Machine Learning Based Intrusion Detection System

Sindhuja Reddy Thummala
Department of Computer Science
University Of Central Florida
Orlando, Florida
sindhuja07@knights.ucf.edu

Priyanka Gopi
Department of Computer Science
University Of Central Florida
Orlando, Florida
priyankagopi@knights.ucf.edu

***Abstract*—The increase of cyber assaults has made cybersecurity a top responsibility for businesses, and intrusion detection systems (IDS) have become critical instruments for detecting these attempts. Machine learning (ML) is being utilized to create intrusion detection systems (IDS) capable of detecting zero-day assaults. However, the typical technique of evaluating an IDS's effectiveness by eliminating attack samples from the training dataset and testing the model is insufficient for evaluating long-term performance since it does not account for changes in the kind of assaults and network infrastructure over time. This research presents a new way for evaluating the long-term effectiveness of ML-based IDS by utilizing a testing dataset generated after the training dataset. The study compares six prominent ML models and discovers that Artificial Neural Network (ANN) are the most resistant to overfitting, while Decision Tree (DT) and Random Forest (RF) are the most susceptible. When the gap between the training and testing datasets is minor, all models perform well. The study concludes that the suggested technique can better analyze the long-term performance of ML-based IDS.**

***Index Terms*—Cybersecurity, Machine Learning, Intrusion Detection System, Overfitting, Cyber Threat Intelligence, Decision Tree, Random Forest.**

I. INTRODUCTION

Because of the epidemic, people have become more reliant on computers and the internet, but this has also resulted in an increase in cyberattacks, making cybersecurity a critical concern for businesses. There are several countermeasures available to safeguard computer systems from assaults, such as network access control, antivirus software, Virtual Private Network (VPN), and Intrusion Detection Systems (IDS). IDS is a program that analyzes traffic for signals of intrusion and can be network-based IDS (NIDS) or host-based IDS (HIDS), depending on the traffic being monitored. IDS can alternatively be characterized as signature-based IDS or anomaly-based IDS, depending on how intrusions are detected.

According to recent research, Machine Learning (ML) algorithms can increase IDS accuracy by learning the behavior of benign and malicious network traffic. However, IDSs frequently fail to identify zero-day attacks and unknown assaults that are not in the existing dataset. This work suggests training and testing ML-based intrusion detection systems (IDS) using various datasets to imitate real-world events. The study identifies and analyzes the performance of several ML techniques, including decision trees, random forests, naive Bayes, artificial neural networks, and deep neural networks.

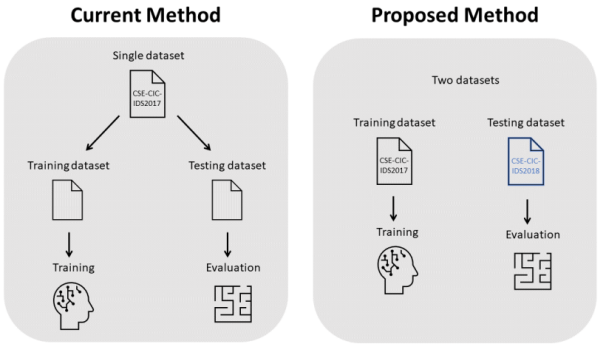

Fig. 1. Previous Studies and Proposed Experiment

II. PROBLEM STATEMENT

To assist in the identification of cyberattacks, intrusion detection systems (IDS) are used as alarm systems. When new types of cyberattacks appear, researchers are employing machine learning technology to create IDS that can recognize even zero-day threats. The efficiency of this system cannot be accurately assessed when the same dataset is used for both training and testing due to the constantly changing nature of data. Before being used to analyze live network traffic, the model needs to be trained on previous data. The difficulty arises from the fact that businesses and attackers are changing network topologies and architecture, as the network environment of the future is likely to differ from the one of the present. New zero-day attacks that are not included in the available dataset may also occur.

III. RELATED WORK

During the last ten years, a lot of research is being conducted regarding the application of machine learning (ML) to intrusion detection systems (IDS). In order to increase precision and productivity, scientists have built sophisticated algorithms. These include fusing supervised and unsupervised models, as well as investigating cutting-edge techniques such as deep learning. Current study evidence employing

specific investigations concentrating on statistical comparisons indicates that IDS using ML models may attain excellent accuracy percentages. The capacity to recognize unidentified dangers is additionally a challenge, and several datasets are utilized for assessment. Various datasets contain various feature sets, so when analyzing these datasets, algorithms are trained and tested again. In order to accurately represent the IDS's longevity, this research suggests multiple sets of training and testing datasets which employ a comparable feature set but were produced at different times.

