symmetry

Article

Evaluation of Machine Learning Algorithms in Network-Based Intrusion Detection Using Progressive Dataset

Tuan-Hong Chua and Iftekhar Salam *

School of Computing and Data Science, Xiamen University Malaysia, Sepang 43900, Malaysia; swe1809792@xmu.edu.my
* Correspondence: iftekhar.salam@xmu.edu.my

Abstract: Cybersecurity has become one of the focuses of organisations. The number of cyberattacks keeps increasing as Internet usage continues to grow. As new types of cyberattacks continue to emerge, researchers focus on developing machine learning (ML)-based intrusion detection systems (IDS) to detect zero-day attacks. They usually remove some or all attack samples from the training dataset and only include them in the testing dataset when evaluating the performance. This method may detect unknown attacks; however, it does not reflect the long-term performance of the IDS as it only shows the changes in the type of attacks. In this work, we focused on evaluating the long-term performance of ML-based IDS. To achieve this goal, we proposed evaluating the ML-based IDS using a dataset created later than the training dataset. The proposed method can better assess the long-term performance as the testing dataset reflects the changes in the attack type and network infrastructure changes over time. We have implemented six of the most popular ML models, including decision tree (DT), random forest (RF), support vector machine (SVM), naïve Bayes (NB), artificial neural network (ANN), and deep neural network (DNN). These models are trained and tested with a pair of datasets with symmetrical classes. Our experiments using the CIC-IDS2017 and the CSE-CIC-IDS2018 datasets show that SVM and ANN are most resistant to overfitting. Our experiments also indicate that DT and RF suffer the most from overfitting, although they perform well on the training dataset. On the other hand, our experiments using the LUFlow dataset have shown that all models can perform well when the difference between the training and testing datasets is small.

Keywords: intrusion detection; machine learning; deep learning; cybersecurity

check for updates

Citation: Chua, T.-H.; Salam, I. Evaluation of Machine Learning Algorithms in Network-Based Intrusion Detection Using Progressive Dataset. *Symmetry* **2023**, *15*, 1251. https://doi.org/10.3390/sym15061251

Academic Editors: Aviv Gibali and Christos Volos

Received: 4 April 2023
Revised: 20 May 2023
Accepted: 26 May 2023
Published: 13 June 2023

1. Introduction

Cybersecurity has gained more attention in recent years as information systems play a more important role in our daily lives. At the same time, cyberattacks are also skyrocketing. The FBI reported that the cyberattacks on their Cyber Division had increased by 400% to nearly 4000 attacks per day [1] since the global pandemic. These include attacks on critical systems, such as the supervisory control and data acquisition (SCADA) system, power system, and the Stuxnet virus on a nuclear power plant. The economic impact of these cyber crimes is huge, and it is estimated that cybercrime will cost companies worldwide from 3 trillion USD in 2015 to an estimated 10.5 trillion USD annually by 2025 [2]. Therefore, organisations should take necessary countermeasures to address cyberattacks. There are multiple countermeasures available to safeguard the security of an organisation's computer system, including network access control, antivirus software, and Virtual Private Networks (VPN). An Intrusion Detection System (IDS) is also one of the control measures. An intrusion detection system (IDS) is a hardware or software application that ensures the security of computer systems by monitoring traffic for the sign of intrusions [3]. Once suspicious activities have been identified, an alarm will be raised and IT personnel can take action accordingly. Depending on the monitored traffic, an IDS can be classified as a network-based IDS (NIDS) or a host-based IDS (HIDS) [3]. An NIDS monitors the network traffic while HIDS monitors operating system files. In the literature, NIDS is usually referred to as flow-based NIDS, where only the packet header is analysed to detect intrusions.

Besides NIDS and HIDS, an IDS can also be classified as signature-based IDS or anomaly-based IDS [3], based on the method that is used to detect intrusions. A signature-based IDS is also known as misuse detection [4], where attacks are identified by utilising the signature of known attacks stored in the database. Hence, it cannot capture attacks that it has never seen before. The advantage of signature-based IDS is zero false-positive rates, as it will never classify a benign activity as malicious [5]. On the other hand, an anomaly-based IDS classifies network traffic with a set of pre-defined rules for "normal activity". If an activity does not fit into those rules, it will be classified as "suspicious activity" [6]. Therefore, an anomaly-based IDS is also known as a rule-based IDS [3]. The main distinction from a signature-based IDS is that it identifies attacks based on traffic behaviour instead of explicit signatures. This gives the anomaly-based IDS the flexibility to identify new attacks.

In recent years, existing works have focused heavily on adapting machine learning (ML) to improve the accuracy of IDS [7–10]. When used for IDS, ML can learn the behaviour of benign and malicious network traffic and differentiate between them. Symmetric patterns in network traffic data can be used to detect different types of attacks, such as port scans or TCP SYN floods. Symmetry can also identify anomalies or outliers in the data by comparing the distributions of features across different groups or periods. Recent research has proven that IDS that adopts ML algorithms can achieve good accuracy and surpass conventional methods. The application of machine learning algorithms for intrusion detection systems has been shown to be effective for real-world applications, such as the SCADA network, power systems, and communication networks [2]. One weakness the IDS typically suffers from is detecting unseen attacks, specifically zero-day attacks. A zero-day attack occurs when the attackers exploit a system vulnerability unknown to the developer or before the developers address it [11]. Although having high accuracy in detecting known attack activities, IDS often has the limitation of detecting zero-day attacks and unseen attacks. The weakness might have been solved by utilising machine learning (ML). Research conducted by Hindy et al. [12] has shown promising results in detecting zero-day attacks by using support vector machine (SVM) and artificial neural networks (ANNs) for IDS.

Studies in recent years have achieved compelling results, and various models have been used for IDS [13]. However, the method of evaluating the IDS is not getting much attention. In most studies, a single dataset is used for both training and evaluation. To evaluate the IDS's ability to detect unseen attacks, the most common way is to purposely exclude some or all attack activities from the training dataset. The KDD Cup 1999 dataset [14] is an example where some attack activities were excluded from the training dataset. The study by Hindy et al. [12] took it one step further by using only the benign traffic for training, while the attack activities were only included in the testing dataset to mimic zero-day attacks. This method can evaluate the ability of the model to detect zero-day attacks, but it is not comprehensive enough to evaluate the long-term performance of the model. In a real-world scenario, we usually have to train a model using an existing dataset while using the model to examine future network traffic. The point is that the network environment in the future will not be the same as today. For instance, organisations and attackers will update their infrastructures and network topologies over time. Additionally, zero-day attacks in the future will never be available in the existing dataset. The interest of this paper was to further increase the deviation between the data used for training and the data used for evaluation to mimic real-world situations. Thus, we used different datasets to train and evaluate our ML models. The difference between the existing methods and the proposed method is further illustrated in Figure 1.

Based on the proposed method, our contributions in this paper are listed below:

(a) We proposed techniques for training and evaluating ML-based IDS using different datasets to mimic real-world scenarios.

(b) We identified multiple datasets for the evaluation, including the CIC-IDS2017, the CSE-CIC-IDS2018, and the LUFlow datasets.

(c) We compared the performance of decision tree (DT), random forest (RF), support vector machine (SVM), naïve Bayes (NB), artificial neural network (ANN) and deep neural network (DNN) using the proposed method.

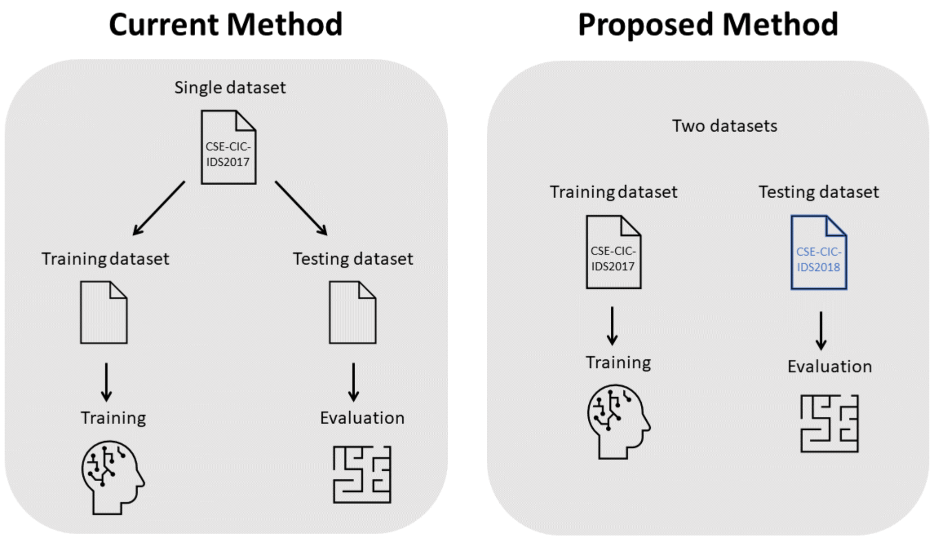


Figure 1. Current studies used the same dataset for both training and testing, while this paper proposed to use different datasets for training and testing.

The remainder of the paper is organised as follows: Section 2 provides some related background. In Section 3, we reviewed related literature. The experiment process and experiment environment are described in Section 4. Section 5 discusses the experiment results, and further discussions on the results are provided in Section 6. Finally, Section 7 concludes this paper.

2. Related Background

In this section, we introduce some of the most well-known datasets that are used in the domain of IDS. We also discuss the machine learning models that are used in this paper.

2.1. Dataset

In this study, the selection of a dataset was one of the biggest challenges. Due to privacy and security reasons, most organisations will never share their network traffic data. Most publicly available datasets are created independently by various organisations, and each has a different feature set. Hence, it is nearly impossible to use multiple datasets to evaluate a model without retraining the model for each dataset. In this paper, we used two methods to identify datasets that suit our needs. First, we found datasets created by the same organisation, which gave the highest chance of using the same feature set and the same unit for each dataset. Second, we found a real-world time series dataset that was collected over a long period. From a time series, we could choose the data collected earlier to be the training dataset, while the data collected later would be the testing dataset. The first method led us to finding the CIC-IDS2017 and the CSE-CIC-IDS2018 datasets. On the other hand, we used the LUFlow dataset for the second method. The details of the selected datasets are discussed in the remainder of this section.

2.1.1. CIC-IDS2017 Dataset

The CIC-IDS2017 dataset [15] is one of the most popular datasets from recent years. The dataset was created by the Canadian Institute for Cybersecurity (CIC) in 2017. As the dataset was created recently, it covers various operating systems, protocols, and attack types. A comprehensive network environment consisting of modems, firewalls, switches, and routers, with different operating systems—including Windows, Mac OS, and Ubuntu—was set up to create the dataset. The behaviour of 25 users was simulated, and protocols such as HTTP, HTTPS, FTP, SSH, and email protocols were used to develop benign traffic. After that, the most common attacks in 2016 were simulated. The attacks include brute force attacks, DoS attacks, DDoS attacks, infiltration attacks, heart-bleed attacks, botnet attacks, and port scan attacks. The dataset was then made publicly available on the University of New Brunswick's website [16] as a CSV file. The biggest downside of the dataset is that it suffers from the high-class imbalance problem [17], where more than 70% of the traffic is benign, and some of the attacks contribute to less than 1% of the overall traffic.

2.1.2. CSE-CIC-IDS2018 Dataset

The CSE-CIC-IDS2018 is an updated version of the CIC-IDS2017 dataset. The dataset was created by CIC in conjunction with the Communications Security Establishments (CSE). The dataset is publicly accessible [18] through Amazon Web Service (AWS) and has been widely used in recent studies. The most significant update is the laboratory environment that was used to create the dataset; it is significantly more robust than its predecessor. Fifty machines were used to attack the infrastructure, and the victim organisation was simulated with 420 machines and 30 servers across five departments. Additionally, different versions of the Windows operating system were installed on the victim's devices, including Windows 8.1 and Windows 10. The servers were also running on different server operating systems, including Windows Server 2012 and Windows Server 2016 [19]. The diversity of the devices in the laboratory environment makes them more similar to real-world networks. The attacks towards the infrastructure are similar to those in the CIC-IDS2017 dataset.

2.1.3. LUFlow Dataset

The LUFlow dataset [20] is one of the latest datasets for the domain of IDS. Lancaster University created the dataset in 2020, and it was still being updated in 2021. The LUFlow dataset is unique because the data are real-world data, which are captured through the honeypots within Lancaster University's public address space. Hence, the dataset can reflect the emerging threats in the real world. The limitation of the dataset is that it only labels the traffic as benign or malicious instead of as an explicit type of attack. This limitation is due to the fact that the data are collected from the real world instead of a laboratory environment; hence, it is impossible to tell the real intention behind each traffic. Additionally, some traffic is classified as an "outlier" if the traffic contains suspicious activity but without connection to a malicious node. The existence of the outlier label indicates that unknown attacks may be classified as outliers instead of malicious and might cause the dataset to be less meaningful.

2.2. Machine Learning Models

Machine learning (ML) models have shown promising results in predicting or classifying data in numerous research fields [21,22]. In the context of IDS, ML is adopted to classify whether the traffic is benign or an attack. In this section, we briefly discuss some of the most widely used ML models in the domain of IDS.

