uses c7200 routers for the public and private-facing edge routers.

With the assumption of network attacks coming from the public side, targeting devices in the private sub-network, the final solution has the IDS connected between the two routers using an Open vSwitch. Open vSwitch is a fully software-based representation of an advanced switch; it has much less RAM and CPU usage than Cisco's IOSvL2, which imitates a physical switch device. Port mirroring, an advanced switch feature, had to be utilised so that the IDS could properly receive traffic going between the routers.

GNS3 also features various Docker containers for commonly used network devices. Docker containers are lightweight, executable packages that utilise minimal hardware for running their application. The virtual environment contains multiple Firefox docker containers representing clients or hospital workstations and can simulate realistic HTTP/HTTPS traffic for the IDS to monitor.

The remaining end-user devices seen in the GNS3 simulated topology are Virtual PCs (VPCs). Built into GNS3, VPCs are tiny programs that use 2MB of RAM per instance and can send pings and connect via telnet to simulate basic TCP/UDP client traffic.

5.3 Topology Construction

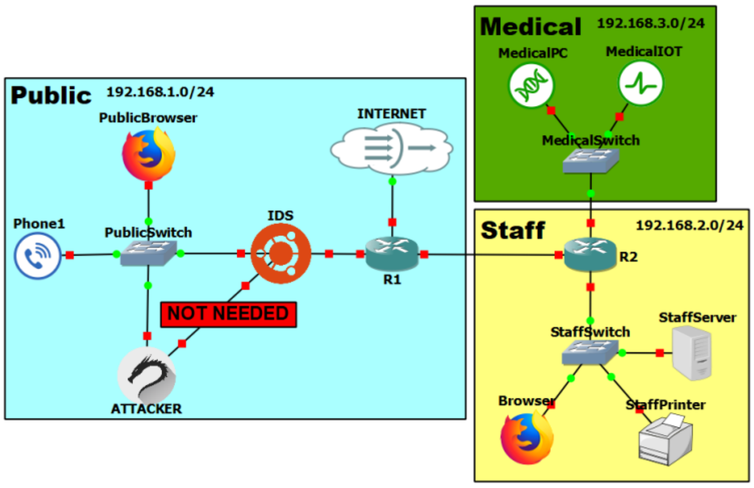


Figure 25: Simulated network topology visualisation in GNS3, with fully functioning devices.

Understanding all the devices to use and how they would interact, the full topology can be implemented in GNS3, as shown in Figure 25. Compared to initial designs (discussed in sections 2.2), the simulation uses fewer routers for performance maximisation, while still behaving like a realistic LAN [75]. All end devices in each subnet connect to a dedicated switch that connects to a router; this mitigates the problem of limited ports of real router devices, improving performance and scalability. Another change was to connect the server end-device to only the staff subnet. Connecting it to only one subnet reduces the number of devices while maintaining its security. In most networks, the server device is a key target for attacks, containing valuable data or being able to cause destruction to the whole network.

One limitation with GNS3 is that it does not provide devices for Wireless Access Points (WAP), instead assuming that Ethernet connections are similar enough. With the omission of WAPs, the IDS solution is simpler, only focusing on wired network traffic and routing logic for intrusion detection. While it is possible to create a sub-network with VPCS and Docker containers that emulate mobile phones and wireless devices with a specialised router, the time and work required make it unfeasible.

The topology visualisation in Figure 25 displays all the devices in the final topology. When a device is on (indicated by a red/green symbol), GNS3 can open a console for it, starting the software depending on the device. A VPC will open a simple terminal, whereas the attacker or IDS will open the VM in VMware. In VMware, the virtual machine behaves as another computer, with an interactive desktop and a command prompt for executing commands.

It is important to understand how the IDS functions as an individual device in the network. The IDS VM is an inline bridge, allowing network traffic to seamlessly pass through its two ports. The two ports do not have IP addresses, meaning that the IDS device does not interfere with the network in any way. This implementation allows for the IDS device to be placed anywhere. Figure 4 shows the IDS placed between the public switch and the first Cisco router, and with a third optional port for communicating with the Kali Linux VM to attack the network for testing.

6 Intrusion Detection System (BU)

The IDS consists of four main components, detailed below.

6.1 KDD’99 Feature Extractor

For the IDS to classify real-time traffic in the simulation, this traffic must be continually captured and transformed into the KDD’99 format. Thus, one of the crucial components of the IDS is the KDD’99 Feature Extractor [76], a utility created by GitHub users *bittomix* and *zhiyuan-liao*.

When run, the compiled executable sniffs network traffic and uses the captured packets to construct connection records. Each record contains 28 of the 41 KDD’99 features; the 13 excluded features are the so-called “content features”, which are gathered from the data sections of captured packets. While this is perhaps a limitation of the extractor, it can be argued that since most connections made over a network in the modern day are encrypted, it would be infeasible to construct such features in the vast majority of cases. Therefore, it is advantageous to train an IDS not to rely on content-based features.

Once constructed, the connection records are printed to stdout (Linux’s standard output stream). The extractor will continue sniffing packets and constructing connection records indefinitely until it is stopped.

6.2 Machine Learning Models

Before being applied in the simulation, the ML models (described in Section 4) first had to be transferred to the IDS with their trained parameters intact. This was achieved using the Python library Joblib [77], which enables the exporting and importing of Python objects such as trained scikit-learn models.

Each trained model, encoder and scaler was stored in compressed format, ready to be used by the IDS.