IV. DATASET

In this section, some of the well-known datasets employed in the field of intrusion detection systems (IDS) are briefly described. Furthermore, we explore the machine learning techniques used in this paper.

	avg_ipt	bytes_in	bytes_out	dest_ip	dest_port	entropy	num_pkts_out	num_pkts_in	proto	src_ip	src_port	time_end	time_start	total_entropy	label	duration
0	0.000000	0	2896	786	9200.0	4.263922	2	0	6	786	32862.0	1593574200046275	1593574200046255	12348.318	benign	0.000020
1	0.285714	0	10136	786	9200.0	2.710831	7	3	6	786	32898.0	1593574199053693	1593574199051282	27476.582	benign	0.002411
2	0.000000	0	6074	786	9200.0	3.692507	5	0	6	786	32902.0	1593574200047203	1593574200047712	22428.287	benign	0.000083
3	0.000000	0	2896	786	9200.0	4.803335	2	0	6	786	32902.0	1593574201053831	1593574201053819	13910.459	benign	0.000012
4	0.000000	0	0	786	32898.0	0.000000	1	0	6	786	9200.0	1593574202069599	1593574202069599	0.000	benign	0.000000

Fig. 2. LUFlow 2020 Train Dataset Attributes

Because businesses are reluctant to share their network traffic data because of privacy and security concerns, acquiring reliable data for intrusion detection systems (IDS) can be difficult. Still, having accurate datasets is necessary in creating and assessing anomaly based IDS's efficient operation. As a consequence of this, many datasets were produced for use in research by different companies. We present the datasets utilized in the studies in this section.

The Lancaster University-created LUFlow dataset. It was newly introduced to the IDS datasets in 2021 after being revised in 2020. It stands out because it collects authentic information through honeypots located inside Lancaster University's public address system. The dataset has constraints, though, in that it does not specifically identify the sort of attack being used; rather, it just classifies traffic as either innocent or dangerous. The reason for this is due to the fact that it is challenging to ascertain each traffic's purpose given that the data is gathered from actual sources in the real world. On top of that, certain network traffic is labeled as "outlier" whenever it exhibits unusual behavior but isn't connected to a rogue node.

V. METHODOLOGY

The procedures used in the study are broken down into a total of five primary phases: dataset pre-processing, feature selection, hyperparameter optimization, cross-validation enabling accuracy validation, finally model performance assessment utilizing various measurements using the test data set. Figure below shows a full flowchart of the research. The LUFlow dataset [28] is the source of this dataset. This is crucial to remember that the LUFlow dataset arranges data corresponding of the date when it was taken, even if this could appear in contrast with the goal of this research. In the present study, the training dataset is made up of the data gathered during July 2020, whereas the testing dataset is made up of data gathered during January 2021.

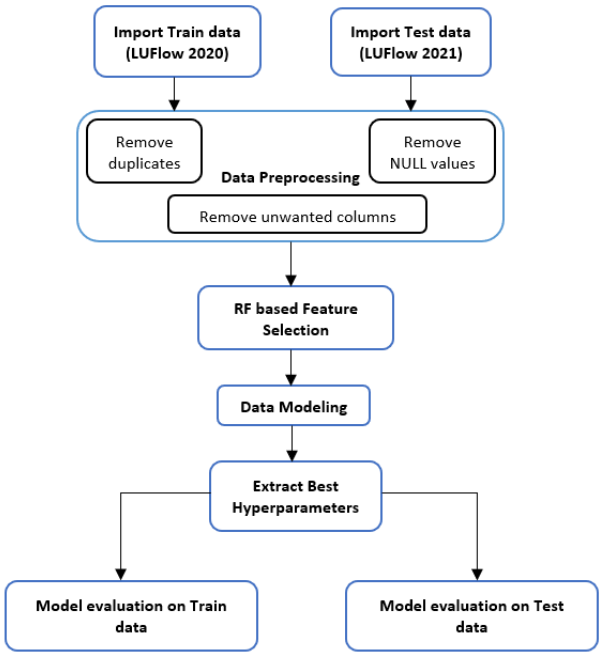

Fig. 3. Flow Chart of Methodology.

A. Data Processing

In this case, the data set undergoes pre-processing with first cleaning, which eliminates duplicates & drops entries with incomplete or infinite values. Then, the majority class is downsampled & many minority classes are combined to create a bigger group in order to tackle the exceptional mismatch issue. Although redefining needs to be done, the training and testing datasets usually contain an identical set of columns in the identical sequence and with identical classes in order to maintain continuity.

B. Feature Selection

When training a model using machine learning using datasets containing plenty of attributes, the writing explores the value of feature selection. In order to decrease the amount of attributes depending on accuracy of modeling, it suggests utilizing the algorithm known as random forest in feature selection. To exclude features like source IP address which could lead towards overfitting, human inspection is recommended as well. Our objective was to develop a smaller collection of characteristics that can shorten training time while enhancing model accuracy.

C. Data Modeling and Hyperparameter Optimization

Following feature selection for the present study, the algorithms are taught using the training dataset. Employing the GridSeachCV function offered by scikit-learn, that looks across a preset hyperparameter space and trains the model

applying an assortment of hyperparameters, the simulation's hyperparameters were tuned. For training the ultimate models, the most suitable hyperparameters with maximum accuracy are chosen. Hyperparameters, which include a DNN's variety of layers that are concealed, are controls throughout the training process. Just a small portion of the training dataset is utilized in order to quicken up the method because tweaking those hyperparameters demands training the algorithms repeatedly.