2.2.1. Decision Tree

A decision tree (DT) classifier is a model that classifies the sample using a tree-like structure. Each decision tree is made up of multiple nodes and leaves, where the nodes represent some conditions while the leaves represent different classes [23]. A decision tree will be built during the training stage based on the training data. Figure 2 shows an example of the creation process of a decision tree. This is an iterative process. In each iteration, one feature will be selected to split the data on a leaf node until all data in a leaf node are homogeneous. The selection of features is based on how well a feature can split the data into a homogeneous group, measured using metrics such as Gini and Entropy. The features closer to the root node split the data better and have a higher correlation with the predicted output. Therefore, the decision tree can facilitate feature selection. In the testing stage, each sample in the testing dataset will be input to the tree and traverse from the root node, i.e., the node at the top of the tree, to a leaf node. The leaf node to which a sample ends represents the class of that sample.

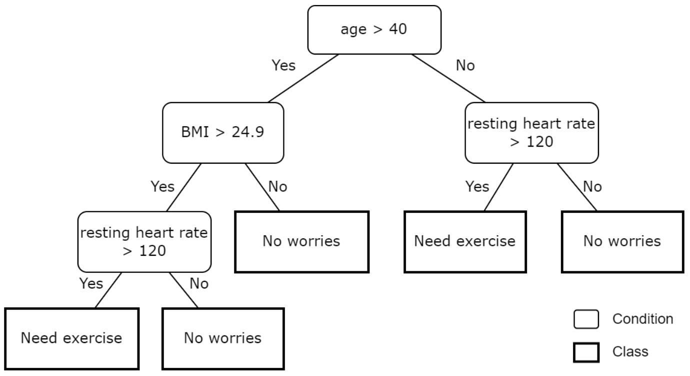


Figure 2. An example of a decision tree.

2.2.2. Random Forest

Random forest (RF) [24] is a model made up of a branch of decision trees. A bootstrapped dataset is created to build each decision tree within the random forest by randomly picking rows from the original training dataset. Since each decision tree is trained using a different bootstrapped dataset, each is slightly different from the others. As illustrated in Figure 3, the votes from each tree are collected, and the classification depends on the class that received the most votes. By involving multiple decision trees, random forest is less prone to overfitting problems and less sensitive to the noise in the dataset while maintaining the simplicity of the decision tree [24].

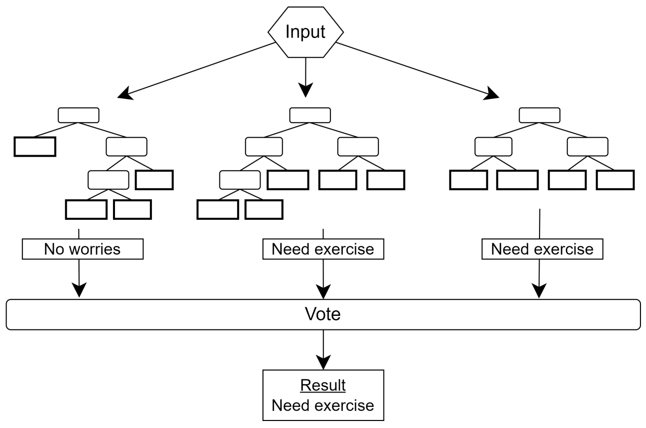


Figure 3. Illustration of a simple random forest with three trees.

2.2.3. Support Vector Machine

Support vector machine (SVM) is a model that aims to define a hyperplane that can separate the training data into its respective classes [25]. As illustrated in Figure 4, the thick line represents a separator in a 2-dimensional space that separates the data of two different classes. To prevent overfitting, the separator in Figure 4 should be in the middle of the dashed lines to ensure that the distance to data points of either class is the maximum. One challenge of SVM is that the data might be inseparable from the input space. To solve this issue, kernel functions are used to non-linearly map the data to a higher dimensional feature space [25]. If the data are still inseparable in the higher dimensional space, the model will map the data to an even higher-dimensional space. The downside of SVM is that it is computationally more expensive than other ML models. Despite that, the model is still widely used as it is less prone to overfitting.

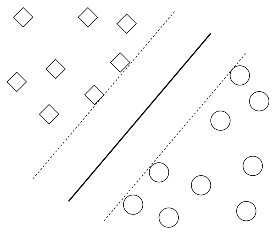


Figure 4. A simple SVM in a 2-dimensional space.

2.2.4. Naïve Bayes

The naïve Bayes (NB) classifier [26] is a simple probabilistic classifier based on the Bayes theorem. The equation of the Bayes theorem is shown in Equation (1).

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}. \tag{1}$$

In the training phase, the probabilities of each classification having a certain characteristic are calculated. The probabilities are then used to classify the data based on the given characteristic. As probability calculation is a simple task, the model is very efficient. Although the model is "naïve" as it assumes that each feature is independent of the other, it sometimes achieves comparable accuracy to other models.

2.2.5. Artificial Neural Network

Artificial neural network (ANN) is a more complicated model than the models discussed above. The multi-layer perceptron (MLP), a type of ANN, was adapted in this work. The structure of an MLP is typically represented using a graph as shown in Figure 5. As its name suggests, an MLP consists of multiple layers of the perceptron [27]. There are multiple nodes on each perceptron layer, where the nodes are also referred to as neurons. The example shown in Figure 5 is an MLP consisting of three perceptron layers. The leftmost layer is also called the input layer. Each neuron from the input layer represents one feature of the input data. The rightmost layer is known as the output layer, and each neuron in the output layer represents the class of the data [28]. The neurons between different layers are connected by unidirectional weighted vertices. Data will be fed to the input layer and moved forward through the hidden layer to the output layer. Hence such a model is often referred to as a feed-forward neural network (FNN) [29,30]. We will refer to this as ANN in the rest of this paper.

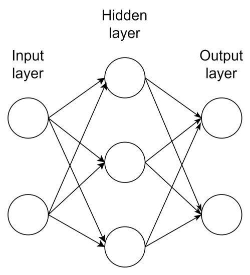


Figure 5. A simple MLP with one hidden layer.

The mapping from a perceptron to a neuron on the next layer is as simple as a linear function and a non-linear activation function. The mapping illustrated in Figure 6 can be represented with the following Equation [29,31]:

$$h_i(x) = f(w_i^T x + b), \tag{2}$$

where x denotes the inputs to the next layer, w_i represents the weights on the vertices, b denotes the biases, and f denotes the activation function. When training an ANN, the task is to optimise the weights (w) and biases (b) so that the model will fit the training data. As an ANN often includes a large number of neurons, there will be many parameters to be optimised. Hence, training an ANN is computationally more expensive than other models, such as decision trees and naïve Bayes. Additionally, a large dataset is required to train the neural network to prevent overfitting [28].

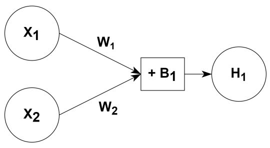


Figure 6. Mapping to a neuron on next perceptron.

The main advantage of neural networks is that they overcome the limitation of processing the data in their raw form. Conventional models, such as decision trees, are limited when processing the raw data; hence, they heavily rely on hand-designed features. In other words, neural networks have a better chance of achieving good accuracy when the features of the training dataset are not well optimised. The neural networks overcome the limitation by transforming the data to a more abstract representation as the data go through different perceptrons. As a result, neural networks can learn very complex patterns [32].

2.2.6. Deep Neural Network

The deep neural network (DNN) has gained popularity since Hinton, Osindero, and Teh [33] successfully trained a DNN with three hidden layers in 2006. The application of DNN in the domain of IDS has also gained popularity in recent years [30]. The DNN is an MLP under the classification of a deep learning (DL) model. In general, an ANN is considered a deep model when it contains two or more hidden layers [30,34]. Compared to MLP with a single hidden layer, DNN performs better when pre-training is eliminated [35]. Additionally, DNN has the ability to learn more complicated patterns with its deep architecture. With the additional complexity, DNNs take longer to train and require a larger dataset to optimise its parameter.

3. Related Works

The adoption of ML in IDS has been widely explored over the last decade. A review by García-Teodoro et al. [6] showed that researchers have been working in the domain since the 2000s. Researchers are still developing more advanced algorithms to improve the accuracy and efficiency of the ML-based IDS.

In 2019, Kaja, Shaout, and Ma [36] proposed a two-stage IDS that combines an unsupervised model and a supervised model to mitigate the false positive and false negative. In the first stage, the model uses k-means clustering, an unsupervised model, to detect malicious activity. In the second stage, supervised models such as decision tree, random forest and naïve Bayes are used to classify the malicious activity. The authors showed that the proposed models could achieve 92.74% to 99.97% accuracy on the KDD99 dataset. In the same year, Kanimozhi and Jacob [37] conducted a study on utilising ANN for IDS. Their study focused on detecting botnet attacks in the CSE-CIC-IDS2018 dataset. With hyperparameter optimisation, they achieved an accuracy of 99.97%, with an average false positive rate of only 0.001.

Besides conventional ML models such as SVM, decision tree and naïve Bayes, researchers are also exploring the application of modern models such as deep learning in the domain of IDS. The review by Ferrag et al. [30] showed that more than 40 works have been carried out for the adoption of deep learning in the domain of IDS. In 2019, Vinayakumar et al. [29] conducted a study focusing on the adoption of deep learning in both HIDS and NIDS. The study proposed a scalable and hybrid DNNs framework. The authors showed that the proposed DNNs framework could achieve an accuracy rate of 93.5%. Although the accuracy rate is not as high compared to DT and RF, the proposed framework is claimed to be computationally inexpensive for training.

We can see from the recent literature that IDS with ML models can easily achieve a higher than 90% accuracy. Hence, some literature focuses on a comparative study, where multiple models are implemented and evaluated using various datasets. In 2018, Verma and Ranga [38] implemented ten models for IDS, including both supervised and unsupervised ML models. The authors evaluated multiple ML models, including ANN, deep learning, KNN, and SVM. They provided a detailed comparative study using the CIDDS-001 dataset. The result showed that KNN, SVM, DT, RF, and DL are the best-performing models, with accuracy rates above 99.9%.

In 2020, Ferrag et al. [30] summarised more than 40 works that implemented deep learning for IDS and described 35 well-known datasets in the domain of IDS. The authors also implemented seven deep learning models and compared the performance with naïve Bayes, ANN, SVM and random forest. In the study, they used CSE-CIC-IDS2018 and the Bot-IoT datasets. The result shows that the deep learning models achieved a 95% detection rate, which outperformed the 90% detection rate achieved by other models.

In 2021, a comprehensive literature review was conducted by Kilincer et al. [39] to compare the performance of SVM, KNN, and DT. Multiple datasets were used for the comparative analysis, including the CSE-CIC-IDS2018, UNSW-NB15, ISCX-2012, NSL-KDD, and CIDDS-001 datasets. The results from the study showed that the accuracy of the models ranged from 95% to 100% except for the UNSW-NB15 dataset. DT consistently performed the best among all implemented models regardless of the dataset.

When evaluating the performance of the IDS, the ability to detect unknown attacks is also an area of concern. In 2020, Hindy et al. [12] focused on the performance of ML-based IDS on detecting unknown attacks. The study proposed an IDS to detect zero-day attacks with high recall rates while keeping the miss rate to a minimum. Additionally, they implemented a one-class SVM to compare with the proposed model. The study used the CIC-IDS2017 dataset and the NSL-KDD dataset for model training and evaluation. To fulfil the setting of zero-day attack detection, only the normal traffics were used when training the model and all attack activities were used to mimic zero-day attacks. The result from the study showed that both models had a low miss rate on detecting zero-day attacks.

As summarised in Table 1, we can see that recent studies commonly involve multiple ML models and multiple datasets for evaluation. However, the evaluation using different datasets is performed separately. The models are retrained when evaluating the dataset using a different dataset. This is because different datasets have different feature sets. In this paper, we identified multiple pairs of datasets for training and testing. Each pair of datasets share the same feature set. In our work, the testing dataset was created later than the training dataset so that the long-term performance of IDS can be better reflected.

Table 1. ML models and datasets that are used in recent literature.

Ref.	ML Models	Datasets Used
[38]	ANN, DL, RF, KNN, SVM, DT, NB, KMC, EMC, SOM	CIDDS-001
[36]	KMC + DT, RF, NB	KDD99
[29]	DNNs, RF, KNN, SVM, DT, NB	KDD99, NSL-KDD, UNSW-NB15, Kyoto, WSN-DS, CIC-IDS2017
[37]	ANN	CSE-CIC-IDS2018
[30]	DNNs, NB, ANN, SVM, RF	CSE-CIC-IDS2018, Bot-IoT
[12]	ANN, SVM	CIC-IDS2017, NSL-KDD
[40]	DNN	CSE-CIC-IDS2018
[39]	SVM, KNN, DT	CSE-CIC-IDS2018, NSL-KDD, CIDDS-001, ISCX2012

4. Framework of Experiment

The experiments of this paper were separated into five main steps. The first step was dataset pre-processing, where the datasets were cleaned, and the necessary processing was carried out. The second step was feature selection, which is an important step in improving the performance of the models. The third step was to optimise the hyperparameters to improve the accuracy of the models. The accuracy of the models was validated using cross-validation after optimising the hyperparameters. Finally, the performances of the models on the testing dataset were evaluated using various metrics. The complete flow of the experiment is illustrated in Figure 7. The details of each experiment step are further discussed in Sections 4.1–4.5.