6.3 Classifier Scripts

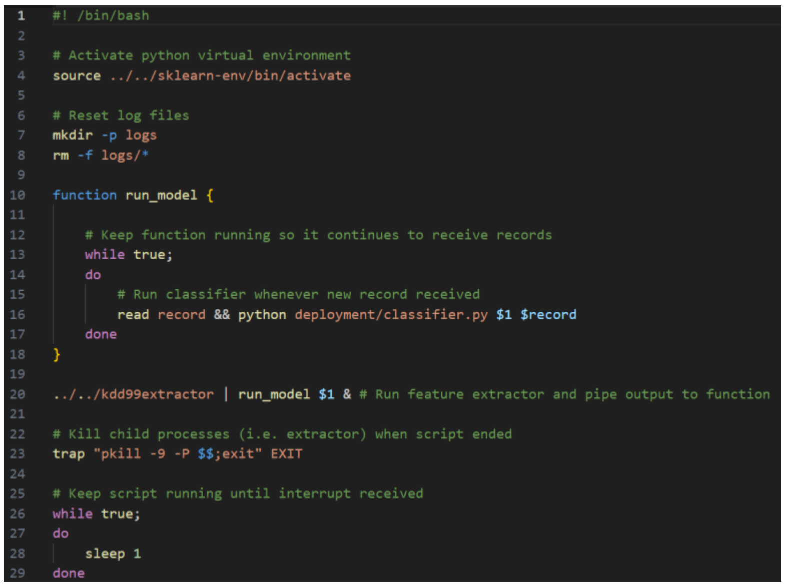
The classification of a given connection record (i.e. datapoint) captured in the simulation is handled by two Python scripts: a generic script, *classifier.py*, and a helper script specific to the ML model being used.

Classifier.py takes the name of the selected model and a connection record as command-line arguments and converts the record into a Pandas dataframe, whose format matches that used to train the model. The model-specific helper script is used to ensure that the correct preprocessing is applied, as well as to load the specified model and use it to predict the connection record’s class.

Once the prediction has been made, *classifier.py* writes the record to one of four log files based on its classification: *normal.txt*, *dos.txt*, *r2l.txt* and *probing.txt*.

6.4 Bash Script

The IDS is operated by running a Bash script, shown in Figure 26. After activating the Python virtual environment and clearing any previous log files, the script runs the KDD’99 Feature Extractor in the background, piping its output to the function *run_model* instead of stdout.



```bash
#! /bin/bash

# Activate python virtual environment
source ../../sklearn-env/bin/activate

# Reset log files
mkdir -p logs
rm -f logs/*

function run_model {

    # Keep function running so it continues to receive records
    while true;
    do
        # Run classifier whenever new record received
        read record && python deployment/classifier.py $1 $record
    done
}

../../kdd99extractor | run_model $1 & # Run feature extractor and pipe output to function

# Kill child processes (i.e. extractor) when script ended
trap "pkill -9 -P $$;exit" EXIT

# Keep script running until interrupt received
while true;
do
    sleep 1
done
```

Figure 26: Bash Script run by IDS

7 Simulation Performance (YH, BU, KV, AW)

7.1 *Testing Framework*

7.1.1 *Test Bash Script*

To benchmark the performance of each loadable ML model for the IDS, an evaluation script, *test.sh*, forms an OpenSSH connection with the Kali Linux VM in GNS3. The test script runs the attacks extensively, taking around 40 minutes to complete. During its course, the script exposes the VE to traffic that matches the dataset's four distinct traffic categories: Normal, DoS, R2L, and Probing.

Table 7 below describes every command used in the test framework and explains attacks from the KDD Cup 1999 dataset that they resemble. Eight distinct commands, each running three times, result in 24 predictions for the test script to record. After each command has run its course, there is a 100-second window for the IDS to detect and classify connections. The long duration allows the KDD Cup extractor to turn raw traffic into KDD-style feature records. Arriving packets do not instantly create logs; the window ensures that the extractor fully processes and records all packets - even from heavy loads.

Command	**Type**	**Description**
`curl -s -head http://example.com`	Normal	A HTTP HEAD request (only headers, no body). Tests the detection of minimal web traffic with no payload.
`curl google.com`	Normal	A standard HTTP GET, emulating full web fetches with TCP handshakes and data transfer.
`hping3 -S -p 80 -i u100000 -c 200 --rand-source 192.168.2.1`	DoS	Rapidly sends 200 TCP SYN packets at port 80 of the staff router with spoofed source IPs.
`ping -i 0.2 -s 1400 -c 100 192.168.2.1`	DoS	Sends 100 large payload ICMP echo requests every 0.2 sec. The ICMP flood resembles the Smurf attack.

`echo "anonymous"; sleep 1;`
`echo "test@test.com"; sleep 1;`
`echo "put test.txt"; sleep 1;`
`echo "quit"`
`| ftp -n 192.168.2.1`
	R2L	Performs an anonymous FTP login and file upload to the staff router. This is intended to match the attack ftp_write.

`for port in 21 23 25 111 135`
`139 445 512 513 514 1099; do`
`  nc -vz -w 1 192.168.2.1 $port`
`done`
	R2L	Scans standard authentication ports for potential weak-credential exploits. Matches Warezclient/Warezmaster with unauthorized file-retrieval attempts.

`nmap --top-ports 100`
`192.168.2.1`
	Probing	Runs an Nmap scan on the 100 most common ports of the target device. The floods of network port-scan patterns resemble the portsweep, nmap, and satan attacks.

`nmap -sN 192.168.2.1`	Probing	Performs a TCP "null" scan (no flags set). The stealthier scan is designed to evade simple signature checks. It covers the same attacks as the last command.

Table 7: Commands used in the test framework, explaining similar attacks from the KDD Cup dataset

While the traffic is being simulated, the IDS collects, predicts and logs connection records. To evaluate its performance, the IDS's predictions must be compared to the correct classes for the simulated traffic.