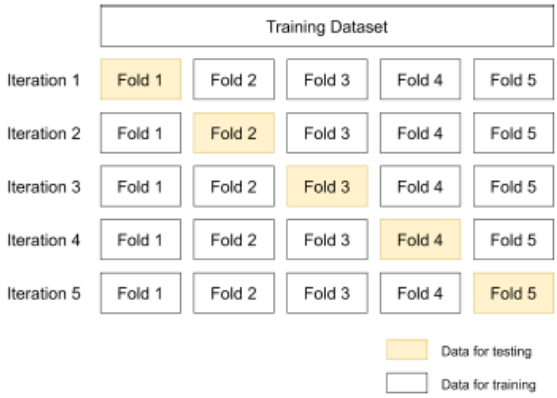

Fig. 4. Using 5-fold Cross Validation Method.

D. ALGORITHMS

Machine learning (ML) has attracted more and more interest lately, as scientists are investigating all the ways they may use it. Machine learning (ML) algorithms are skilled at classifying or forecasting data. IDS uses machine learning to categorize traffic as either innocuous or malicious. We give an in-depth examination of some of the most popular ML models in the IDS area in this section.

1) Decision Tree: An algorithm that classifies information using a structure resembling a tree is called a decision tree (DT) classification. Nodes and leaves make up the tree, where nodes stand in for requirements and leaves for categories. A decision tree is constructed using the data that was used for training throughout the training phase, with every iteration choosing an attribute to divide the information on a leaf node unless the leaf component's contents are homogeneous.

The choice of a feature is determined by its ability to it divides the data, as determined by measurements like Entropy and Gini. The choice of characteristics is facilitated by the fact that characteristics located near the root node partition the data more effectively and exhibit greater associations to the expected result. Each of the samples through the dataset being tested is uploaded through the tree during the phase of testing and moves from the root node towards the leaf node, which corresponds with the sample's class. a combination of the model's excellent precision and simplicity of training, the decision tree has become one of the more often used algorithms in the IDS field.

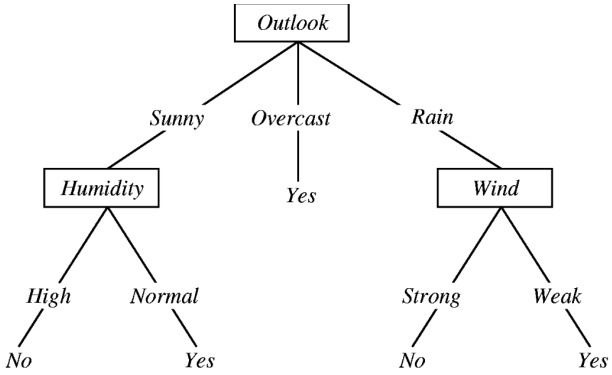

Fig. 5. Example of Decision Tree.

```
# build the decision tree model

DTmodel = tree.DecisionTreeClassifier(criterion='entropy', ccp_alpha=ideal_ccp_alpha)
DTmodel.fit(train_X, luflow_train_y)
y_pred_dt = DTmodel.predict(test_X)
accuracy = metrics.accuracy_score(luflow_test_y, prediction)
print(f"Decision Tree model Accuracy: {accuracy:.5f}")

Decision Tree model Accuracy: 0.99799
```

Fig. 6. Implementation of Decision Tree.

We used the 5-fold cross-validation used to evaluate model performance. We considered 'entropy' as the criterion for splitting the data. The ideal alpha value that we selected were based on highest mean accuracy. The model was trained/tested on LUFlow 2020. We got an Accuracy: 0.99799 for this dataset.

2) Random Forest: The algorithm known as random forest (RF) consists of several decision trees. A bootstrapped dataset is employed for constructing every decision tree, and it is produced by selecting at random rows from the first training dataset. They differ distinctively from one another since every decision tree is trained utilizing a separate bootstrapped dataset.

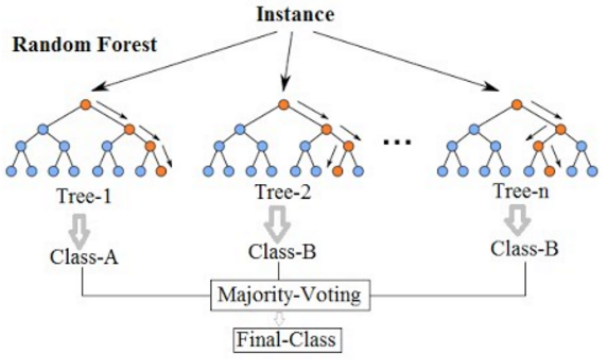

Fig. 7. Random Forest Example.

As illustrated in Figure 3, after gathering the votes for every tree, the classification is made according to the class that gained the highest number of votes. Given that it uses numerous decision trees while keeping the decision tree

model's effectiveness, random forest has the advantage of being less vulnerable to overfitting as well as less susceptible to dataset noise. As a result of its strong performance and capacity for managing complicated datasets, RF is a frequently employed model in IDS.

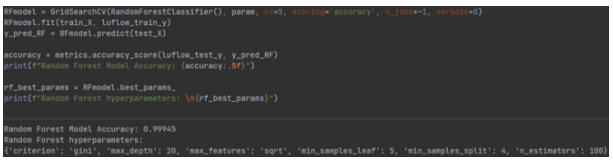

Fig. 8. Implementation of Random Forest.