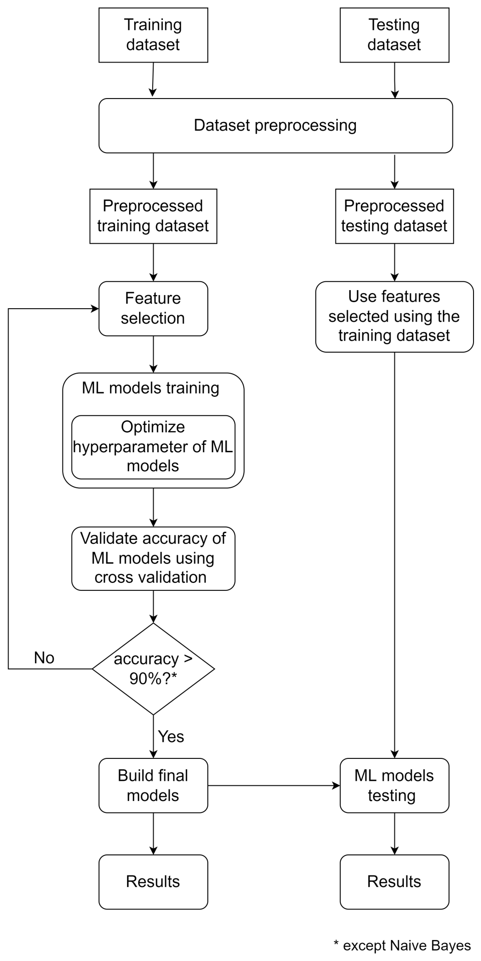

Training dataset
Testing dataset
Dataset preprocessing
Preprocessed training dataset
Preprocessed testing dataset
Feature selection
Use features selected using the training dataset
ML models training
Optimize hyperparameter of ML models
Validate accuracy of ML models using cross validation
No
accuracy > 90%?*
Yes
Build final models
ML models testing
Results
Results
* except Naive Bayes

Figure 7. The workflow of the experiment process.

The experiment of this paper was conducted twice, each using a different set of datasets. The first set of datasets was the CIC-IDS2017 dataset [15] and the CSE-CIC-IDS2018 dataset. As described in Section 2.1, both datasets were created by Canadian Institute for Cybersecurity (CIC). As created by the same organisation, both datasets share a common set of features. The CSE-CIC-IDS2018 dataset was created one year after the CIC-IDS2017 dataset. Hence, these datasets meet the needs of this paper. We used the CIC-IDS2017 dataset as the training dataset and the CSE-CIC-IDS2018 dataset as the testing dataset.

The second set of datasets was derived from the LUFlow dataset [20]. Although it may seem contradictory to the aim of our experiment, it is important to note that the LUFlow dataset organises the data according to the day that the data are collected. In this experiment, the data collected in July 2020 were used as the training dataset, while the data collected in January 2021 were used as the testing dataset.

4.1. Data Pre-Processing

The first step of the experiment was to pre-process the dataset. First, we performed dataset cleaning by eliminating unwanted entries in the dataset. Entries containing missing or infinity values were dropped as they only contribute to a relatively small portion of the dataset. We also removed duplicates to expose the models to as many unique samples as possible.

In network intrusion detection, the dataset should be symmetrical, meaning the model must be trained and tested on an equal number of normal and abnormal network traffic instances. Maintaining symmetry is crucial because it ensures the model is not biased towards one type of traffic. If the training data were imbalanced with more instances of benign traffic than attack traffic, then the model may classify all traffic as benign, even if it were abnormal. Therefore, in the next step of the data pre-processing, we addressed the high-class imbalance problem, where some classes had significantly more samples than others. The problem results in a bias towards the majority class, which in turn makes the accuracy of the model meaningless. We downsampled the majority class to address the high-class imbalance problem and to ensure symmetry in the dataset. In our approach, samples were selected randomly to reduce the number of samples for the majority class. Additionally, multiple minority classes were combined to form a larger class.

We also ensured that both datasets had the same set of columns with the same sequence, as we proposed using different datasets for training and testing. Both datasets were also relabelled, if necessary, to ensure that both datasets had the same classes.

4.2. Feature Selection

Feature selection is an important step to improve the performance of an IDS, from the perspective of time efficiency and accuracy [40]. Since modern datasets such as the CIC-IDS2017 dataset contain around 80 features, training the model without any feature selection will consume time. Other than that, some features may include noise and reduce the accuracy of the model. As pointed out by Aksu et al. [41], the accuracy of the models starts to drop when more than 40 features of the CIC-IDS2017 dataset are used to train the model.

We used the random forest algorithm by utilising the RandomForestClassifier provided by scikit-learn to perform feature selection. A random forest was trained using the training data, and the top n features with the highest importance score were selected. The random forest algorithm has been widely adopted for feature selection. As an example, Sharafaldin, Lashkari, and Ghorbani [15] and Kostas [42] also used the random forest for feature selection on the CIC-IDS2017 dataset.

After reducing the number of features using random forest, we further reduced the number of features using brute force. We then built preliminary ML models using a different number of features. A for loop was used to add a new feature in each iteration to construct the ML models until all n features were included. The accuracy of each model was recorded with respect to the number of features. Based on the accuracy rate of the models, a more concise set of features was selected.

Moreover, some features were removed by human inspection. Some features should be removed, although having a high correlation with the output variable, the source IP address, for example, is one of them. When a dataset includes a large amount of malicious traffic from one IP address, the "source IP address" may be ranked as an important feature by the random forest. However, the feature should be removed to prevent overfitting, as classifying the traffic based on the IP address may not be relevant in the future.

4.3. ML Models Training

We trained the models using the training dataset after selecting the features. When training the models, the hyperparameters of the models were optimised using grid search by utilising the GridSeachCV function provided by scikit-learn. The grid search method works by searching through a predefined hyperparameter space, and different combinations of hyperparameters are used to train the model. The hyperparameters that give the best accuracy are chosen to train the final models. The hyperparameters of a model are parameters that govern the training process. Take DNN as an example; the number of hidden layers is the hyperparameter, while the weights and biases of the neural network are the parameters of the model. Since optimising the hyperparameters requires training the models multiple times, only a fraction of the training dataset is used to speed up the entire process.

In this work, we implemented decision tree (DT), random forest (RF), support vector machine (SVM), naïve bayes (NB), artificial neural network (ANN) and deep neural network (DNN). The implementation of each model and the hyperparameters that were optimised for each model are described as follows:

The decision tree of this work was implemented using the DecisionTreeClassifier provided by the scikit-learn library. To optimise the tree, pruning was performed to prevent overfitting. The method used to prune the decision tree is called cost complexity pruning, which is controlled by the ccp_alpha parameter. The higher the alpha value, the more nodes will be pruned, and the total impurity of the leaves will increase. The optimisation was performed by brute-forcing different values of alpha and choosing the alpha value that gives the highest accuracy.

The random forest of this work was implemented using the RandomForestClassifier, which is also provided by the scikit-learn library. Just as with the decision tree, pruning is an important method to optimise the model. However, the optimisation of the random forest is carried out by using a grid search. Hence, the pruning was performed by specifying the maximum depth of each tree, the minimum number of samples that are required on a leaf node, and the minimum number of samples that are required on a node for a split to be considered. Additionally, two formulas were tested to measure the quality of the split, Gini and entropy. By using a different strategy to evaluate the quality of a split, different features were chosen, and this would result in a different tree. Moreover, the number of trees in the random forest was also optimized.

The SVM was implemented using SVC (support vector classifier) provided by the scikit-learn library. For SVM, several important hyperparameters need to be optimised. The most important hyperparameter of SVM is the kernel function that is being used. By using a different kernel, the accuracy and time complexity of the models will be affected. The kernels used in this work included linear, Gaussian radial basis function (RBF) and sigmoid, as shown in the below equations.

$$K(x_i, x_j) = (x_i, x_j) \tag{3}$$

$$K(x_i, x_j) = \exp(-\gamma \parallel x_i - x_j \parallel^2 + C) \tag{4}$$

$$K(x_i, x_j) = \tanh(\gamma(x_i, x_j) + r). \tag{5}$$

The kernel function takes two points, x_i and x_j, as input and returns their similarity in higher dimensional space as output. Besides the kernel functions, the gamma (γ) value used by the RBF and Sigmoid kernels is optimised. Moreover, C is also an important hyperparameter of SVM. The hyperparameter C helps the SVM to prevent overfitting by allowing errors. The larger the value of C, the more errors will be allowed and hence the less sensitive it will be to outliers.

The naïve Bayes was implemented using GaussianNB, which is available in the scikit-learn library. As naïve Bayes is a very simple model, it has only one hyperparameter to be optimised, the var_smooth parameter. As Gaussian naïve Bayes assumes that the probability distribution follows a Gaussian normal distribution, increasing the var_smooth value allows the model to account for samples that are further away from the distribution mean. In other words, the hyperparameter smoothes the distribution curve.

The ANN was implemented using the MLPClassifier provided by the scikit-learn library. The most important hyperparameter is the number of hidden layers and the number of neurons on each layer. The ANN in this work was implemented with only one hidden layer and 10 to 50 neurons on the hidden layer. The activation function was also optimised. The activation functions used in this work were Logistic, Tanh and ReLU, as shown below.

$$f(x) = \frac{1}{1 + \exp(-x)} \tag{6}$$

$$f(x) = \tanh(x) \tag{7}$$

$$f(x) = \max(0, x). \tag{8}$$

The third hyperparameter to be optimised was the optimisation function, which optimises the weights and biases of the neural network. For the MLPClassifier, the optimisation function is controlled by the solver parameter. Two optimisation functions were tested in this work: stochastic gradient descent (SGD) and Adam. The last hyperparameter to be optimized for ANN was the alpha value. The parameter is a penalty term which can be used to prevent an overfitting or underfitting problem. A larger alpha value would encourage smaller weights and thus prevent overfitting, while a smaller alpha value would encourage larger weights and prevent underfitting.

The implementation of DNN is very similar to that of the ANN. The only difference is the number of hidden layers of the neural network. For the implementation of this work, the DNN had three to four hidden layers with an equal number of neurons on each layer. Additionally, the hyperparameters that were optimized were the same as for the ANN.

4.4. Model Accuracy Verification

In this step, the goal was to verify the accuracy of the models. Before we tested the models using a different dataset, it was important to ensure that they performed well on the training dataset. Hence, we measured the performance of the models using cross-validation. The training dataset was split into k-folds and the models were validated through k iteration. In each iteration i, the i-th fold of the training dataset was used for testing and the rest of the training dataset was used for training (as illustrated in Figure 8). The main goal was to ensure that the accuracy between each fold did not differ by too much, which indicates overfitting. After that, the accuracy of the models was compared with other literature. If the model achieved a comparable accuracy with other literature, we would proceed to the next step. Otherwise, the experiment process was revised for improvement.

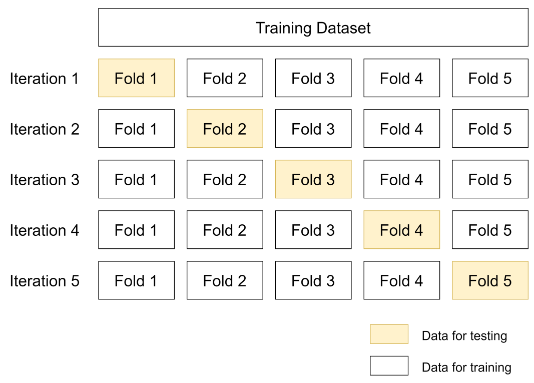


Figure 8. An example of k fold cross-validation with $k = 5$.

4.5. Model Evaluation

After verifying the accuracy of the models, the last and most crucial step of this paper was performed. The last step was training the final model and testing the models using the testing dataset. First, we used 70% of the training dataset to train the final models. Next, we used the rest of the training dataset to calculate the accuracy of the models on the training dataset. Finally, we used the testing dataset to test the models.

The interest of this paper was to compare the accuracy of each model when a different dataset was being used. At the same time, a comparison between different models was conducted in terms of accuracy and efficiency. The performance metrics that were used to measure the performance of the models included accuracy, precision, recall, and F1-score, as shown in Equations (9)–(12).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{9}$$

$$Precision = \frac{TP}{TP + FP} \tag{10}$$

$$Recall = \frac{TP}{TP + FN} \tag{11}$$

$$F1Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}. \tag{12}$$

In the above equations, true positive (TP) and true negative (TN) denote the number of samples that were correctly classified as positive and negative, respectively. False positive (FP) and false negative (FN) indicate the number of incorrectly classified samples as positive and negative, respectively. We also visualised the classification result using a confusion matrix.

Moreover, we also measured the time complexity of each model. Our primary focus was the time consumption of each model to be trained and the time consumption for prediction. We did not compare with other literature regarding time efficiency as the time consumption depends on the implementation of the model, the number of samples, and the hardware used to execute the experiment.

5. Experiments and Discussions

In this section, the experiment results are discussed in detail. Section 5.1 discusses the software and hardware environments used for our experiments. Section 5.2 discusses the experimental results using the CIC-IDS2017 and the CSE-CIC-IDS2018 datasets, while Section 5.3 discusses the experimental results using the LUFlow dataset. The experimental details have also been made publicly accessible via a GitHub repository and a pre-print version of the work; refer to the references [43,44] to access the source codes and experimental details for all our experiments.