Whereas labelling predictions as *True Positive, False Positive, True Negative,* or *False Negative* is straightforward when using the KDD'99 test set, this is somewhat more complicated when it comes to the simulation. When running a particular attack, normal traffic may get mixed up amongst the attack traffic. Therefore, rather than having a clear, correct label to compare to for each connection record, it is unclear which records captured within the 100-second window represent normal traffic, and which ones are associated with the attack; all that is known is that a particular type of attack has been launched, and that the IDS ought to have detected it. In light of this, the test script defines a prediction for any of the 3 attack classes (*DoS*, *R2L* or *Probing*) to be positive if at least 1 connection record captured during the 100-second window is predicted as belonging to that class. On the other hand, a prediction for the *Normal* class is defined as being positive if no connection records are predicted as belonging to any of the attack classes within that same window.

Using these definitions, *test.sh* tallies up the true/false positives and negatives for each of the 4 classes, which are then used to calculate precision, recall and F1 scores for each class, as well as an overall accuracy score. These metrics form a report which is saved as a text file for analysis.

7.1.2 Integrating Other Intrusion Detection Systems

In the 2024 paper "Expectations versus reality: evaluating intrusion detection systems in practice" [7], Hesford J, Cheng D, Wan A, Huynh L, Kim S, and Kim describe creating a test framework for multiple commonly used IDS. With the flexible simulation environment created by the group, it is possible to use GNS3 and VMware to load other IDS that can read the same network traffic. The VE allows the deployment of various IDS used in the field, including Deep Neural Network, Kitsune, HELAD, and Suricata [7]. These IDS can generate the same performance metrics as the group's created IDS with its loaded model.

A significant challenge in creating a test framework for multiple IDS is that they must be derived from the same dataset [7]. Datasets more recent than KDD'99 have entirely different methods of generating and reading pcap files for record extraction. Due to the work required to create such a framework, later sections will only discuss performance metrics generated by the group's IDS and make no comparisons with other IDS.

7.2 System Performance

The Windows 11 device that hosted every Virtual Machine, including the primary GNS3 VM, has an 11th-generation i5-1135G7 CPU running at 2.40 GHz and 7.6 GB of RAM running at 3200MHz. Throughout the 40-minute duration of each test, the system peaks at ~%80% usage for both CPU and RAM. The IDS makes a minimal contribution to the usage, using <1MB of RAM. The pressure on hardware primarily comes from the attacks that Kali Linux runs. By not overly straining the host hardware, the performance of the IDS and loaded ML models did not suffer from hardware bottlenecks.

The KDD'99 feature extraction primarily determines how well the IDS reads network traffic in the simulation. One factor is the alert lag, defined as how long it takes the IDS to read a malicious connection record after an attack starts firing packets. Certain features from the KDD'99 dataset require the extractor to have a buffer, needing a time window to represent the feature in a record.

7.3 Model performance in simulation

7.3.1 Supervised results

Each supervised model was integrated into the IDS for testing in simulation. The tests use 24 connections per class, which are then identified as one of 4 count types: True Positive (TP), False Positive (FP), True Negative (TN) and False Negative (FN). These classes are the three attack categories (DoS, R2L, Probing) and normal traffic.

These are the same count types used to calculate the result for the KDD test dataset. Section 4.5 goes into detail regarding the definitions of each type.

The values for each classification for each model are presented below in Figure 27.

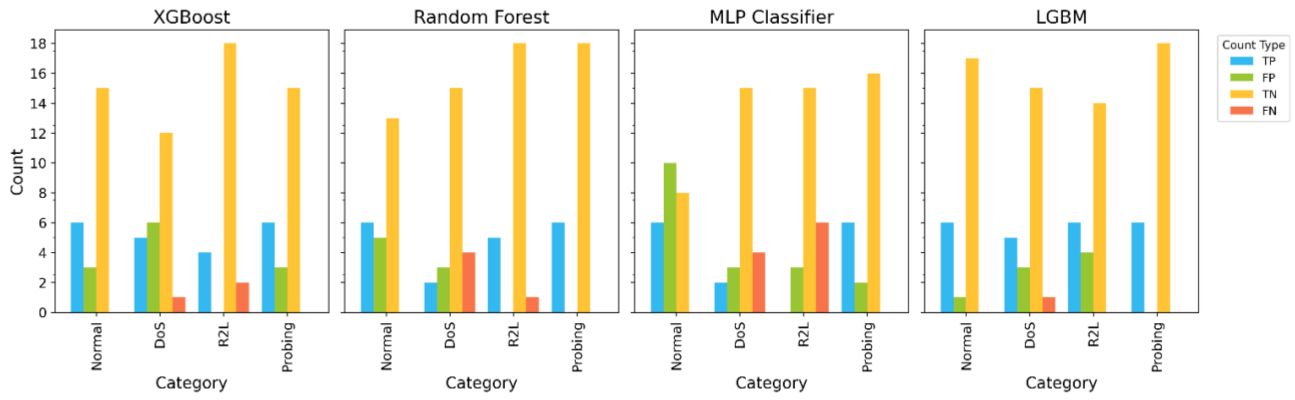


Figure 27: Results for each models' prediction values for each class

These count types are then used to calculate accuracy, precision, recall, and F1-score, which evaluate a model's performance. Each of the models' metrics is presented below in Table 8:

MODEL	ACCURACY	PRECISION	RECALL	F1-SCORE
XGBOOST	0.64	0.70	0.86	0.75
RANDOM FOREST	0.70	0.74	0.79	0.74
LIGHTGBM	0.74	0.77	0.96	0.85
MLP CLASSIFIER	0.44	0.38	0.58	0.44

Table 8: Metric results for each supervised model

From Table 8, the metrics determined that the best-performing model from the simulation testing was LightGBM. Followed by Random Forest, XGBoost, and finally MLP Classifier.

As backed in Figure 27, LightGBM has the most true positives and negatives and the least true and false negatives out of all the models. This gives it the highest metrics out of any model.