We performed the model traing and testing using the Hyperparameters, which were tuned using GridSearchCV. We used the cross-validation with 5-fold validation in this model. The best hyperparameters were selected here based on highest accuracy obtained. For this model we got an Accuracy: 0.99945 for the same dataset used for Decsiion Tree. The 'entropy' criterion for splitting is used in the decision tree classification model, and the cost complexity pruning path approach is used to get alpha values. To assess model performance, cross-validation with 5-fold validation is used, and the ideal alpha value is chosen based on the highest mean accuracy. The optimal alpha value is used to build the model, which is then trained and tested on LUFlow 2020 data. The model's accuracy, which was calculated to be 0.99799, demonstrates its strong predictive capability. Next in our experiment, GridSearchCV was utilized to finetune the Random Forest model's hyperparameters. Criteria, max_depth, min_samples_leaf, n estimators, max_features, and min_samples_split were the hyperparameters that were tuned. For model evaluation, we used 5-fold cross-validation. The criteria used to choose the optimum hyperparameters were: "criterion": "gini", "max_depth", "max_features", "sqrt", "min_samples_leaf", "min_samples_split", and "n_estimators". On the test data, the Random Forest model's accuracy was found to be 0.99945.

3) Naïve Bayes: The naive Bayes classification algorithm is a simple Bayes theorem-based probabilistic predictor. Relying on the probability of another event, it aids in calculating the likelihood that a specific event will take place. Utilizing this algorithm, they may determine the likelihood that a data point belongs to a particular class provided we know the probability associated with that class, the likelihood that the data point's features will appear given that class, plus the likelihood of the features will occur in general. The algorithm calculates these probabilities using the training set during the training phase. It is quite effective because calculating probabilities is an easy operation. This approach can achieve accuracy comparable to more complicated models even though it makes the assumption that each of the variables are distinct from one another. When performd the model accuracy we got the Lower accuracy L: 0.73672, than DT and RF models. For this model we used the optimum hyperparameters: 'var_smoothing' =

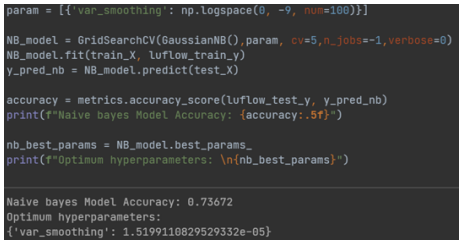

Fig. 9. Implementation of Naïve Bayes

1.5199110829529332e05 for better results. Lower precision, recall, and F1-score for 'benign' class, and slightly higher for 'malicious' class compared to Decision Tree and Random Forest models. The total Computation time: 2 minutes and 24 seconds. With an accuracy of 73.67%, the Naive Bayes model performed less accurately than the Decision Tree and Random Forest models. GridSearchCV was used to train the model using hyperparameter tuning, and an ideal value was found for the 'var_smoothing' hyperparameter. In comparison to the Decision Tree and Random Forest models, the model had poorer precision, recall, and F1-score for the "benign" class, but significantly higher values for the "malicious" class.

4) Artificial Neural Network: Contrary to the previous approaches mentioned, the artificial neural network (ANN) is a more complicated approach. Intrusion detection systems (IDS) frequently employ the multi-layer perceptron (MLP), a kind of ANN. The MLP consists of many perceptron layers, with every one being made up of many different kinds of neurons. In contrast to the output layer, which reflects the data's class, the input layer reflects the characteristics underlying the input data. The process of mapping between a single perceptron onto another is a linear function as well as non-linear activation process, and neurons among various layers are linked by weighted vertices. Both the biases and weights are tuned to fit the training data throughout training. A big dataset is necessary to avoid overfitting, yet training an artificial neural network (ANN) is expensive in terms of computation. The primary benefit of neural networks is the capacity for processing raw data and then abstracting it through layers of neurons to enable them to develop more MLPClassifier model's hyperparameters tuned using GridSearchCV. For quicker processing, reduce the hidden_layer_sizes to (30,), (20,), and (10,). Test accuracy + best hyperparameters = 95.57%. Best hyperparameters : solver = "adam", activation = "relu," alpha = "0.001", hidden_layer_sizes" and activation = "relu".

5) Deep Neural Network: After Hinton, Osindero, and Teh trained a DNN with 3 concealed layers effectively in 2006, the DNN gained momentum. It has also become more commonplace over the past few years to use DNN for intrusion detection systems (IDS). A Multi-Layer Perceptron (MLP) with multiple layers that are concealed is a sort of deep learning model known as DNN. DNN works more effectively