5.1. Experiment Environment

We first introduce the experiment environment of this work. The environment for the experiment was important as it affected the performance of the implementation. The ML models of this paper were implemented using the Python programming language in the conda environment. The conda version used is 4.10.1, which is based on Python 3.8.8. Python libraries, including scikit-learn [45], pandas, NumPy, and Matplotlib, were used for the implementation and experiments of this work. The hardware environment used for this paper was a laptop powered by the Intel® Core™ i7-8550U Central Processing Unit (CPU). The main frequency of the processor was 1.80 GHz, equipped with 8 GB of Random-Access Memory (RAM). The Graphics Processing Unit (GPU) of the system was an Intel® UHD Graphics 620 integrated graphics unit. The system was running on the Windows 11 Home 64-bit operating system.

5.2. Experiment Using CIC's Dataset

In this section, the experiment results of using the datasets created by the CIC are discussed. The CIC-IDS2017 dataset was used for training and the CSE-CIC-IDS2018 dataset was used for testing.

5.2.1. Dataset Pre-Processing

Both the CIC-IDS2017 and the CSE-CIC-IDS2018 datasets are huge datasets, where the former contains 800MB of data and the latter contains 6.4GB of data. The CSE-CIC-IDS2018 dataset is too large to be handled by the hardware used in our work; hence, we only used 10% of the dataset.

Although the CIC-IDS2017 and CSE-CIC-IDS2018 datasets are some of the most recently created datasets, they still require some pre-processing. In both datasets, there are a small number of entries that contain missing values or infinity values. Those entries were removed as they only contributed to a small portion of the dataset. Duplicates in both datasets were also removed.

After the datasets were cleaned, they were resampled to reduce the class imbalance problem. As shown in Tables 2 and 3, both datasets have severe class imbalance problems. The ratio of benign samples to malicious samples is around 4:1. Additionally, most malicious samples contributed to less than 1% of the dataset. The benign class and some attack classes were downsampled to make the class distribution more balanced. For the CIC-IDS2017 dataset, attack classes containing more than 100,000 samples were downsampled. After that, the benign class was also downsampled so that the ratio of benign samples to malicious samples was 1:1.

For the CSE-CIC-IDS2018 dataset, the process was very similar, except that all malicious samples were used. Additionally, the columns were renamed to match those of the CIC-IDS2017 dataset. We also removed two columns that only exist in the CIC-IDS2017 dataset. The class distribution of both datasets after cleaning and resampling is shown in Table 2 and Table 3, respectively. Although the class distribution was still uneven after resampling, minority classes were not upsampled as it would cause biases in the models. Instead, the malicious samples of both datasets were relabelled as 'malicious' to prevent the models from overfitting a certain attack class.

Table 2. Class distribution of the CIC-IDS2017 dataset.

Classes	**Before Cleaning**		**After Cleaning & Resampling**	
	No. of Rows	**(%)**	**No. of Rows**	**(%)**
Benign	2,273,097	80.3004%	324,881	50.0000%
DoS Hulk	231,073	8.1629%	100,000	15.3903%
PortScan	158,930	5.6144%	90,694	13.9580%
DDoS	128,027	4.5227%	100,000	15.3903%
DoS GoldenEye	10,293	0.3636%	10,286	1.5830%
FTP-Patator	7938	0.2804%	5931	0.9128%
SSH-Patator	5897	0.2083%	3219	0.4954%
DoS slowloris	5796	0.2048%	5385	0.8288%
DoS Slowhttptest	5499	0.1943%	5228	0.8046%
Bot	1966	0.0695%	1948	0.2998%
Web Attack-Brute Force	1507	0.0532%	1470	0.2262%
Web Attack-XSS	652	0.0230%	652	0.1003%
Infiltration	36	0.0013%	36	0.0055%
Web Attack-Sql Injection	21	0.0007%	21	0.0032%
Heartbleed	11	0.0004%	11	0.0017%

5.2.2. Feature Selection

Feature selection is an important process when training the models using the CIC-IDS2017 dataset, as it includes 78 features. For the CIC dataset experiment, only the CIC-IDS2017 dataset was used for feature selection. We did not inspect the features manually as the dataset included many features.

Table 3. Class distribution of the CSE-CIC-IDS2018 dataset (10% of the entire dataset).

Classes	Before Cleaning No. of Rows	(%)	After Cleaning & Resampling No. of Rows	(%)
Benign	1,347,953	83.0747%	257,791	50.0000%
DDOS attack-HOIC	68,801	4.2402%	68,628	13.3108%
DDoS attacks-LOIC-HTTP	57,550	3.5468%	57,550	11.1621%
DoS attacks-Hulk	46,014	2.8359%	45,691	8.8620%
Bot	28,539	1.7589%	28,501	5.5279%
FTP-BruteForce	19,484	1.2008%	12,368	2.3988%
SSH-Bruteforce	18,485	1.1392%	16,312	3.1638%
Infilteration	16,160	0.9959%	16,034	3.1099%
DoS attacks-SlowHTTPTest	14,110	0.8696%	7251	1.4064%
DoS attacks-GoldenEye	4154	0.2560%	4153	0.8055%
DoS attacks-Slowloris	1076	0.0663%	1049	0.2035%
DDOS attack-LOIC-UDP	163	0.0100%	163	0.0316%
Brute Force -Web	59	0.0036%	59	0.0114%
Brute Force -XSS	25	0.0015%	25	0.0048%
SQL Injection	7	0.0004%	7	0.0014%

When performing feature selection using the random forest algorithm, 10% of the dataset was used to train the model. The features were then ranked according to their respective importance scores calculated during the training process. The ranking of the features is displayed in Figure 9.

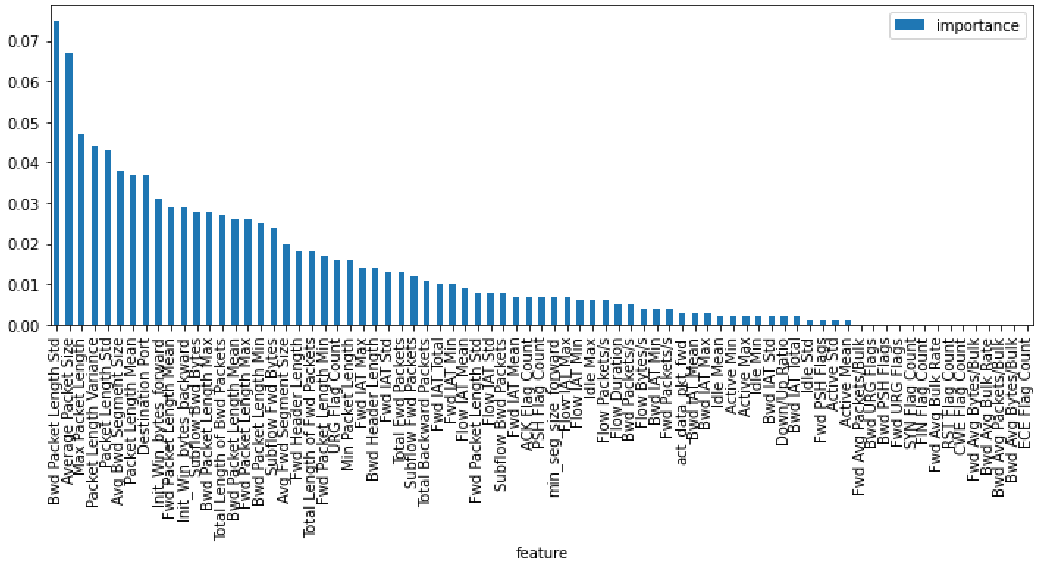


Figure 9. Importance score of each feature in CIC-IDS2017 dataset.

To further reduce the features, the brute force method was used. Preliminary ML models were trained by using the top-ranked features. Starting from Bwd Packet Length Std, one feature was added in each iteration according to their ranking. Figure 10 shows the accuracy of the models as the number of features increases. From the figure, we can see that the top 11 features give the best overall accuracy.

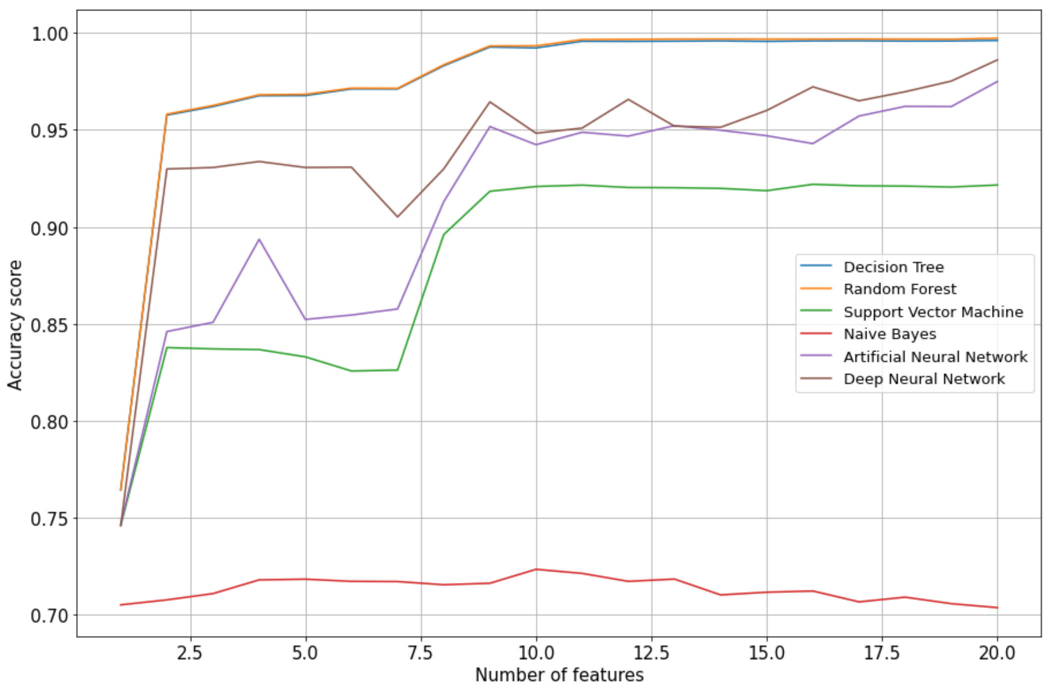


Figure 10. Accuracy of the models with respect to the number of features.

5.2.3. Hyperparameter Optimisation

After the features were selected, the ML models were built, and the hyperparameters of each model were optimised using GridSearchCV, a function provided by scikit-learn. The hyperparameters that were optimised for each model have been discussed in Section 4.3. In this section, the accuracy improvement after the optimisation is discussed. From the brute force feature selection, we could estimate the performance of the models. The DT and RF were expected to achieve 99% accuracy, while ANN and DNN were expected to achieve 95% accuracy. SVM was expected to achieve 92% accuracy, and NB might achieve 72% accuracy. After the optimisation, the accuracy of DT, RF, and NB were almost the same. Hence, optimising the hyperparameter of these models was not very useful. For SVM, the accuracy of the model improved to 96%. The main improvement was mainly brought by the C and γ values. The optimal C was 100, and the optimal γ was 1. For the kernel function, the optimal was the Radial Basis Function (RBF) kernel, which was also the default kernel. It is also worth mentioning that the optimal hyperparameter of SVM was quite consistent, regardless of the change of feature and the sample size of the dataset. For ANN, the accuracy remained the same after the optimisation. This was because the number of neurons chosen during the feature selection was 40, which was very close to the optimum value of 50. For DNN, the optimal hidden layer size was three hidden layers with 15 neurons on each layer. After optimisation, the accuracy of DNN improved to 97%. For both ANN and DNN, the optimum activation function was Tanh, while the optimum solver function was Adam. After the optimisation, we found that most models worked quite well with the default hyperparameters. Hence, optimising the hyperparameters of the models may not be so important, except for ANN and DNN. For ANN and DNN, the hidden layer size still needed optimisation, as the number of neurons and hidden layers will affect the accuracy of the models.

5.2.4. Accuracy Validation

In this step, the accuracy of the models trained using the optimal hyperparameter was validated. The accuracy score of each model was calculated using 5-fold cross-validation. The score of each model in each fold is displayed in Table 4. The table shows that all models achieved higher than 95% of accuracy score except NB. Additionally, the standard deviation of the accuracy of each model was very minimal. As the maximum standard deviation was only 0.0029, the accuracy of each model on the CIC-IDS2017 dataset was very consistent. On the other hand, the NB had the worst performance, with only 73% accuracy. As the original target was to ensure all models achieved at least 90% accuracy, efforts were made to improve the accuracy of NB. For example, the feature selection process was improved by including the brute force method. Additionally, different variances of NB algorithms were tested, including Gaussian naïve Bayes and Bernoulli naïve Bayes. However, the accuracy of NB did not improve much.

Table 4. The accuracy result of 5-fold cross-validation on CIC-IDS2017 dataset.

Model	Fold	Accuracy	Mean Accuracy	Standard Deviation
DT	Fold-1	0.9972	0.9970	0.0002
	Fold-2	0.9967		
	Fold-3	0.9971		
	Fold-4	0.9971		
	Fold-5	0.9970		
RF	Fold-1	0.9968	0.9972	0.0004
	Fold-2	0.9978		
	Fold-3	0.9974		
	Fold-4	0.9968		
	Fold-5	0.9971		
SVM	Fold-1	0.9654	0.9650	0.0008
	Fold-2	0.9641		
	Fold-3	0.9660		
	Fold-4	0.9641		
	Fold-5	0.9655		
NB	Fold-1	0.7313	0.7311	0.0002
	Fold-2	0.7312		
	Fold-3	0.7307		
	Fold-4	0.7311		
	Fold-5	0.7313		
ANN	Fold-1	0.9688	0.9693	0.0029
	Fold-2	0.9678		
	Fold-3	0.9705		
	Fold-4	0.9741		
	Fold-5	0.9653		
DNN	Fold-1	0.9773	0.9772	0.0006
	Fold-2	0.9777		
	Fold-3	0.9777		
	Fold-4	0.9770		
	Fold-5	0.9761		

As the accuracy of the model was validated using cross-validation, the accuracy of each model was compared against two references, including the work carried out by Kostas [42] and the work by Vinayakumar et al. [29]. The reason for choosing these two works was that both studies used the CIC-IDS2017 dataset, and most models used by this paper were included in their studies. Additionally, both works included binary classification, where the data were only classified as benign or malicious. Moreover, Kostas also performed feature selection using the random forest algorithm; hence, the work is comparable to ours.