However, the most consistently performing model is Random Forest, ranked second for both the KDDcup99 and simulation evaluations. XGBoost was the top-performing model from the test data results, and LightGBM was third. MLP Classifier was consistently ranked last in both rankings, resulting in it being the worst-performing model for this IDS.

So, the hypothesis that XGBoost is the best model to perform was correct for the test data results, but not for the simulation.

It is also evident that the four models' performance on the simulation had decreased compared to the performance on the KDD Cup 99 dataset. This is understandable as the models were tested in two different environments. The KDD Cup dataset was cleaned and pre-processed, and was originally collected on a military network. In contrast, the simulation in this project was based on a hospital network. The two cases are likely to have different infrastructures and traffic patterns. Additionally, the simulation contains more noise than the test dataset due to the simulation actively running normal traffic while testing the IDS's performance. Another reason for the drop in performance was that certain features were not contained in the feature extractor used. The feature extractor did not contain the content features of the KDD Cup dataset, particularly the "logged_in" feature, which was discovered to be crucial for the model's performance. Therefore, the absence of this feature in the simulation caused a reduction in the performance of the models.

7.3.2 Unsupervised results

Although the unsupervised models were prepared to be implemented into the IDS, these simulation results could not be collected due to time constraints and challenges with integrating the scripting.

Going forward, if more time were allowed with this project, unsupervised models could have been thoroughly tested, like the supervised, and it could have been determined if a signature or anomaly-based detection system would have been better for this network infrastructure.

7.3.3 Result Discussion

This study hypothesised that the best-performing supervised model overall would be XGBoost. This hypothesis was correct for test data evaluation but not for simulation results. However, this study cannot prove the hypothesis that supervised models are more efficient at attack detection than unsupervised models.

In the end, the most effective algorithm for this network infrastructure was found with high metrics for detecting different types of attacks and normal traffic. LightGBM has low false positives, meaning it will rarely give false alarms on the system, and it has low false negatives, meaning that it will rarely miss network attacks. This makes it an ideal intrusion algorithm for an IDS fitted to this case study system.

8 Related Works (YH, AW)

Deployable Network Intrusion Systems have been studied and developed since the early 1990s. In a 1987 IEEE study, Dorothy E. Denning [78] describes a general-purpose Intrusion Detection Expert System (IDES) for real-time attack detection. The IDES model is independent of any system and profiles records generated by a target system to flag anomalous ones. ML-based IDS also has studies dating back to the 1990s using DARPA datasets (including precursors to KDD Cup 1999). A 2004 study by Mukkamala, Sung, and Abraham [79] details the significance of KDD'99 ensemble methods for an effective IDS. The paper describes a similar dataset preprocessing stage, such as data imbalances and feature importance. Additionally, their ML algorithms are comparable to what group members researched and developed, such as SVMs and neural networks. Additionally, author Hanane Chliah [80] in a 2025 study utilised sampling techniques such as Smote-Tomek to balance their unbalanced dataset; similar techniques were used in this project regarding the imbalance within the Kddcup99 dataset.

9 Practical Considerations and Further Study (YH, ZW, AW)

For further work on this project, there were a few things. Firstly, while the KDD Cup 1999 dataset is one of the most cited and researched IDS-based datasets, it is 26 years old, so some attacks in the dataset may be outdated. Additionally, the KDD Cup dataset includes numerous redundant records and has an extreme imbalance between classes. Therefore, in future work, it would be more appropriate to use the new and improved version of the KDD Cup, the NSL-KDD, which builds upon the limitations and failures of the KDD Cup dataset. Unfortunately, due to time constraints, the unsupervised models were not tested on the simulation, as the IDS would need extra configuration to handle binary classification. Therefore, in the future, testing the three unsupervised models using binary classification in the IDS would be necessary. Furthermore, another improvement considered was to combine the best-performing supervised model and the best-performing unsupervised model into an ensemble to create a robust algorithm that can detect known attacks and anomalies.

Additionally, hardware usage is the primary limiting factor in utilising GNS3 for the virtual environment. Every implemented Virtual Machine requires one dedicated CPU core and 1024 MB of RAM to operate at a bare minimum level. The final implementation has the host laptop's quad-core CPU, 8 GB of RAM spread across three Virtual Machines, and the Windows 11 OS hosting it all. Therefore, the GNS3 VM only has one GB of RAM. Every non-VM device in the GNS3 network connects as part of the GNS3 VM; they require a specified amount of RAM and use up processing. Although they are essential network components, all practical firewall devices require significant RAM. The popular Cisco ASAv firewall device requires 2048 MB of RAM to simulate. For further work, a host device with more RAM can utilise more networking devices in GNS3 to better resemble real networks.

This research involves network security, but at the Internet protection and security level, it simply includes firewall-related software (such as PfSense). However, the specific configuration strategy of the firewall and the mechanism of working with the machine learning detection system are not explored in depth. Reasonable configuration of firewalls is crucial for security protection. If there is an improper firewall configuration during the project setting process, it may not be able to block external attacks effectively, and it may lack effective coordination with the machine learning-based detection system, making it difficult to form a complete security protection system, which in turn could cause the system to be unable to respond and handle complex network attacks in the real world quickly. In subsequent work, it is necessary to combine machine learning detection models to specify firewall configuration strategies in a targeted manner to achieve more efficient interception of common Internet attacks. Attempts to add firewalls to the virtual network introduced performance issues, consuming too much memory. This problem could be overcome in the future through the use of more powerful hardware or more optimised software to enable the firewall to run with a machine learning model.

10 Conclusion (YH, BU)

As cyberattacks continue to become more prevalent, it is more important than ever to explore methods of detecting and preventing them. This project explored the efficacy of machine learning techniques in detecting cyberattacks by incorporating them into a real-time intrusion detection system (IDS). In doing so, the project achieved its objectives of training multiple machine learning models on attacker behaviour data and evaluating their performance both on a test dataset and within a simulated network.