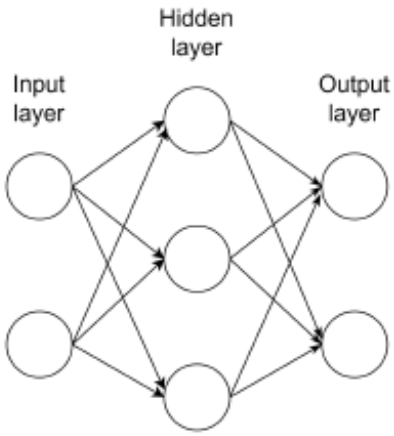

Fig. 10. Hidden layer in MLP Model.

```
param = [
    {'hidden_layer_sizes': [(30,), (20,), (10,)],
    'activation': ['relu'],
    'solver': ['adam'],
    'alpha': [0.0001, 0.001]}
]

ann = MLPClassifier(max_iter=500)
ANNmodel = GridSearchCV(ann,param, cv=5,n_jobs=-1,verbose=0)

ANNmodel.fit(train_X, luflow_train_y)
y_pred_ann = ANNmodel.predict(test_X)

accuracy = metrics.accuracy_score(luflow_test_y, y_pred_ann)
print(f"Accuracy: {accuracy:.5f}")

ann_best_params = ANNmodel.best_params_
print(f"ANN Best hyperparameters: \n{ann_best_params}")
```

```
Accuracy: 0.95578
ANN Best hyperparameters:
{'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (30,), 'solver': 'adam'}
CPU times: user 15.8 s, sys: 384 ms, total: 16.2 s
Wall time: 4min 39s
```

Fig. 11. Implementation of Artificial Neural Network

with no pre-training than MLP with a single hidden layer. The deep architecture of DNN also enables it to learn more intricate patterns. But because of their greater complication DNNs require more time to train & need a greater quantity of data for improving their parameters.

```
param = [{'hidden_layer_sizes': [(10, 10), (12, 12), (8, 8), (10, 10, 10)],
         'activation': ['tanh', 'relu', 'logistic'],
         'solver': ['adam', 'sgd'],
         'alpha': [0.001, 0.0001, 0.00001]}]

DNN_model = GridSearchCV(MLPClassifier(max_iter=500), param, cv=5, n_jobs=-1,verbose=0)
DNN_model.fit(train_X, luflow_train_y)
y_pred_dnn = DNN_model.predict(test_X)

accuracy = metrics.accuracy_score(luflow_test_y, y_pred_dnn)
print(f"DNN Model Accuracy: {accuracy:.5f}")

dnn_best_params = DNN_model.best_params_
print(f"DNN Best hyperparameters: \n{dnn_best_params}")
```

```
DNN Model Accuracy: 0.96853
DNN Best hyperparameters:
{'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (10, 10, 10), 'solver': 'adam'}
CPU times: user 1min 7s, sys: 3.99 s, total: 1min 11s
Wall time: 1h 5min 35s
```

Fig. 12. Implementation of Deep Neural Network

Neural Networks learn complex patterns even when the features of the training dataset are not well optimized. For the LUFlow dataset, we tuned the hyperparameters of an MLPClassifier (ANN) model using GridSearchCV. For quicker processing, we set activation to ”relu,” solver to ”adam,” and alpha to [0.0001, 0.001] while limiting the hidden_layer_sizes to (30, 20, and 10). Alpha = 0.001, hidden_layer_sizes = 30, and solver = ”adam” were the best hyperparameters identified. On the test data, the model had an accuracy of 95.57%. Tuning the hyperparameters took 4 minutes and 39 seconds in total, which is For the LUFlow dataset, we tuned the hyperparameters of an MLPClassifier (artificial neural network) model using GridSearchCV. The hidden_layer_sizes, activation, solver, and alpha hyperparameters have all been tweaked. Alpha = 0.001, hidden_layer_sizes = 30, and solver = ”adam” were the best hyperparameters identified. On the test data, the model had an accuracy of 95.57%. Tuning the hyperparameters took 4 minutes and 39 seconds in total, which is quick enough. For DNN, we optimized hyperparameters of an MLPClassifier using GridSearchCV. Best hyperparameters were found as activation = ’relu’, alpha = 0.001, hidden_layer_sizes = (10, 10, 10), solver = ’adam’, with an accuracy of 0.96853 on test data. Total time taken for hyperparameter tuning was about an hour.

E. Feature Selection

Since this seeks to choose a smaller number of features to categorize network traffic, feature selection becomes a crucial part of enhancing an IDS’s effectiveness. Since it may minimize the amount of attributes & arrange each based to the relevance result, the random forest algorithm comes in handy during feature selection. Researchers demonstrated that the suggested approach has equivalent accuracy while drastically lowering detection time when weighed against KitNET.

F. Model Evaluation

Employing cross-validation upon the training dataset, the algorithms are checked overall correctness prior they are evaluated against a different dataset in this phase. The data set for training is broken down in k-folds, which results in the i-th fold being employed to evaluate and the remaining folds being utilized for training in each iteration. The precision of the model is measured as the mean accuracy over the iterations of k. When the models acquire an accuracy that is equal to that of other works of literature, the experiment moves on onto the next stage. If not, the trial and error technique is enhanced.

The ultimate & most important stage in this study is to train the completed model while employing the testing dataset to evaluate it. The algorithms have been trained employing 70% of the training dataset, while the other thirty percent is used to assess the models’ accuracy. The accuracy and efficacy of various models are compared using indicators of performance like accuracy, F1-score,precision and recall. TP, FN,TN and FP, which indicate the amount samples that have been properly and erroneously deemed positive or negative, are used to calculate these measures. Additionally, confusion matrices are utilized for displaying the categorization outcomes. Every model’s temporal complexity is assessed as well, having em-

phasis being placed on how long training as well as forecasting take.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (3)$$

$$Precision = \frac{TP}{TP+FP} \quad (4)$$

$$Recall = \frac{TP}{TP+FN} \quad (5)$$

$$F1Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (6)$$

Fig. 13. Snippet of Formulas used in Model Evaluation.