Table 5 shows that the models implemented in this paper are comparable to other models. The models using DT, RF, and SVM have a clear lead compared to the other existing works. The accuracy result of NB is between the two studies. The NB implemented by Kostas [42] performed significantly better than that of this paper. In contrast, the NB implemented by Vinayakumar et al. [29] had much poorer accuracy than ours.

Table 5. Comparison of accuracy (rounded to two decimal) for the CIC dataset.

ML Models	Accuracy		
	This Work	**Kostas [42]**	**Vinayakumar et al. [29]**
Decision Tree	1.00	0.95	0.94
Random Forest	1.00	0.94	0.94
Support Vector Machine	0.96	-	0.80
Naïve Bayes	0.73	0.87	0.31
Artificial Neural Network	0.95	0.97	0.96
Deep Neural Network	0.97	-	0.94

The difference is due to the fact that Kostas performed model-specific feature selection; therefore, the features used to train each model in Kostas's work were different. With such optimisation, the NB is able to achieve significantly better accuracy. On the other hand, the accuracies of the models implemented by Vinayakumar et al. are lower than those of this work, except for ANN. This is because the focus of Vinayakumar et al. was on proposing an optimised DNN that is less costly to train. Other models implemented by Vinayakumar et al. were just to provide a reference on how their proposed models perform. Hence, they may not have performed optimisations to achieve the maximum possible accuracy. Additionally, it is important to note that the ANN and DNN of this paper were compared against the DNN proposed by Vinayakumar et al. with one and three hidden layers, respectively. Apart from these, Hossain et al. [10] also applied RF to the CICIDS2017 dataset, and the training accuracy reported in their work was similar to the one reported in this work. However, Hossain et al.'s work split the same dataset to generate the training and testing dataset randomly, which does not reflect the changes in the dataset (network environment) over a period of time. Therefore, the context of our work is different compared to Hossain et al.'s work and provides a better understanding of the performance as we evaluated the testing accuracy using a progressive dataset. Overall, the models implemented in this paper achieved better accuracy than the other literature and provide better insight into the performance of ML models with progressive datasets.

5.2.5. Model Evaluation

After ensuring that the models have achieved good accuracy, the models were tested using the testing dataset, the CSE-CIC-IDS2018 dataset. As described in Section 4.5, the models were trained again using the optimal hyperparameters and tested again using the training dataset. This step was necessary to evaluate the performance of the models on the training dataset. The accuracy and the F1-score of the models on the training dataset are displayed in Table 6. The accuracies of the models were also visualised using the confusion matrixes as shown in Figure 11. The confusion matrix clearly shows that DT and RF have the best accuracy, followed by SVM, ANN, and DNN, while NB is biased towards the benign class.

The most interesting part of this paper is the evaluation of the models using another dataset. Table 7 shows the performance of the models on the CSE-CIC-IDS2018 dataset. The accuracy of each model was also visualised using confusion matrixes as shown in Figure 12. It is important to note that the models were not trained using the CSE-CIC-IDS2018 dataset. The confusion matrixes show that all models have poor performance except SVM. Looking at the accuracy score and F1-score, SVM has the highest accuracy, with 76% accuracy, followed by ANN, with 70% accuracy. The confusion matrixes also tell the problem of the models—bias towards the benign samples. In other words, the models have overfitted the malicious samples to different extents. Among them, SVM and ANN suffer less from overfitting. From Figure 12, we can see that both models allow more error, which incorrectly classifies more benign samples as malicious. Such behaviour is reflected in the recall score of the benign class. The recall scores of both models are around 0.9, which is the lowest among all implemented models. In return, both models correctly classify more malicious samples and yield a much higher recall score than other models on the malicious class.

Table 6. Performance of the models on the CIC-IDS2017 dataset.

Models	Evaluation Metrics				Class
	Accuracy	Precision	Recall	F1-Score	
DT	0.9959	0.9987	0.9932	0.9959	benign
		0.9931	0.9987	0.9959	malicious
RF	0.9967	0.9990	0.9945	0.9967	benign
		0.9944	0.9990	0.9967	malicious
SVM	0.9600	0.9923	0.9278	0.9590	benign
		0.9312	0.9927	0.9610	malicious
NB	0.7296	0.6576	0.9671	0.7829	benign
		0.9360	0.4884	0.6418	malicious
ANN	0.9549	0.9831	0.9264	0.9539	benign
		0.9294	0.9839	0.9558	malicious
DNN	0.9735	0.9930	0.9542	0.9732	benign
		0.9552	0.9932	0.9738	malicious

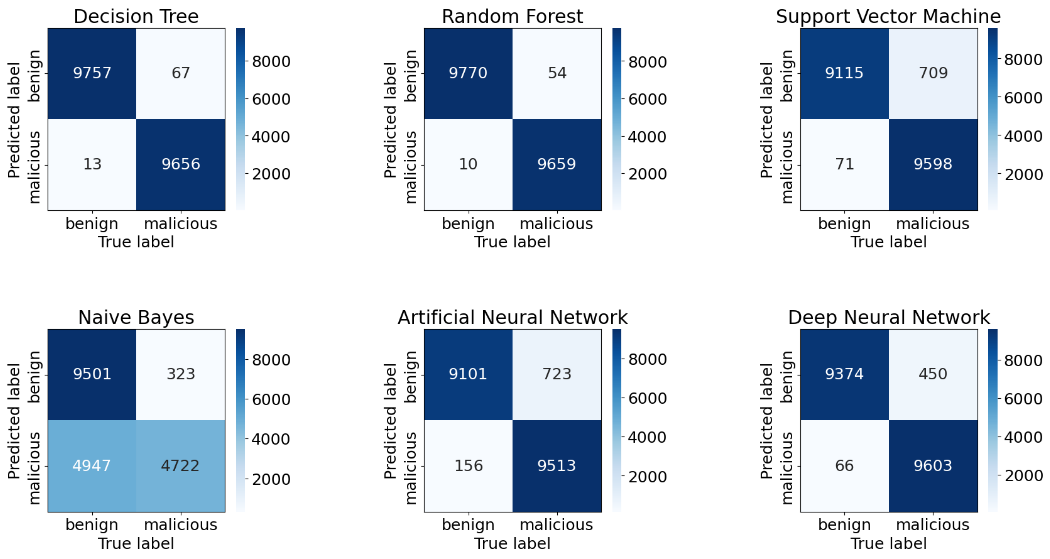


Figure 11. Confusion matrix of each model on CIC-IDS2017 dataset.

Table 7. Performance of the models on the CSE-CIC-IDS2018 dataset.

Models	Evaluation Metrics				Class
	Accuracy	Precision	Recall	F1-Score	
DT	0.5942	0.5546	0.9660	0.7046	benign
		0.8660	0.2208	0.3518	malicious
RF	0.5949	0.5550	0.9668	0.7052	benign
		0.8690	0.2214	0.3529	malicious
SVM	0.7559	0.6977	0.9049	0.7879	benign
		0.8639	0.6063	0.7125	malicious
NB	0.4972	0.4992	0.9920	0.6641	benign
		0.0417	0.0003	0.0007	malicious
ANN	0.7000	0.6428	0.9031	0.7510	benign
		0.8360	0.4959	0.6226	malicious
DNN	0.6518	0.5977	0.9335	0.7288	benign
		0.8468	0.3689	0.5139	malicious

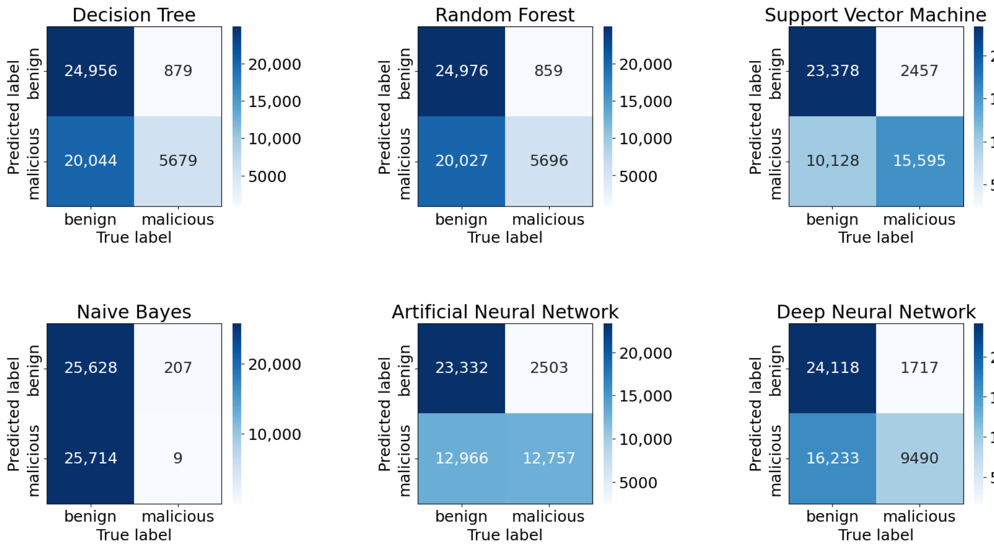
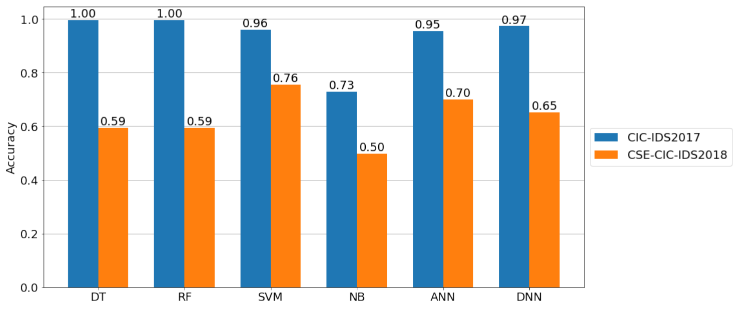
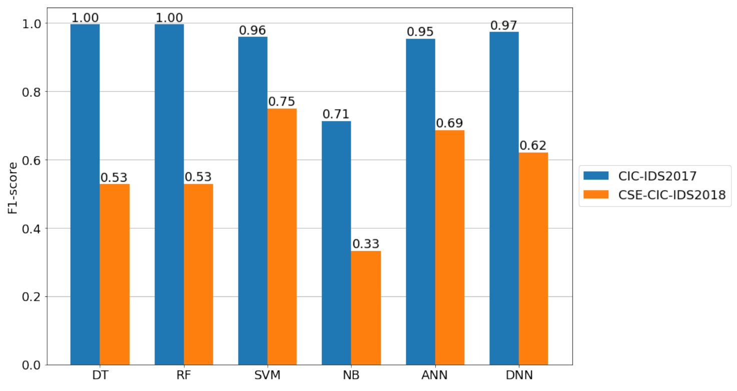


Figure 12. Confusion matrix of each model on CSE-CIC-IDS2018 dataset.

We used the data presented in Figure 13a,b to compare which model suffers the most from overfitting. Figure 13a compares the accuracy of the models on different datasets while Figure 13b compares the F1-score. From Figure 13a, we can see that SVM has the smallest drop in terms of accuracy, followed by ANN and NB. However, it is important to note that NB has a significant drop in terms of the F1-score, as shown in Figure 13b. For DT and RF, the accuracy and F1-score of both models drop significantly on the testing dataset. As a result, SVM and ANN suffer less from overfitting, while DT and RF suffer the most.



(**a**) Accuracy of the models on the CIC's dataset

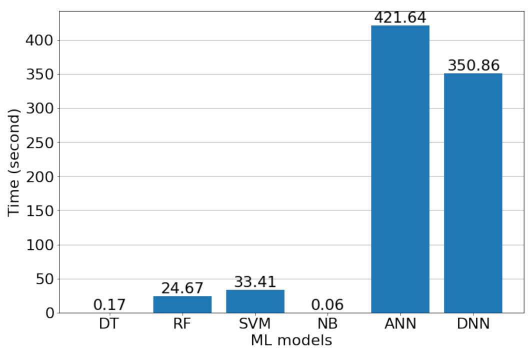


(**b**) F1-score of the models on CIC's dataset

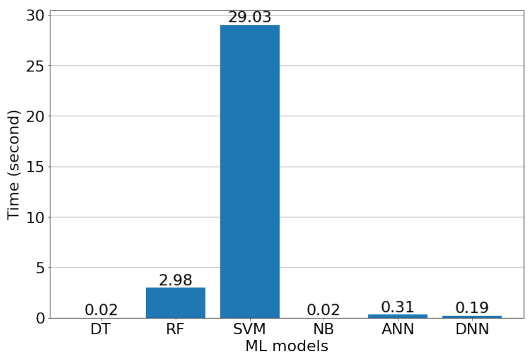
Figure 13. Accuracy and F1-score of the models on the CIC's dataset.