Following a literature review into network infrastructure frameworks for hospitals, a simplified topology was successfully designed and simulated using GNS3. The simulation included a Kali Linux virtual machine capable of launching cyberattacks on the network, and an Ubuntu Server virtual machine, on which to host the intrusion detection system. The final IDS implementation met its design goals, namely, to be capable of gathering network traffic information in real time and classifying it as either normal traffic or an attack. However, the system was limited by the fact that certain features could not be extracted, reducing the amount of information available to the machine learning models.

For the training of the machine learning models, the KDD Cup 1999 (KDD'99) dataset was selected due to its immediate relevance and regular citations in the field. Nonetheless, many problems were found with this dataset, such as extreme class imbalances and duplicate datapoints. These issues were largely mitigated in the data preprocessing stage, but are still likely to have somewhat limited the models' performance.

After successfully building and training seven different machine learning models on the KDD'99 dataset, XGBoost was found to perform the best on the corresponding test dataset, in line with the initial hypothesis. A significant limitation in this stage was the fact that the unsupervised models were not tested in the simulation due to time constraints. Of those tested, LightGBM was found to be the best-performing model in the simulation, while the Random Forest model had the most consistent performance between both the test dataset and the simulation. The findings indicate LightGBM was the most efficient classifying intrusion detection algorithm, but Random Forest is the most generalisable model; however, the few problems regarding the feature extractor and the simulation not containing firewalls might have slightly skewed results.

While the intrusion detection system developed in this project is merely a prototype tested in a simulation, it should be straightforward to deploy it into any network, real or simulated. Such a system could complement existing network security measures as an early warning system. However, the system's performance in an unknown environment is not guaranteed, so different models and datasets may perform better.

11 Acknowledgements

The authors would like to express their gratitude to the project supervisor, Dr Rasheed Hussain, for their expertise, support and guidance through this project's development.

References

[1] Uk cybercrime statistics 2023 | online security | twentyfour [Internet]. Available from: https://www.twenty-four.it/services/cyber-security-services/cyber-crime-prevention/cyber-crime-statistics-uk/

[2] Cyber security breaches survey 2024 [Internet]. GOV.UK. Available from: https://www.gov.uk/government/statistics/cyber-security-breaches-survey-2024/cyber-security-breaches-survey-2024

[3] What is network security [Internet]. Cisco. Available from: https://www.cisco.com/c/en_uk/products/security/what-is-network-security.html

[4] What is an intrusion detection system (Ids)? | ibm [Internet]. 2023. Available from: https://www.ibm.com/think/topics/intrusion-detection-system

[5] The pros & cons of intrusion detection systems | rapid7 blog [Internet]. Rapid7. 2017. Available from: https://old.rapid7.com/blog/post/2017/01/11/the-pros-cons-of-intrusion-detection-systems/

[6] Introduction to the cyber assessment framework [Internet]. Available from: https://www.ncsc.gov.uk/collection/cyber-assessment-framework/introduction-to-caf

[7] Hesford J, Cheng D, Wan A, Huynh L, Kim S, Kim H, et al. Expectations versus reality: evaluating intrusion detection systems in practice [Internet]. arXiv; 2024. Available from: http://arxiv.org/abs/2403.17458

[8] Cyber attack on hospitals impacts 1,130 operations in London [Internet]. BBC News. 2024. Available from: https://www.bbc.com/news/articles/c5114k2zg08o

[9] NHS cyber-attack: Boy, 14, has cancer operation postponed [Internet]. BBC News. 2024. Available from: https://www.bbc.com/news/articles/cv22yyljgw4o

[10] NHS England confirm patient data stolen in cyber attack [Internet]. BBC News. 2024. Available from: https://www.bbc.com/news/articles/c9777v4m8zdo

[11] Chumbar S. The crisp-dm process: a comprehensive guide [Internet]. Medium. 2023. Available from: https://medium.com/@shawn.chumbar/the-crisp-dm-process-a-comprehensive-guide-4d893aecb151

[12] Kdd-cup-99 task description [Internet]. Available from: https://kdd.ics.uci.edu/databases/kddcup99/task.html

[13] OSI model: 7 layers & common security attacks | Stackscale [Internet]. 2023. Available from: https://www.stackscale.com/blog/osi-model/

[14] Michal Remper. Cisco Medical Grade Network Campus Architectures [Internet]. 2011. Available from: https://www.cisco.com/c/dam/global/sk_sk/assets/expo2011/pdfs/Cisco_Medical_Grade_Network_MichalRemper.pdf

[15] How to set up NHS WiFi [Internet]. NHS Digital . Available from: https://digital.nhs.uk/services/nhs-wifi/how-to-set-up-nhs-wifi

[16] NHS WiFi [Internet]. NHS Digital. Available from: https://digital.nhs.uk/services/nhs-wifi

[17] Graham M, Mehra A, Hambleton P. Creating Secure, Agile and Resilient Healthcare [Internet]. 2023. Pages 1, 16. Available from: https://www.cisco.com/c/dam/global/en_uk/solutions/industries/pdfs/healthcare-secure-infrastructure.pdf

[18] England NHS. NHS England » Purpose of the GP electronic health record [Internet]. [cited 2025 Apr 23]. Available from: https://www.england.nhs.uk/long-read/purpose-of-the-gp-electronic-health-record/

[19] The components & advantages of the star topology [Internet]. [cited 2025 Apr 23]. Available from: https://www.netmaker.io/resources/star-topology

[20] Kumar S, Singh M, Singh S. A Statistical Analysis on KDD Cup '99 Dataset for the Network Intrusion Detection System [Internet]. 2020. Available from: https://www.researchgate.net/publication/341077418_A_Statistical_Analysis_on_KDD_Cup'99_Dataset_for_the_Network_Intrusion_Detection_System