VI. Experimentation

We implemented the code with Python. The code used makes utilize Python 3.8.8 and its modules such as scikit-learn, pandas etc. We used the Google Colab for the implementation of the project.

The details of the study utilizing the LUFlow dataset are highlighted in this part of the paper. The dataset used for testing is made up of data gathered in January 2021, while the training dataset is made up of data gathered in July 2020. The one used for training is termed LUFlow 2020, and the testing dataset is called LUFlow 2021. Each data set is alluded to after the date that it was gathered.

A. Data Pre-Processing

The LUFlow dataset's preparation handles will be clarified in the following paragraphs. Just ten percent of the pooled LUFlow 2020 and LUFlow 2021 datasets were utilized in this study because to the huge magnitude of the datasets, which were pooled by each day. The sets of data turned out

```
luflow2020 = luflow2020.dropna(how='any')
luflow2020 = luflow2020.drop_duplicates()

luflow2021 = luflow2021.dropna(how='any')
luflow2021 = luflow2021.drop_duplicates()
```

Fig. 14. Using 5-fold Cross Validation Method.

```
train_X = luflow_train_X[['dest_port','bytes_out','src_port','total_entropy','num_pkts_in', 'duration']]
test_X = luflow_test_X[['dest_port','bytes_out','src_port','total_entropy','num_pkts_in', 'duration']]


scale = StandardScaler()   # scaling
X_train = scale.fit_transform(train_X)
X_test = scale.transform(test_X)
```

Fig. 15. Using 5-fold Cross Validation Method.

```
attack = luflow2020[luflow2020['label']=='malicious']
benign = luflow2020[luflow2020['label']=='benign'].sample(n=len(attack)).reset_index(drop=True)

luflow2020_exclude_outlier = pd.concat([attack, benign])
```

Fig. 16. Using 5-fold Cross Validation Method.

to have duplicated and values that were missing, and these were eliminated. The somewhat unbalanced class distribution got fixed through picking at random innocuous examples to create an one-to-one proportion to harmful data. Exceptions have been eliminated in the dataset since the ML models view these to be noise. Table 9 displays the class distribution for the filtered & resampled datasets.

Table 8: Class distribution of LUFlow dataset before cleaning.

Classes	LUFlow 2020		LUFlow 2021	
	No. of rows (10%)	No. of rows (%)	No. of rows (10%)	No. of rows (%)
benign	1396168	55.71%	1638952	56.36%
malicious	905395	36.12%	591372	35.52%
outlier	204787	8.17%	469345	8.12%

Table 9: Class distribution of LUFlow dataset after cleaning.

Classes	LUFlow 2020		LUFlow 2021	
	No. of rows	No. of rows (%)	No. of rows	No. of rows (%)
benign	879740	50%	569003	50%
malicious	879740	50%	569003	50%

Fig. 17. Tables of Class Distributions.

B. Feature Selection and Data Modeling

Even with just sixteen features in the LUFlow dataset, feature selection is still required in order to minimize distortion while improving the accuracy of the model. For the purpose of preventing overfitting, the'src_ip' & 'dest_ip' variables were deliberately deleted. The 'time_start', 'time_end', & 'duration' columns have been eliminated due to their hadn't been vital to classification. During feature selection, random forest is the method employed. The other characteristics are sorted based

Feature	Description
src_ip	The source IP address associated with the flow. This feature is anonymised to the corresponding Autonomous System.
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 in the same manner as before.
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.

Fig. 18. Table Showing Features and their Descriptions

on their relevance result, which is calculated using random forest, following each of these characteristics have been eliminated. The characteristics' ordering is represented visually. The best-scoring characteristic in Figure 15 is "dest_port," which has an evaluation that is much greater than "bytes_out." Furthermore, the relevance ratings of the final 5 components were significantly less compared to the scores of the features that were rated higher. The minimal significance value suggests that those features don't offer a lot of classification material.

Brute force may be employed to increase the efficiency of the features in the feature selection process. The precision the

each model can be seen with the total amount of features once every one of them have undergone brute force testing. While ANN and DNN need a minimum of five or seven traits for excellent precision, Naive Bayes is most accurate if the first two through 6 characteristics are employed. Even using a single feature, DT and RF attain exceptional precision. The last characteristic set has been decided upon using the best 6 features.