In terms of time consumption, ANN and DNN are costly to train. As shown in Figure 14a, the training time of ANN and DNN is significantly longer than the other models implemented in this work. However, in terms of time consumption on classifying the samples, the time consumed by ANN and DNN is greatly reduced. As shown in Figure 14b, SVM consumes significantly more time to predict the class of the samples. On the other hand, the time consumed by ANN and DNN to classify the samples is even less than that of the RF. DT and NB are the most efficient models in terms of time consumption for both training and predicting the class of the samples.

When considering both the performance and the computation time, ANN is the best model. Despite being the most time-consuming model to train, the long-term performance might be worth the trade-off. Although the SVM is more efficient to train and has higher accuracy than ANN, the prediction time is much higher. Considering that an IDS might need to detect intrusions in real-time, ANN is the best overall model.



(**a**) Training time on CIC-IDS2017 dataset



(**b**) Prediction time on CSE-CIC-IDS2018 dataset

Figure 14. Time consumption for training and prediction on the CIC dataset.

5.3. Experiment Using LUFlow Dataset

This section discusses the experiment details using the LUFlow dataset. For the experiment using the LUFlow dataset, the data collected in July 2020 were used as the training dataset, while the data collected in January 2021 were used as the testing dataset. For the rest of this paper, the dataset is referred to by the year in which the data were collected. Hence, the training dataset is also referred to as the LUFlow 2020 dataset, while the testing dataset is referred to as the LUFlow 2021 dataset.

5.3.1. Dataset Pre-Processing

As the LUFlow dataset is organised according to the day on which the data were collected, the first step was to combine the files of each day into one single file for processing. However, the datasets are huge in size; the LUFlow 2020 dataset contains 3.7 GB of data, while the LUFlow 2021 includes 2.5 GB of data. Hence, only 10% of the samples were randomly selected for this work. We discovered that both datasets contain a small number of missing values and duplicated samples. Additionally, no entry contains infinity values in both datasets. These unwanted entries were dropped.

We also found that the datasets are slightly imbalanced. As shown in Table 8, the benign samples account for 56% of the dataset. In contrast, the malicious traffic only accounts for 36% of the dataset. Hence, benign samples were randomly selected so that the ratio of benign samples to malicious samples was 1:1. Moreover, samples in the class “outlier” were removed from the dataset as they were noise to the ML models. The class distribution of the datasets after cleaning and re-sampling is shown in Table 9.

Table 8. Class distribution of LUFlow dataset before cleaning.

Classes	LUFlow 2020		LUFlow 2021	
	No. of Rows (10%)	**Percentage**	**No. of Rows (10%)**	**Percentage**
benign	1,396,168	55.71%	1,638,952	56.36%
malicious	905,395	36.12%	591,372	35.52%
outlier	204,787	8.17%	469,345	8.12%

Table 9. Class distribution of LUFlow dataset after cleaning.

Classes	LUFlow 2020		LUFlow 2021	
	No. of Rows	**Percentage**	**No. of Rows**	**Percentage**
benign	879,740	50%	569,003	50%
malicious	879,740	50%	569,003	50%

5.3.2. Feature Selection

The LUFlow dataset is not large in terms of the number of features. The dataset only contains 16 features, where the description of each feature is listed in Table 10. Although there are only a few features, feature selection was still necessary as it would help remove noise from the dataset and improve the performance of the models.

Table 10. Description of features of LUFlow dataset. Retrieved from reference [20].

Feature	Description
src_ip	The source IP address associated with the flow. This feature is anonymised.
src_port	The source port number associated with the flow.
dest_ip	The destination IP address associated with the flow. The feature is also anonymised.
dest_port	The destination port number associated with the flow.
protocol	The protocol number associated with the flow. For example, TCP is 6
bytes_in	The number of bytes transmitted from source to destination.
bytes_out	The number of bytes transmitted from destination to source.
num_pkts_in	The packet count from source to destination.
num_pkts_out	The packet count from destination to source.
entropy	The entropy in bits per byte of the data fields within the flow. This number ranges from 0 to 8.
total_entropy	The total entropy in bytes over all of the bytes in the data fields of the flow.
mean_ipt	The mean of the inter-packet arrival times of the flow.
time_start	The start time of the flow in seconds since the epoch.
time_end	The end time of the flow in seconds since the epoch.
duration	The flow duration time, with microsecond precision.
label	The label of the flow, as decided by Tangerine. Either benign, outlier, or malicious.

We first removed some features manually before using the random forest to select the features. First, the 'src_ip' and the 'dest_ip' columns were removed, as using the IP address to classify the traffics may cause overfitting in the long run. Additionally, the 'time_start' and the 'time_end' columns were also removed. The two columns recorded the start and end time using Unix timestamp, which kept increasing over time. The 'duration' column represents the time between 'time_start' and 'time_end'. Hence, these two columns could be safely removed.

After removing these four features, the remaining features were ranked according to their importance score computed by random forest. The ranking of the features was visualised using Figure 15. From Figure 15, 'dest_port' has the highest score, and its score is significantly higher than the 'bytes_out' feature. The importance scores of the last five features are substantially lower than the higher-ranked features. The low importance score indicates that those features do not provide much information to classify the data.

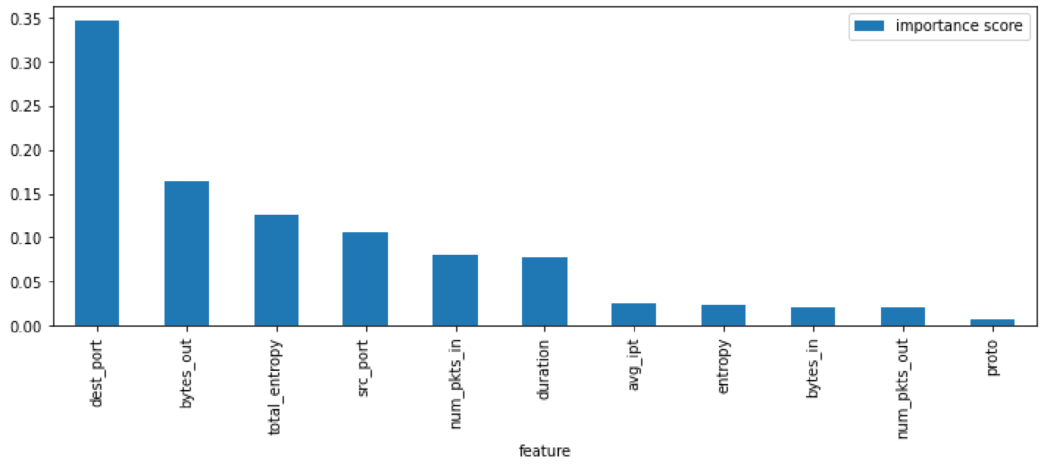


Figure 15. Importance score of each feature in LUFlow dataset.

Finally, the features were further reduced using brute force. For implementation using the LUFlow dataset, all features were tested using brute force. Figure 16 shows the accuracy of the models with respect to the number of features. The figure shows that naïve Bayes reach their maximum accuracy when the top two to six features were used. On the other hand, SVM, ANN, and DNN achieved high accuracy when at least the top six features were included. For DT and RF, the accuracy was very high, even with only one feature. As a result, the top six features were chosen in the final feature set.

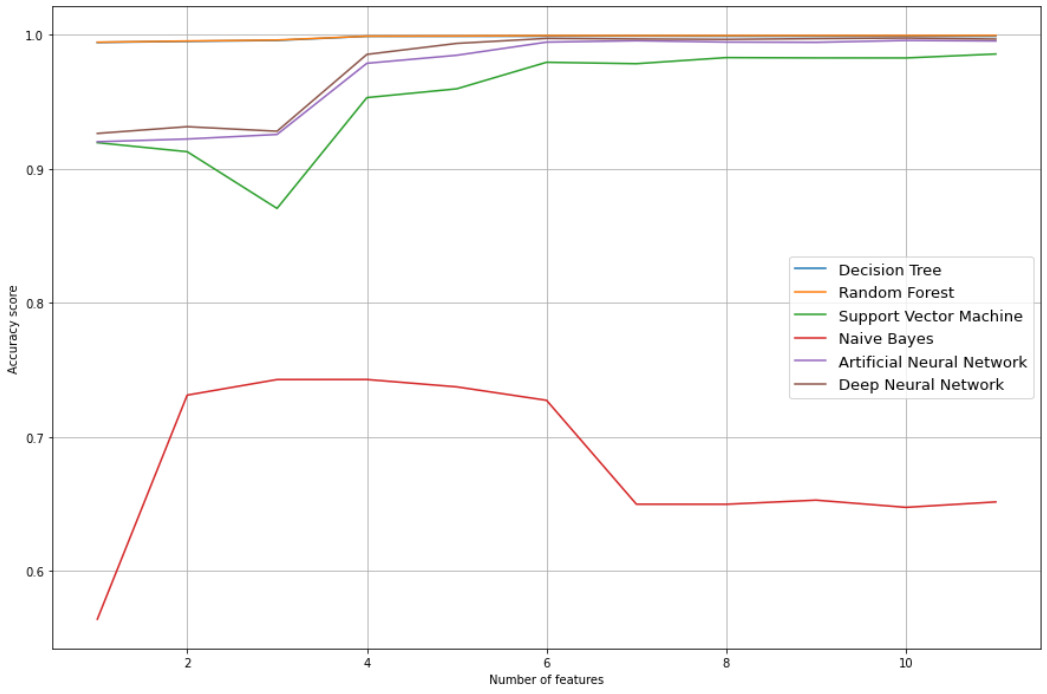


Figure 16. Accuracy of the models with respect to the number of features in the LUFlow dataset.

5.3.3. Hyperparameter Optimisation

The third step of the experiment was optimising the hyperparameter of the models. The hyperparameters that were optimised for each model were described in Section 4.3. From Figure 16, we can see that the ML models are able to retrieve good accuracy when the top six features are used. Hence, the improvement from the hyperparameter optimisation is minimum. SVM is the model that receives the biggest improvement, where the accuracy of the model has improved to 99%. The optimal hyperparameters of SVM on the LUFlow dataset are the same as on the CIC-IDS2017 dataset, where the optimal C is 100, the optimal γ is 1, and the optimal kernel function is RBF. For ANN and DNN, the optimal number of neurons is slightly less than that on the CIC-IDS2017 dataset. The ANN reaches its optimum when the number of neurons on its hidden layer is 40. For DNN, the optimal hidden layer size is three hidden layers with 10 neurons on each layer. The optimisation result on the LUFlow dataset once again shows that optimising the hyperparameters of the models is not very important.

5.3.4. Accuracy Validation

After optimising the hyperparameter of the models, it is important to check if the models are over-optimised, which will result in overfitting. We used five-fold cross-validation to validate the accuracies of the models. The accuracy of the models in each fold is listed in Table 11. We can see from Table 11 that all models achieved excellent accuracies, except NB. The standard deviation of the accuracy of each model is very minimal, indicating that the accuracies of the models are very consistent. Looking at the accuracy score, DT and RF are the best models with an average of 99.94% accuracy. The worst-performing model is NB, with only 72% accuracy, making it the least reliable model. As in the previous experiment, efforts have been made to improve the performance of NB. However, the accuracy of NB hardly improves any further.

Table 11. The accuracy result of 5-fold cross-validation on LUFlow dataset.

Model	Fold	Accuracy	Mean Accuracy	Standard Deviation
DT	Fold-1	0.9995	0.9994	0.0001
	Fold-2	0.9993		
	Fold-3	0.9994		
	Fold-4	0.9994		
	Fold-5	0.9995		
RF	Fold-1	0.9995	0.9959	0.0001
	Fold-2	0.9994		
	Fold-3	0.9995		
	Fold-4	0.9996		
	Fold-5	0.9995		
SVM	Fold-1	0.9961	0.9959	0.0003
	Fold-2	0.9960		
	Fold-3	0.9962		
	Fold-4	0.9956		
	Fold-5	0.9954		
NB	Fold-1	0.7201	0.7198	0.0026
	Fold-2	0.7154		
	Fold-3	0.7210		
	Fold-4	0.7234		
	Fold-5	0.7190		
ANN	Fold-1	0.9953	0.9956	0.0003
	Fold-2	0.9960		
	Fold-3	0.9959		
	Fold-4	0.9956		
	Fold-5	0.9952		
DNN	Fold-1	0.9984	0.9984	0.0002
	Fold-2	0.9983		
	Fold-3	0.9983		
	Fold-4	0.9987		
	Fold-5	0.9986		

Compared with the previous experiment, the performances of DT, RF, and NB on the LUFlow dataset are aligned with those achieved using the CIC-IDS2017 dataset. On the other hand, SVM, ANN, and DNN have great performance on the LUFlow dataset as well, with accuracies above 99.5%.

As the standard deviation of the accuracy is very small for each model, we can conclude that the hyperparameters of each model are well-optimised. To further verify it, we compared the mean accuracy of each model with the work carried out by Mills [46]. As shown in Table 12, the accuracy of each model in this work is higher than that implemented by Mills. With the comparison, we can also verify that the NB in this work has been well-optimised despite being the worst compared to other models.

Table 12. Comparison of accuracy for the LUFlow dataset.