[21] Özgür A, Erdem H. A review of KDD99 dataset usage in intrusion detection and machine learning between 2010 and 2015 [Internet]. 2016. Available from: https://peerj.com/preprints/1954.pdf

[22] Denial of Service (Dos) guidance [Internet]. Available from: https://www.ncsc.gov.uk/collection/denial-service-dos-guidance-collection

[23] Intrusion detection systems | saylor academy [Internet]. Available from: https://learn.saylor.org/mod/book/tool/print/index.php?id=29755&chapterid=5449

[24] 1.3.5.11. Measures of Skewness and Kurtosis [Internet]. Available from: https://www.itl.nist.gov/div898/handbook/eda/section3/eda35b.htm

[25] Kristianto NG. Decoding the power of encoding in machine learning [Internet]. Medium. 2023 Available from: https://medium.com/@nicholasgabrielkr/decoding-the-power-of-encoding-in-machine-learning-39572e9cc6a3

[26] Label encoding in python [Internet]. GeeksforGeeks. 2018. Available from: https://www.geeksforgeeks.org/ml-label-encoding-of-datasets-in-python/

[27] Jawale C. Using label encoder on unbalanced categorical data in machine learning using python [Internet]. Medium. 2020 Available from: https://medium.com/@chexki_/using-label-encoder-on-unbalanced-categorical-data-in-machine-learning-using-python-435f521323b1

[28] Mubeen H. What is one hot encoding? [Internet]. Available from: https://www.educative.io/blog/one-hot-encoding

[29] Curse of dimensionality [Internet]. DeepAI. 2019. Available from: https://deepai.org/machine-learning-glossary-and-terms/curse-of-dimensionality

[30] O C. Learn one hot encoding in 3 minutes [Internet]. Medium. 2023. Available from: https://blog.devgenius.io/learn-one-hot-encoding-in-3-minutes-d1ef270d6e86

[31] Mutual_info_classif [Internet]. scikit-learn. Available from: https://scikit-learn/stable/modules/generated/sklearn.feature_selection.mutual_info_classif.html

[32] Streefkerk R. Pearson correlation coefficient: Definition and calculation [Internet]. Scribbr. 2023. Available from: https://www.scribbr.com/statistics/pearson-correlation-coefficient/

[33] REVIEW OF KDD CUP '99, NSL-KDD AND KYOTO 2006+ DATASETS . MILITARY TECHNICAL COURIER [Internet]. 2018;588. Available from: https://www.redalyc.org/pdf/6617/661770389006.pdf

[34] 4.2. Permutation feature importance [Internet]. scikit-learn. Available from: https://scikit-learn/stable/modules/permutation_importance.html

[35] Variancethreshold [Internet]. scikit-learn. Available from: https://scikit-learn/stable/modules/generated/sklearn.feature_selection.VarianceThreshold.html

[36] Standardscaler [Internet]. scikit-learn. Available from: https://scikit-learn/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[37] Robustscaler [Internet]. scikit-learn. Available from: https://scikit-learn/stable/modules/generated/sklearn.preprocessing.RobustScaler.html

[38] Minmaxscaler [Internet]. scikit-learn. Available from: https://scikit-learn/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

[39] He H, Garcia EA. Learning from imbalanced data. IEEE Transactions on Knowledge and Data Engineering [Internet]. 2009 Sep;21(9):1263–84. Available from: https://ieeexplore.ieee.org/document/5128907

[40] Batista G, Prati R, Carolina M. A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data [Internet]. 2004 Jun. Available from: https://www.researchgate.net/publication/220520041_A_Study_of_the_Behavior_of_Several_Methods_for_Balancing_machine_Learning_Training_Data

[41] SMOTE and Tomek Links for imbalanced data [Internet]. Available from: https://kaggle.com/code/marcinrutecki/smote-and-tomek-links-for-imbalanced-data

[42] Maaten L, Hinton G. Visualizing Data using t-SNE. 2008 Nov; Available from: https://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf

[43] What is supervised learning? | ibm [Internet]. 2024. Available from: https://www.ibm.com/think/topics/supervised-learning

[44] Casas P, Mazel J, Owezarski P. Unsupervised Network Intrusion Detection Systems: Detecting the Unknown without Knowledge [Internet]. 2008 Nov. Available from:

https://homepages.laas.fr/owe/PUBLIS/paper_comcom_2011.pdf

[45] Verkerken M, D'hooge L, Wauters T, Volckaert B, De Turck F. Unsupervised Machine Learning Techniques for Network Intrusion Detection on Modern Data. In. Available from: https://www.researchgate.net/publication/346675772_Unsupervised_Machine_Learning_Techniques_for_Network_Intrusion_Detection_on_Modern_Data

[46] Touré A, Imine Y, Semnont A, Delot T, Gallais A. A framework for detecting zero-day exploits in network flows. Computer Networks [Internet]. 2024 Jun 1 ;248:110476. Available from: https://www.sciencedirect.com/science/article/pii/S1389128624003086

[47] Wang S, Balarezo Serrano J, Sithamparanathan K, Al-Hourani A, Gomez K, Rubinstein B. Machine Learning in Network Anomaly Detection: A Survey [Internet]. 2021. Available from: https://www.researchgate.net/publication/356074571_Machine_Learning_in_Network_Anomaly_Detection_A_Survey

[48] What is bagging? | ibm [Internet]. 2021. Available from: https://www.ibm.com/think/topics/bagging

[49] Lab_sheets_public/lab7/lab7-trees_and_ensembles. Ipynb at main · uob-coms30035/lab_sheets_public [Internet]. GitHub. Available from: https://github.com/uob-COMS30035/lab_sheets_public/blob/main/lab7/lab7-trees_and_ensembles.ipynb