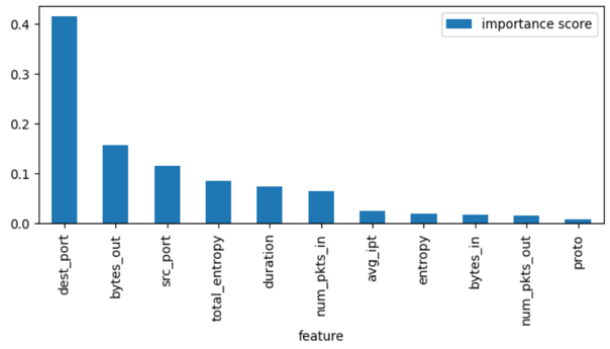

Fig. 19. Random Forest algorithm based Feature Importance

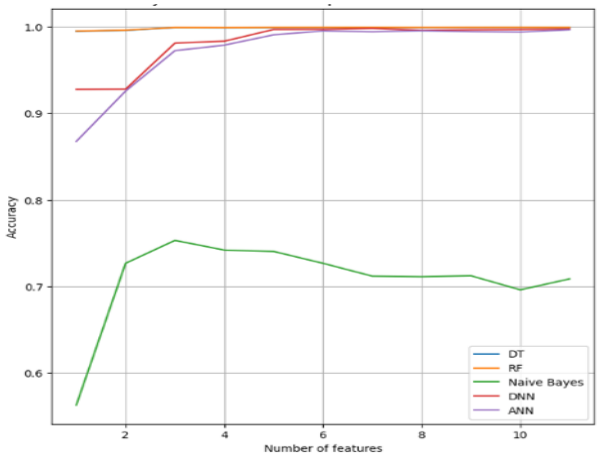

Fig. 20. Brute Force Performance Evaluation.

C. Evaluation

For the purpose of identifying whether any models were overfitting, precision for each model had been verified five times. Apart for Naive Bayes, that showed a precision of only 73.23%, other models showed exceptional accuracy. Every model's accuracy had a low standard deviation, demonstrating how the correctness for the simulations was constant. The most accurate models were Decision Tree and Random Forest (RF), having a median accuracy of 99.895%. NB, that also happened to be not the most trustworthy model, had the least favorable performance. For the LUFlow dataset, ANN and DNN achieved accuracy levels are 99.74% and 99.3%, which was a substantial improvement. It remained novel, therefore no prior work utilizing it had been done for contrast, consequently the accuracy of various models wasn't matched with other publications.

Decision Tree

	precision	recall	f1-score	support
benign	0.9996	0.9985	0.9990	227539
malicious	0.9985	0.9996	0.9990	227365
accuracy			0.9990	454904
macro avg	0.9990	0.9990	0.9990	454904
weighted avg	0.9990	0.9990	0.9990	454904

Random Forest

	precision	recall	f1-score	support
benign	0.9996	0.9981	0.9989	227539
malicious	0.9981	0.9996	0.9989	227365
accuracy			0.9989	454904
macro avg	0.9989	0.9989	0.9989	454904
weighted avg	0.9989	0.9989	0.9989	454904

Naive Bayes

	precision	recall	f1-score	support
benign	0.9511	0.4899	0.6467	227539
malicious	0.6563	0.9748	0.7845	227365
accuracy			0.7323	454904
macro avg	0.8037	0.7324	0.7156	454904
weighted avg	0.8037	0.7323	0.7156	454904

Fig. 21. Classification Report for ML Models

Deep Neural Network

	precision	recall	f1-score	support
benign	0.9951	0.9997	0.9974	227539
malicious	0.9997	0.9951	0.9974	227365
accuracy			0.9974	454904
macro avg	0.9974	0.9974	0.9974	454904
weighted avg	0.9974	0.9974	0.9974	454904

Artificial Neural Network

	precision	recall	f1-score	support
benign	0.9872	0.9990	0.9931	227539
malicious	0.9990	0.9871	0.9930	227365
accuracy			0.9930	454904
macro avg	0.9931	0.9930	0.9930	454904
weighted avg	0.9931	0.9930	0.9930	454904

Fig. 22. Classification Report for Deep Learning Models

Utilizing 20% apiece of the LUFlow 2020 & LUFlow 2021 datasets, researchers evaluated each of the machine learning (ML) models in this section using the testing dataset, LUFlow 2021. 70% from the LUFlow 2020 data was used to train the final machine learning models, and classification report were used to assess how well they performed using the training dataset. Despite the exception of NB, that overfits the good samples and has a poor recall score with malicious samples, the majority of models have acceptable accuracy. There was also no prejudice against a particular class, as other ML models achieved comparable precision and recall ratings.

VII. Results

We were able to assess the effectiveness of different models using machine learning to identify cyberattacks thanks to an experiment done on the LUFlow dataset. Results point out that the models developed using the LUFlow dataset are not overfitted. The DT approach is better in a system having less

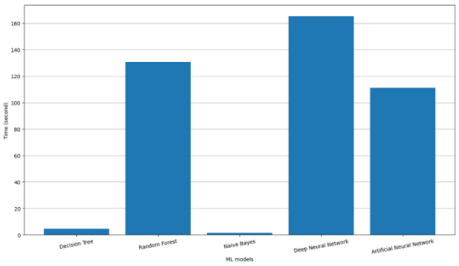

Fig. 23. Bar chart for Model Accuracy Comparision

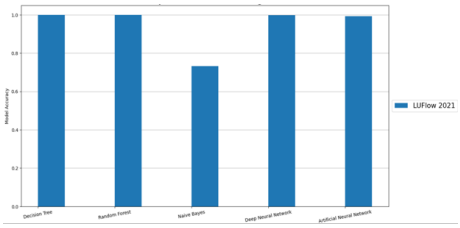

Fig. 24. Bar chart for Model accuracy on Testing Data

frequent structure changes & less attacks, while the ANN model works better for a system having constantly updated structure and large costs of cyberattacks. No matter the ML model chosen, the IDS must be constantly refreshed with more recent data.