ML Models	**Accuracy**	
	This Work	**Mills [46]**
Decision Tree	0.9994	0.9298
Random Forest	0.9959	0.9427
Support Vector Machine	0.9959	0.6423
Naïve Bayes	0.7198	0.3812
Artificial Neural Network	0.9956	-
Deep Neural Network	0.9984	-

5.3.5. Model Evaluation

Since we ensured that the models were able to provide good accuracy by using cross-validation, we then tested the model using the testing dataset, the LUFlow 2021 dataset. As described in Section 4.5, the final ML models were trained using the LUFlow 2020 dataset with the optimal hyperparameters. As the size of the datasets is huge, only 20% of the LUFlow 2020 and the LUFlow 2021 datasets was used in our experiments. We used 70% of the LUFlow 2020 dataset for training the model; the other 30% was used to evaluate the performance of the models on the training dataset. The performance of the models on the training dataset is listed in Table 13 and visualised using confusion matrixes in Figure 17.

From Figure 17, we see that most models have achieved a good accuracy except NB. The problem with NB is that it overfits the benign samples, which is reflected in the recall score of NB on the malicious sample, with a score of only 0.4732. In other words, NB correctly classifies less than half of the benign samples. For other ML models, the precision and recall scores are very similar, indicating that they do not bias towards a specific class.

Table 13. Performance of the models on LUFlow 2020 dataset.

Models	**Evaluation Metrics**				**Class**
	Accuracy	**Precision**	**Recall**	**F1-Score**	
DT	0.9994	0.9996 0.9993	0.9993 0.9996	0.9994 0.9994	benign malicious
RF	0.9994	0.9995 0.9993	0.9993 0.9995	0.9994 0.9994	benign malicious
SVM	0.9956	0.9954 0.9958	0.9957 0.9954	0.9956 0.9956	benign malicious
NB	0.7276	0.9600 0.6518	0.4732 0.9804	0.6339 0.7830	benign malicious
ANN	0.9960	0.9953 0.9967	0.9967 0.9953	0.9960 0.9960	benign malicious
DNN	0.9982	0.9978 0.9985	0.9985 0.9978	0.9982 0.9982	benign malicious

The performance of the models on the LUFlow 2021 dataset is very interesting, as the result is opposite to that of our first experiments on the CIC dataset. As shown in Figure 18, notice that the models still perform very well on the LUFlow 2021 dataset. From the accuracy score of the models (see Table 14), we observe that the accuracy scores of most models do slightly decrease. For example, the accuracy score of the DT has fallen from 0.9994 to 0.9990, which is just a marginal drop. However, the recall score of NB on benign class has increased from 0.4732 to 0.5046, which is very surprising.

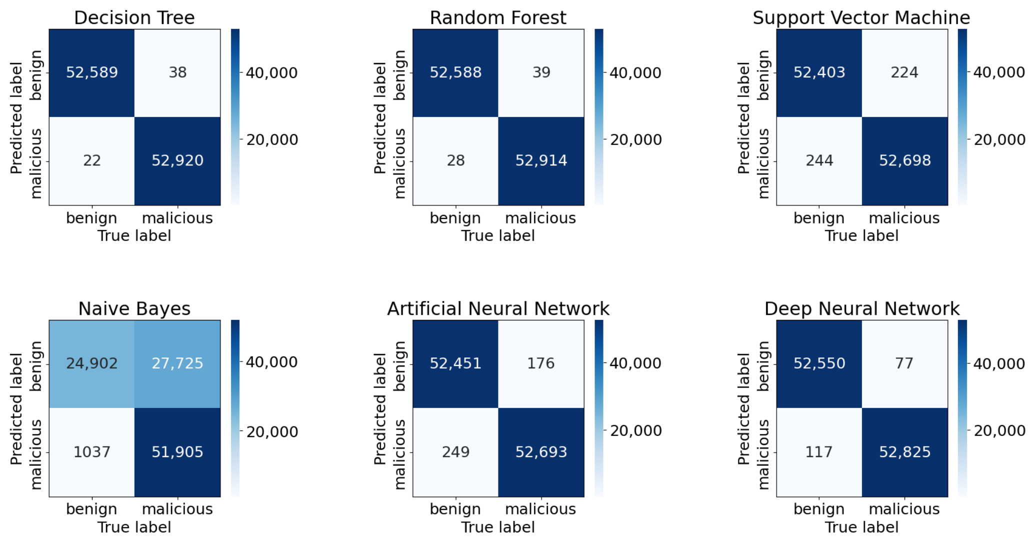


Figure 17. Confusion matrix of each model on LUFlow 2020 dataset.

Table 14. Performance of the models on the LUFlow 2021 dataset.

Models	**Accuracy**	**Evaluation Metrics** **Precision**	**Recall**	**F1-Score**	**Class**
DT	0.9990	0.9994	0.9986	0.9990	benign
		0.9986	0.9994	0.9990	malicious
RF	0.9991	0.9995	0.9986	0.9991	benign
		0.9986	0.9995	0.9990	malicious
SVM	0.9934	0.9882	0.9989	0.9935	benign
		0.9988	0.9880	0.9934	malicious
NB	0.7377	0.9477	0.5046	0.6585	benign
		0.6612	0.9720	0.7870	malicious
ANN	0.9930	0.9874	0.9988	0.9931	benign
		0.9988	0.9872	0.9930	malicious
DNN	0.9975	0.9955	0.9996	0.9975	benign
		0.9996	0.9955	0.9975	malicious

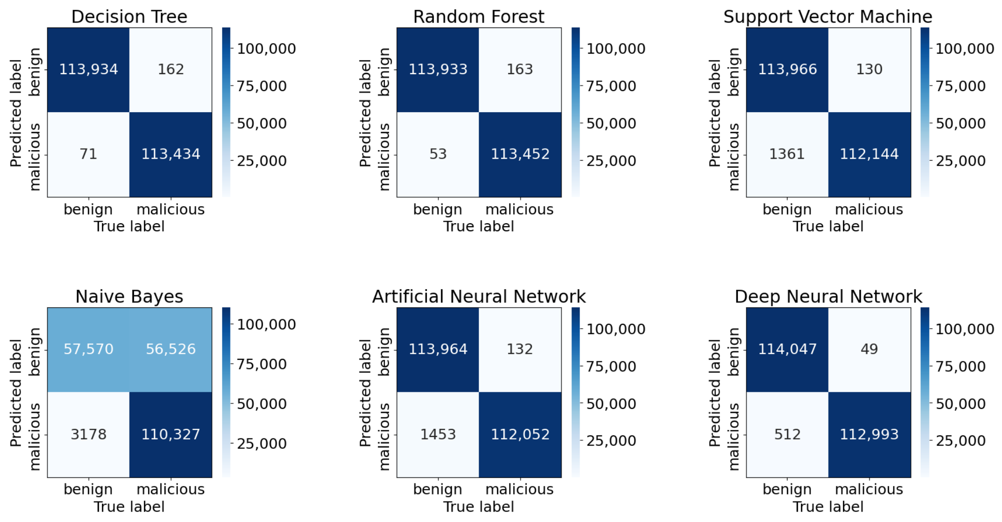
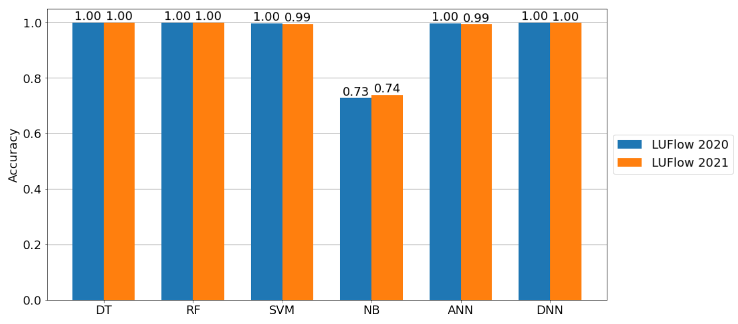
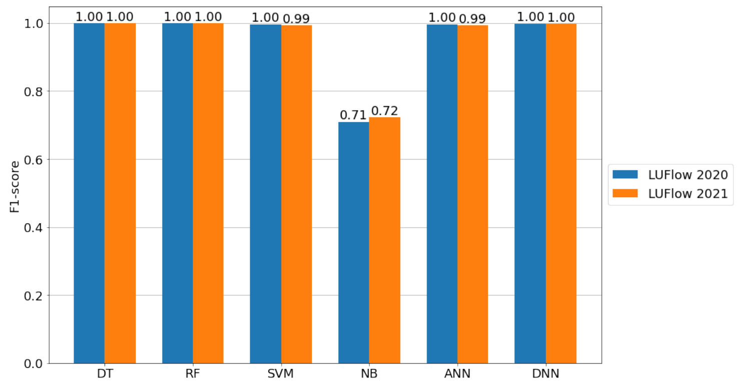


Figure 18. Confusion matrix of each model on LUFlow 2021 dataset.

Figure 19a,b provide further comparisons on the performance of the models on the training and testing dataset. From Figure 19a, we observe that the accuracy of SVM and ANN dropped the most. Comparing the accuracy score shown in Tables 13 and 14, we see that the accuracy scores of SVM and ANN have dropped more than 0.002. Additionally, the precision scores of both models on the benign class of the LUFlow 2021 dataset are lower than their recall score (the difference is about 0.01). The decrease in precision score may indicate that the models have started to bias towards the benign class. However, since the accuracy scores and the F1-scores of both models are still high, we cannot conclude that the models have started to bias towards a specific class.



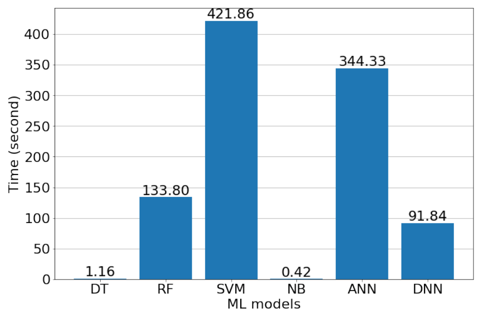
(**a**) Accuracy of the models on LUFlow dataset



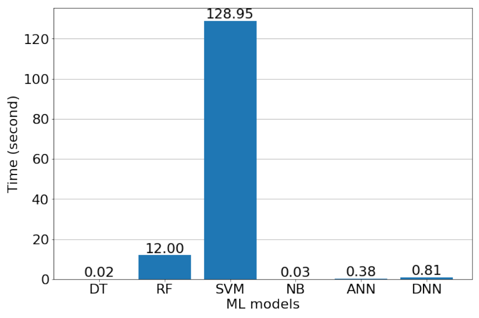
(**b**) F1-score of the models on LUFlow dataset

Figure 19. Accuracy and F1-score of the models on the LUFlow dataset.

In terms of time consumption for training, SVM and ANN are the most expensive models, as shown in Figure 20a. On the other hand, DT and NB are still the most efficient models. Interestingly, DNN consumes less time to train than RF, SVM, and ANN. Two reasons contribute to the reduction in training time: first, the number of neurons on each layer for the DNN is reduced to 10, compared to 15 neurons in the experiment using the CIC's datasets. The CIC's dataset is also wide, while the LUFlow dataset is long. In other words, the CIC's datasets contain significantly more features than the LUFlow dataset, while the LUFlow dataset contains considerably more samples than the CIC's dataset. Hence, the input layer contains fewer neurons. Additionally, the time complexity of each model is different. In terms of the time consumption for prediction, Figure 20b shows that most models perform efficiently, while SVM consumes significantly more time than others. When taking the performance of the models into consideration, DT is the best overall model in this experiment. Except for NB, the performance of each model is equally good. Hence, the high training time of other models is not worth it.



(**a**) Training time on LUFlow dataset.



(**b**) Prediction time on LUFlow dataset.

Figure 20. Time consumption for training and prediction on LUFlow dataset.

6. Discussion of Results

We conducted the experiment twice in this work, once using the CIC's datasets and the other time using the LUFlow dataset. Interestingly, the two experiments show two different results. The experiment results using the CIC's datasets show that all models suffered from different degrees of overfitting. On the other hand, the experiment results using the LUFlow dataset show that the models do not have an overfitting problem. This difference comes from the fact that the two experiments reflect two different scenarios.

The CIC's datasets reflect a scenario where the environment used by the victims and the attackers has become more complex, and attackers are trying to break into the system using different tools and techniques. Hence, the deviation between the CIC-IDS2017 and the CSE-CIC-IDS2018 dataset is large. First, the CSE-CIC-IDS2018 dataset was created with more machines and various operating systems. Additionally, the class distributions of the CIC-IDS2017 and the CSE-CIC-IDS2018 dataset are different. For example, port scan attacks account for 14% of the CIC-IDS2017 dataset, but there is no port scan attack in the CSE-CIC-IDS2018 dataset. On the other hand, bot attacks and infiltration increase from 0.30% and 0.01% to 6% and 3%, respectively, in the CSE-CIC-IDS2018 dataset. The changes in the network environment and the type of attacks cause the models to perform relatively poorly on the CSE-CIC-IDS2018 dataset. Overall, we found that DT and RF suffer a lot from overfitting; SVM and ANN are less prone to overfitting in the scenario where the network infrastructures of the victims and the attackers have changed drastically.

On the other hand, the LUFlow dataset reflects a scenario with minimum changes in the environment used by attackers and no changes in the environment used by the victims. Additionally, Lancaster University's public address space may suffer fewer targeted attacks. Moreover, the type of attacks it faces may not change much within six months; we used data collected in July 2020 for the training dataset and used data collected in January 2021 for testing, which are six months apart. That is, this experiment mimics the real-world scenario of a small organisation over six months. From the result that we get from the second experiment, we have found that no ML models suffer from overfitting. In other words, all models implemented by us can maintain their accuracy for six months in the use case of a small organisation. However, it is important to note that the LUFlow dataset contains 8% of samples that are classified as an outlier. There may be some unseen attacks in the outlier class, but they are not correctly classified in the dataset.