[50] A simple explanation of gini impurity - victorzhou. Com [Internet]. Available from: https://victorzhou.com/blog/gini-impurity/

[51] Decision trees, random forests, and overfitting – machine learning for biologists [Internet]. Available from: https://carpentries-incubator.github.io/ml4bio-workshop/04-trees-overfitting/index.html

[52] Introduction to Boosted Trees — xgboost 3.0.0 documentation [Internet]. Available from: https://xgboost.readthedocs.io/en/release_3.0.0/tutorials/model.html

[53] Technology T. Light gbm light and powerful gradient boost algorithm [Internet]. Medium. 2023. Available from: https://medium.com/@turkishtechnology/light-gbm-light-and-powerful-gradient-boost-algorithm-eaa1e804eca8

[54] What is a neural network? | ibm [Internet]. 2021. Available from: https://www.ibm.com/think/topics/neural-networks

[55] Goled S. How do activation functions introduce non-linearity in neural networks? | aim [Internet]. Analytics India Magazine. 2021. Available from: https://analyticsindiamag.com/ai-features/how-do-activation-functions-introduce-non-linearity-in-neural-networks/

[56] Özgür A, Erdem H. A Review of KDD99 Dataset Usage in Intrusion Detection and Machine Learning between 2010 and 2015 [Internet]. PeerJ Preprints; 2016. Available from: https://peerj.com/preprints/1954.pdf

[57] Zeng B, Li S, Gao X. Threshold-driven K-means sector clustering algorithm for wireless sensor networks. EURASIP Journal on Wireless Communications and Networking [Internet]. 2024 Sep 4 [cited 2025 Apr 30];2024(1):68. Available from: https://doi.org/10.1186/s13638-024-02403-2

[58] Tony Liu F, Ming Ting K, Zhou ZH. Isolation Forest. In 2008. Available from: https://www.researchgate.net/publication/224384174_Isolation_Forest

[59] Mougan C. Isolation forest from scratch [Internet]. TDS Archive. 2020 [cited 2025 Apr 30]. Available from: https://medium.com/data-science/isolation-forest-from-scratch-e7e5978e6f4c

[60] zfeng. Coding gaussian mixture model (And em algorithm) from scratch [Internet]. Medium. 2020 [cited 2025 Apr 30]. Available from: https://medium.com/@zhe.feng0018/coding-gaussian-mixture-model-and-em-algorithm-from-scratch-f3ef384a16ad

[61] Gridsearchcv [Internet]. scikit-learn. Available from: https://scikit-learn/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

[62] Lab_sheets_public/lab2/lab2-neural_nets. Ipynb at main · uob-coms30035/lab_sheets_public [Internet]. GitHub. Available from: https://github.com/uob-COMS30035/lab_sheets_public/blob/main/lab2/lab2-neural_nets.ipynb

[63] Cross-validation: evaluating estimator performance [Internet]. scikit-learn. Available from: https://scikit-learn/stable/modules/cross_validation.html

[64] Olamendy JC. A comprehensive guide to stratified k-fold cross-validation for unbalanced

data [Internet]. Medium. 2024. Available from: https://medium.com/@juanc.olamendy/a-comprehensive-guide-to-stratified-k-fold-cross-validation-for-unbalanced-data-014691060f17

[65] Gupta B. Random search in machine learning [Internet]. Scaler Topics. 2023. Available from: https://www.scaler.com/topics/machine-learning/random-search-in-machine-learning/

[66] Sapre S, Ahmadi P, Islam K. A Robust Comparison of the KDDCup99 and NSL-KDD IoT Network Intrusion Detection Datasets Through Various Machine Learning Algorithms [Internet]. 2019. Available from: https://arxiv.org/pdf/1912.13204

[67] Classification evaluation metrics: accuracy, precision, recall, and f1 visually explained [Internet]. Cohere. Available from: https://cohere.com/blog/classification-eval-metrics

[68] Jayaswal V. Performance metrics: confusion matrix, precision, recall, and f1 score [Internet]. Towards Data Science. 2020. Available from: https://towardsdatascience.com/performance-metrics-confusion-matrix-precision-recall-and-f1-score-a8fe076a2262/

[69] What is hyperparamter tuning with examples? [Internet]. Built In. Available from: https://builtin.com/articles/hyperparameter-tuning

[70] Vijay R, Manoj S, Ravikanth PV, Vikas Y, Indira Priyadarshini P. Augmenting Network Intrusion Detection System using Extreme Gradient Boosting (XGBoost). In. Available from: https://www.researchgate.net/publication/352816736_Augmenting_Network_Intrusion_Detection_System_using_Extreme_Gradient_Boosting_XGBoost?enrichId=rgreq-244456ca9b76036500e6748a55d1859c-XXX&enrichSource=Y292ZXJQYWdlOzM1MjgxNjczNjtBUzoxMDM5OTY3MTA3MTc4NDk4QDE2MjQ5NTg4ODg1NzI%3D&el=1_x_3&_esc=publicationCoverPdf

[71] Khafajeh H. An Efficient Intrusion Detection Approach Using Light Gradient Boosting. 2020; Available from: https://www.researchgate.net/publication/347934641_AN_EFFICIENT_INTRUSION_DETECTION_APPROACH_USING_LIGHT_GRADIENT_BOOSTING

[72] Hult G. Evaluation of Unsupervised Anomaly Detection in Structured API Logs: A Comparative Evaluation with a Focus on API Endpoints [Internet]. Linköping University; 2024. Available from: https://liu.diva-portal.org/smash/get/diva2:1866087/FULLTEXT01.pdf

[73] Cisco Systems. Network architecture blueprint (C-nab) for nhs organisations [Internet]. 2011. Available from: https://www.cisco.com/c/dam/global/en_uk/assets/public_sector/health_care/assets/cnab_business_requirements.pdf