VIII. Conclusion

This research paper outlines an innovative method to assess how effective it is over time of machine learning-based intrusion detection systems (IDS). The suggested strategy entails training an IDS using a single data set then assessing it with another set of data from the training dataset. Five machine learning (ML) models, including decision tree, random forest, naive bayes, artificial neural network, in addition to deep neural network, have been employed in the experiments utilizingLUFlow dataset. The findings point out that ANN is the model with the highest longevity, but DT is better suited for petites businesses. The suggested strategy may enhance feature selection & better detect overfitting. The next study aims to assess alternative comprehensive models suggested in the prior research, including unsupervised ML models.

The difficulty of having limited GPU and computing power were the main challenges in the project. The development of more effective feature selection techniques is a future area for advancement. For additional progress, models need to be further hyper-tuned and also we can implement SVM model for more baseline comparisions.We can also test the models on more datasets.

Acknowledgment

We would like to offer our sincere appreciation to Dr. Cliff C. Zou from the University of Central Florida for providing us with the chance to work on this real-time project as part of the Malware and Software Vulnerability Analysis (CAP6135) course. His mentoring and advice were crucial to our success. We also want to thank Kaggle for supplying us with the LUFlow dataset, which was necessary for carrying out our research. We also like to acknowledge our teammate's amazing efforts in this project. We are thankful for their equally important contributions, and we feel privileged to have collaborated as a team. Below is the Github link for the code of this project. https://github.com/MalwareFinalProject

References

[1] N. Kaja, A. Shaout, D. Ma, An intelligent intrusion detection system, Applied Intelligence 49 (9) (2019) 3235–3247. doi:10.1007/s10489-019-01436-1

[2] V. Kanimozhi, T. P. Jacob, Artificial intelligence based network intrusion detection with hyperparameter optimization tuning on the realistic cyber dataset CSE-CIC-IDS2018 using cloud computing, in: 2019 international conference on communication and signal processing (ICCSP), IEEE, 2019, pp. 33–36. doi:10.1109/ICCSP.2019.8698029.

[3] M. A. Ferrag, L. Maglaras, S. Moschoyiannis, H. Janicke, Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study, Journal of Information Security and Applications 50 (2020) 102419. doi:10.1016/j.jisa.2019.102419.

[4] M.A. Ferrag, L. Maglaras, S. Moschoyiannis, H. Janicke, Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study, Journal of Information Security and Applications 50 (2020) 102419. doi:10.1016/j.jisa.2019.102419.

[5] G. E. Hinton, R. R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (5786) (2006) 504–507. doi:10.1126/science.1127647

[6] A. Khraisat, I. Gondal, P. Vamplew, An anomaly intrusion detection system using C5 decision tree classifier, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2018, pp. 149–155. doi:10.1007/978-3-030-04503-6_14.

[7] LUFlow Network Intrusion Detection dataset, https://www.kaggle.com/datasets/mryanm/luflow-network-intrusion-detection-data-set

[8] R. Vinayakumar, M. Alazab, K. Soman, P. Poornachandran, A. Al-Nemrat, S. Venkatraman, Deep learning approach for intelligent intrusion detection system, IEEE Access 7 (2019) 41525–41550. doi:10.1109/ACCESS.2019.2895334.

[9] W. Liu, Z.Wang, X. Liu, N. Zeng, Y. Liu, F. E. Alsaadi, A survey of deep neural network architectures and their applications, Neurocomputing 234 (2017) 11–26. doi:10.1016/j.neucom.2016.12.038.

[10] K. Kostas, Anomaly Detection in Networks Using Machine Learning, Master's thesis, University of Essex, Colchester, UK (2018).

[11] Kaspersky, What is a zero-day attack? - definition and explanation, https://www.kaspersky.com/ resource-center/definitions/zero-day-exploit

[12] R. Vinayakumar, M. Alazab, K. Soman, P. Poornachandran, A. Al-Nemrat, S. Venkatraman, Deep learning approach for intelligent intrusion detection system, IEEE Access 7 (2019) 41525–41550. doi:10.1109/ACCESS.2019.2895334.

[13] Github repository: Evaluation-of-machine-learning-algorithm-in-network-based-intrusion-detectionsystem, https://github.com/tuanhong3498/Evaluation-of-Machine-Learning-Algorithm-in-Network- Based-Intrusion-Detection-System.

[14] I. Sharafaldin, A. H. Lashkari, A. A. Ghorbani, 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), Vol. 1, 2018, pp. 108–116. doi:10.5220/0006639801080116.

[15] MonsterCloud, Top cyber security experts report: 4,000 cyber attacks a day since covid-19 pandemic, https://www.prnewswire.com/news-releases/top-cyber-security-experts-report-4-000-cyberattacks- a-day-since-covid-19-pandemic-301110157.html.