The experiment results from the first set of experiments have shown that the method proposed in this paper has a better chance of detecting overfitting. From our experiment, we have shown that the models perform very well on the training dataset. We even evaluated the performance of the models on the training dataset using cross-validation, and there is no sign of overfitting. However, when we evaluated the models using the testing dataset, overfitting of the models was discovered. On the other hand, models implemented in other literature may also suffer the overfitting problem, but it will not be discovered as other literature does not include a second dataset to evaluate the model.

Our experiment results have also shown that ANN is the best model for a system for which the infrastructures are updated frequently, and the cost of cyberattacks is very high. The ANN has a balance between resistance to overfitting and the time consumed for prediction. In contrast, DT is a better choice for a system that does not frequently update its infrastructure and does not suffer from massive attacks. This is because DT is one of the most efficient models at training and classifying new samples. Additionally, DT also provides the best accuracy on the training dataset in both experiments. It is also important to note that, in the first set of experiments, the accuracy of the models on the testing dataset is lower than 80%, which may be unacceptable for an IDS. Hence, periodically updating the IDS with newer data is necessary, regardless of the ML model being used.

7. Conclusions

We proposed a new method to evaluate the long-term performance of a machine learning (ML)-based intrusion detection system (IDS). Our proposed method uses a dataset created later than the training dataset to evaluate the ML models. To the best of our knowledge, this is the first work that has trained and evaluated the ML-based IDS using separate datasets. We identified two sets of suitable datasets, the CIC dataset and the LUFlow dataset, to conduct our experiments using six ML models: decision tree (DT), random forest (RF), support vector machine (SVM), naïve Bayes (NB), artificial neural network (ANN), and deep neural network (DNN). We presented a comprehensive set of experiments and offered valuable insights into the most suitable models for different organisational contexts. From our experimental results, we concluded that ANN is the best model if long-term performance is important. On the other hand, DT is more suitable for small organisations that are less targeted by attacks. The experimental results also showed that our proposed method can detect overfitting better. It is important to note that models proposed in other existing works may also suffer from the overfitting problem that occurred in our first experiment. However, the problem may not be discovered as the other existing works did not include a second dataset to evaluate their models.

In this study, we focused on proposing a new method to evaluate ML-based IDS. Therefore, we selected conventional ML models and implemented them using the existing library so that more effort could be put into the experiment itself. Despite simple models being used, our experiments were able to show the long-term performance differences between various models. In the future, we aim to evaluate unsupervised ML models and other more comprehensive models proposed by other authors. For example, we can evaluate one-class SVM using our proposed method, as Hindy et al. [12] have shown that one-class SVM has good accuracy in detecting zero-day attacks. Additionally, we can improve this work by using multi-classification instead of binary classification. Moreover, future work may also aim to improve the feature selection method used in this paper. The selected features can greatly impact the performance of the models. Hence, it is possible that the overfitting observed in our first experiment can be improved by feature selection.

Author Contributions: Conceptualization, T.-H.C. and I.S.; methodology, T.-H.C.; implementation, T.-H.C.; validation, T.-H.C.; formal analysis, T.-H.C. and I.S.; writing—original draft preparation, T.-H.C.; writing—review and editing, I.S.; supervision, I.S.; project administration, I.S.; funding acquisition, I.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Xiamen University Malaysia Research Fund under grant XMUMRF/2022-C9/IECE/0032.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The experiment details of this paper and the source codes are made publicly accessible via a GitHub repository: https://github.com/tuanhong3498/Evaluation-of-Machine-Learning-Algorithm-in-Network-Based-Intrusion-Detection-System (accessed on 25 May 2023).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. MonsterCloud. Top Cyber Security Experts Report: 4000 Cyber Attacks a Day Since COVID-19 Pandemic. Available online: https://www.prnewswire.com/news-releases/top-cyber-security-experts-report-4-000-cyber-attacks-a-day-since-covid-19-pandemic-301110157.html (accessed on 25 May 2023).
2. Du, D.; Zhu, M.; Li, X.; Fei, M.; Bu, S.; Wu, L.; Li, K. A Review on Cybersecurity Analysis, Attack Detection, and Attack Defense Methods in Cyber-Physical Power Systems. *J. Mod. Power Syst. Clean Energy* **2022**, *11*, 727–743. [CrossRef]
3. Liao, H.J.; Lin, C.H.R.; Lin, Y.C.; Tung, K.Y. Intrusion detection system: A comprehensive review. *J. Netw. Comput. Appl.* **2013**, *36*, 16–24. [CrossRef]
4. Khraisat, A.; Gondal, I.; Vamplew, P. An anomaly intrusion detection system using C5 decision tree classifier. In *Trends and Applications in Knowledge Discovery and Data Mining*; Springer: Cham, Switzerland, 2018; pp. 149–155. [CrossRef]
5. Kreibich, C.; Crowcroft, J. Honeycomb: Creating intrusion detection signatures using honeypots. *ACM SIGCOMM Comput. Commun. Rev.* **2004**, *34*, 51–56. [CrossRef]
6. García-Teodoro, P.; Díaz-Verdejo, J.; Maciá-Fernández, G.; Vázquez, E. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Comput. Secur.* **2009**, *28*, 18–28. [CrossRef]
7. Khan, M.A.; Karim, M.R.; Kim, Y. A Scalable and Hybrid Intrusion Detection System Based on the Convolutional-LSTM Network. *Symmetry* **2019**, *11*, 583. [CrossRef]
8. Sarnovsky, M.; Paralic, J. Hierarchical Intrusion Detection Using Machine Learning and Knowledge Model. *Symmetry* **2020**, *12*, 203. [CrossRef]
9. Wang, C.; Sun, Y.; Wang, W.; Liu, H.; Wang, B. Hybrid Intrusion Detection System Based on Combination of Random Forest and Autoencoder. *Symmetry* **2023**, *15*, 568. [CrossRef]
10. Hossain, F.; Akter, M.; Uddin, M.N. Cyber Attack Detection Model (CADM) Based on Machine Learning Approach. In Proceedings of the 2021 2nd International Conference on Robotics, Electrical and Signal Processing Techniques (ICREST), Dhaka, Bangladesh, 5–7 January 2021; pp. 567–572. [CrossRef]
11. Kaspersky. What Is a Zero-day Attack?—Definition and Explanation. Available online: https://www.kaspersky.com/resource-center/definitions/zero-day-exploit (accessed on 25 May 2023).
12. Hindy, H.; Atkinson, R.; Tachtatzis, C.; Colin, J.N.; Bayne, E.; Bellekens, X. Utilising deep learning techniques for effective zero-day attack detection. *Electronics* **2020**, *9*, 1684. [CrossRef]
13. Shaukat, K.; Luo, S.; Varadharajan, V.; Hameed, I.A.; Xu, M. A Survey on Machine Learning Techniques for Cyber Security in the Last Decade. *IEEE Access* **2020**, *8*, 222310–222354. [CrossRef]
14. Stephen, D.B.; Dennis, K.; Michael, J.P.; Padhraic, S. The UCI KDD archive of large data sets for data mining research and experimentation. *SIGKDD Explor. Newsl.* **2000**, *2*, 81–85. [CrossRef]
15. Sharafaldin, I.; Lashkari, A.H.; Ghorbani, A.A. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP 2018), Funchal, Portugal, 22–24 January 2018; Volume 1, pp. 108–116. [CrossRef]
16. Intrusion Detection Evaluation Dataset (CIC-IDS2017). Available online: https://www.unb.ca/cic/datasets/ids-2017.html (accessed on 26 May 2023).
17. Thakkar, A.; Lohiya, R. A review of the advancement in intrusion detection datasets. *Procedia Comput. Sci.* **2020**, *167*, 636–645. [CrossRef]
18. A Realistic Cyber Defense Dataset (CSE-CIC-IDS2018). Available online: https://registry.opendata.aws/cse-cic-ids2018 (accessed on 26 May 2023).
19. CSE-CIC-IDS2018 on AWS. Available online: https://www.unb.ca/cic/datasets/ids-2018.html (accessed on 2 June 2023).
20. Mills, R.; Marnerides, A.K.; Broadbent, M.; Race, N. Practical Intrusion Detection of Emerging Threats. *IEEE Trans. Netw. Serv. Manag.* **2021**, *19*, 582–600. [CrossRef]
21. Géron, A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2019.
22. Chong, B.Y.; Salam, I. Investigating deep learning approaches on the security analysis of cryptographic algorithms. *Cryptography* **2021**, *5*, 30. [CrossRef]
23. Safavian, S.R.; Landgrebe, D. A survey of decision tree classifier methodology. *IEEE Trans. Syst. Man Cybern.* **1991**, *21*, 660–674. [CrossRef]
24. Breiman, L. Random forests. *Mach. Learn.* **2001**, *45*, 5–32. [CrossRef]
25. Cortes, C.; Vapnik, V. Support-vector networks. *Mach. Learn.* **1995**, *20*, 273–297. [CrossRef]
26. Lewis, D.D. Naive (Bayes) at forty: The independence assumption in information retrieval. In Proceedings of the European Conference on Machine Learning, Chemnitz, Germany, 21–23 April 1998; Volume 1398, pp. 4–15. [CrossRef]
27. Benmessahel, I.; Xie, K.; Chellal, M. A new evolutionary neural networks based on intrusion detection systems using multiverse optimization. *Appl. Intell.* **2018**, *48*, 2315–2327. [CrossRef]
28. Liu, W.; Wang, Z.; Liu, X.; Zeng, N.; Liu, Y.; Alsaadi, F.E. A survey of deep neural network architectures and their applications. *Neurocomputing* **2017**, *234*, 11–26. [CrossRef]
29. Vinayakumar, R.; Alazab, M.; Soman, K.; Poornachandran, P.; Al-Nemrat, A.; Venkatraman, S. Deep learning approach for intelligent intrusion detection system. *IEEE Access* **2019**, *7*, 41525–41550. [CrossRef]

30. Ferrag, M.A.; Maglaras, L.; Moschoyiannis, S.; Janicke, H. Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study. *J. Inf. Secur. Appl.* **2020**, *50*, 102419. [CrossRef]
31. Bengio, Y. *Learning Deep Architectures for AI*; Now Publishers Inc: Norwell, MA, USA, 2009. [CrossRef]
32. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef]
33. Hinton, G.E.; Osindero, S.; Teh, Y.W. A fast learning algorithm for deep belief nets. *Neural Comput.* **2006**, *18*, 1527–1554. [CrossRef]
34. Kasongo, S.M.; Sun, Y. A Deep Learning Method with Filter Based Feature Engineering for Wireless Intrusion Detection System. *IEEE Access* **2019**, *7*, 38597–38607. [CrossRef]
35. Hinton, G.E.; Salakhutdinov, R.R. Reducing the dimensionality of data with neural networks. *Science* **2006**, *313*, 504–507. [CrossRef] [PubMed]
36. Kaja, N.; Shaout, A.; Ma, D. An intelligent intrusion detection system. *Appl. Intell.* **2019**, *49*, 3235–3247. [CrossRef]
37. Kanimozhi, V.; Jacob, T.P. Artificial intelligence based network intrusion detection with hyper-parameter optimization tuning on the realistic cyber dataset CSE-CIC-IDS2018 using cloud computing. In Proceedings of the 2019 International Conference on Communication and Signal Processing (ICCSP), Chennai, India, 4–6 April 2019; pp. 33–36. [CrossRef]
38. Verma, A.; Ranga, V. On evaluation of Network Intrusion Detection Systems: Statistical analysis of CIDDS-001 dataset using machine learning techniques. *Pertanika J. Sci. Technol.* **2018**, *26*, 1307–1332.
39. Kilincer, I.F.; Ertam, F.; Sengur, A. Machine learning methods for cyber security intrusion detection: Datasets and comparative study. *Comput. Netw.* **2021**, *188*, 107840. [CrossRef]
40. Li, X.; Chen, W.; Zhang, Q.; Wu, L. Building auto-encoder intrusion detection system based on random forest feature selection. *Comput. Secur.* **2020**, *95*, 101851. [CrossRef]
41. Aksu, D.; Üstebay, S.; Aydin, M.A.; Atmaca, T. Intrusion detection with comparative analysis of supervised learning techniques and fisher score feature selection algorithm. In Proceedings of the International Symposium on Computer and Information Sciences, Poznan, Poland, 20–21 September 2018; pp. 141–149. [CrossRef]
42. Kostas, K. Anomaly Detection in Networks Using Machine Learning. Master's Thesis, University of Essex, Colchester, UK, 2018.
43. GitHub Repository: Evaluation-of-Machine-Learning-Algorithm-in-Network-Based-Intrusion-Detection-System. Available online: https://github.com/tuanhong3498/Evaluation-of-Machine-Learning-Algorithm-in-Network-Based-Intrusion-Detection-System (accessed on 25 May 2023).
44. Chua, T.H.; Salam, I. Evaluation of Machine Learning Algorithms in Network-Based Intrusion Detection System. *arXiv* **2022**, arXiv:2203.05232. [CrossRef]
45. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
46. Mills, R.M. *Enhancing Anomaly Detection Techniques for Emerging Threats*; Lancaster University: Lancaster, UK, 2022.