[74] Kali docs | kali linux documentation [Internet]. Kali Linux. Available from: https://www.kali.org/docs/

[75] GNS3 Documentation Team. Your First Cisco Topology [Internet]. GNS3 Documentation. Available from: https://docs.gns3.com/docs/getting-started/your-first-cisco-topology/

[76] AI-IDS/kdd99_feature_extractor [Internet]. AI-IDS; 2025. Available from: https://github.com/AI-IDS/kdd99_feature_extractor

[77] Joblib: running Python functions as pipeline jobs — joblib 1.4.2 documentation [Internet]. Available from: https://joblib.readthedocs.io/en/stable/

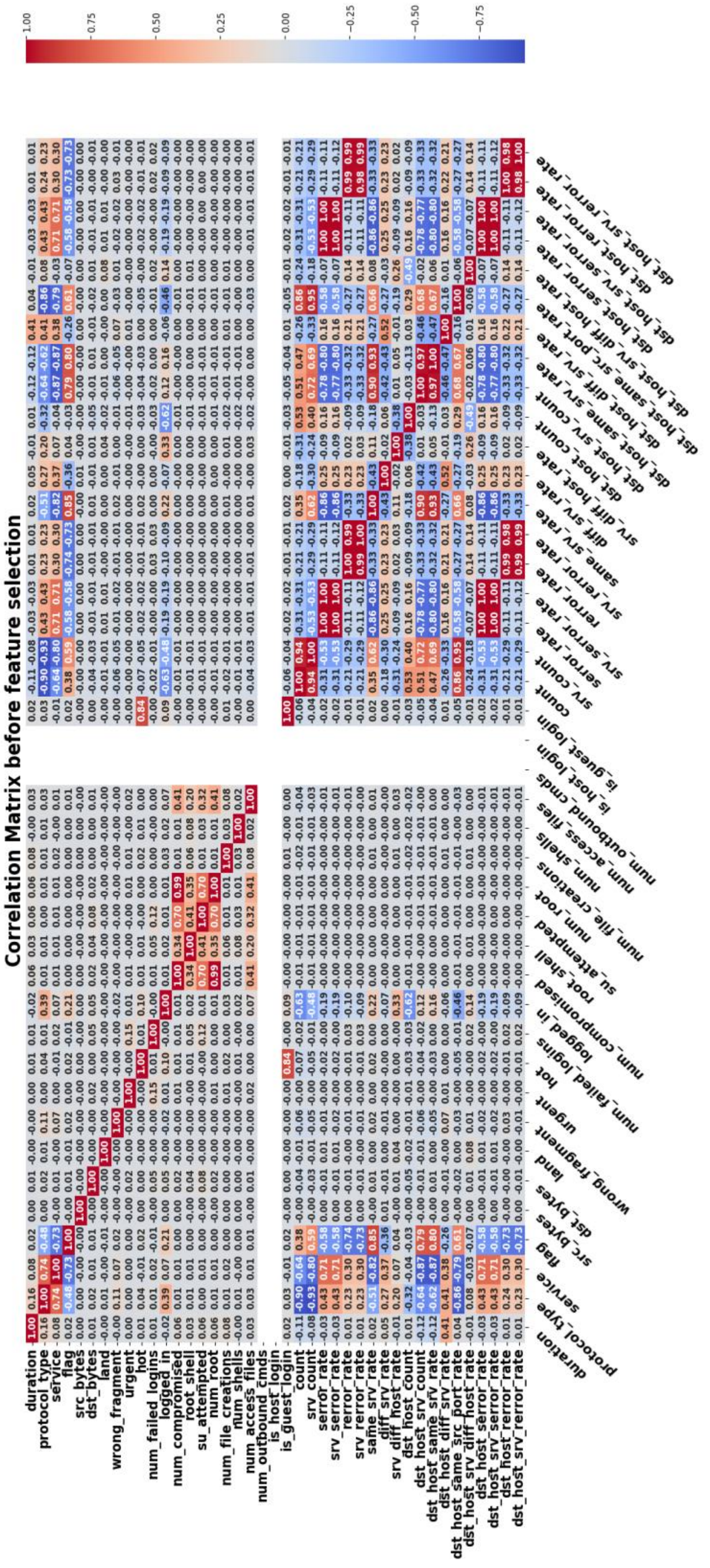
[78] Denning D. An Intrusion-Detection Model. 1987; Available from: https://www.cs.colostate.edu/~cs656/reading/ieee-se-13-2.pdf

[79] Mukkamala S, Sung A, Abrahamb A. Intrusion detection using an ensemble of intelligent paradigms [Internet]. 2004. Available from: https://waw.softcomputing.net/jnca2.pdf

[80] Chliah H, Ait El Hadj M, Battou A, Laoufi A. Tlbs-ids: a new hybrid network intrusion detection system for imbalanced dataset in a higher education setting. SN COMPUT SCI [Internet]. 2025 Feb 8;6(2):149. Available from: https://doi.org/10.1007/s42979-025-03656-4

Appendix

Figure A


Correlation Matrix before feature selection
duration
protocol_type
service
flag
src_bytes
dst_bytes
land
wrong_fragment
urgent
hot
num_failed_logins
logged_in
num_compromised
root_shell
su_attempted
num_root
num_file_creations
num_shells
num_access_files
num_outbound_cmds
is_host_login
is_guest_login
count
srv_count
serror_rate
srv_serror_rate
rerror_rate
srv_rerror_rate
same_srv_rate
diff_srv_rate
srv_diff_host_rate
dst_host_count
dst_host_srv_count
dst_host_same_srv_rate
dst_host_diff_srv_rate
dst_host_same_src_port_rate
dst_host_srv_diff_host_rate
dst_host_serror_rate
dst_host_srv_serror_rate
dst_host_rerror_rate
dst_host_srv_rerror_rate