The Use of Machine Learning in The Detection of Anomaly Network Intrusion

Akshat Behera
A20516439
abehera2@hawk.iit.edu

Nagarjuna Bolla
A20524548
nbolla@hawk.iit.edu

Final Project Report

1 Introduction

1.1 Abstract

The goal of our project is to focus on developing an effective machine learning-based approach for detection of anomaly network intrusion detection using the UNSW-NB15 dataset. As we know that the concerns about the digital safety and digital security of personal and professional data have never been higher than as they are now, in the modern age of digitization.

In order to identify and stop malicious malwares/viruses/attacks and the attackers from taking advantage of network vulnerabilities, network intrusion detection has become a crucial part of cybersecurity. However, In the present times we have seen that traditional rule-based approaches have proven ineffective in identifying sophisticated and novel attacks as attackers constantly refine their techniques to bypass detection.

Thus, this project explores the potential of neural networks such as NN using MLP Classifier, CNNs, and RCNNs, in detecting potential threats. The project also investigates the benefits of feature selection methods in improving classifier performance and aims to compare the effectiveness of NNs with traditional classifiers. By enhancing classifier accuracy, this project aims to develop a potent network intrusion detection system that can identify even the most innovative and complex attacks, bringing untold benefits to organizations worldwide.

Overall, our project aims to advance the field of Network Intrusion Detection by exploring the use of Neural Networks and Feature Selection methods to improve classifier accuracy. This could lead to more effective and efficient Network Intrusion Detection systems capable of detecting even the most advanced and novel attacks.

1.2 Why UNSW NB-15?

The UNSW-NB15 [2] dataset has gained significant attention and popularity among researchers for network intrusion detection. This dataset is designed to simulate real-world network traffic, consisting of both normal and attack traffic, making it highly diverse and suitable for evaluating network intrusion detection methods. One of the strengths of this dataset is that it provides detailed features that can be used for feature engineering and building machine learning models. With the availability of such a dataset, we can experiment with various machine learning algorithms and evaluate their effectiveness in detecting network intrusions. By utilizing this dataset, we can develop and test machine learning-based approaches that can accurately and efficiently detect network intrusions, potentially leading to more secure network systems and infrastructure.

1.2.1 UNSW NB-15 Dataset Description

A network traffic dataset called UNSW-NB15 offers labeled traffic data that was produced by a network simulator. There are 49 features in all, including binary, nominal, integer, float, and timestamp types. The dataset contains 45 of the 49 features that were extracted from each network connection, as well as both legitimate and illicit traffic.

The dataset has been uploaded and is available our Project Github Repository.

The data is labeled as 0 and 1, where 0 indicates Normal (No Attack) and 1 indicates Attack (Any of the 9 Categories), with the "attack_cat" attribute specifying the type of attacks of the 9 categories. The nine categories of attacks included in this dataset are Fuzzers, Analysis, Backdoors, DoS Exploits, Generic, Reconnaissance, Shellcode, and Worms. A training set and a testing set were created from the UNSW-NB15

dataset, with a total of 175,341 records taken from the training set and split into 70-30 for training/test, and a total of 82,332 records taken from the testing set and split into 70-30 for training/test.

1.3 Related/Previous Work

Organizations all over the world are very concerned about cybersecurity, and one of its essential elements is network intrusion detection, which seeks to identify and stop attackers from taking advantage of network vulnerabilities. In order to achieve this, Machine Learning (ML) methods have been put forth, and numerous researchers are investigating their efficacy for Network Intrusion Detection.

Researchers have proposed various Machine Learning (ML) approaches to aid in Network Intrusion Detection, including supervised, unsupervised, and hybrid methods. Several studies have explored the effectiveness of ML approaches in Network Intrusion Detection. For example, Moustafa et al. (2015) [4] suggested using the UNSW-NB15 dataset, which includes both legitimate and malicious traffic, to assess Network Intrusion Detection Systems. They also recommended a feature selection technique that picks the most pertinent features using Principal Component Analysis (PCA). On the dataset, they tested a number of machine learning algorithms, such as Support Vector Machine, Naive Bayes, Random Forest, Decision Trees, and K-Nearest Neighbor. They discovered that by using PCA to choose features, the classifiers' accuracy increased, with Naive Bayes achieving the highest accuracy of 98.33%. They did point out that no single classifier was efficient for all types of attacks and that the performance of the classifiers varied depending on the type of attack.

Similarly, Alazab et al. (2016) [1] conducted a study of anomaly-based Network Intrusion Detection methods, highlighting the difficulties in this area and recommending a framework that makes use of numerous ML algorithms. They divided the methods into three groups: hybrid, anomaly-based, and signature-based. They discovered that while signature-based techniques had lower false-positive rates than anomaly-based techniques, the latter had higher detection rates. They suggested a hybrid strategy that combines both detection methods in order to increase detection precision while reducing false positives.

In a more recent survey, Islam and Ahmed (2019) [3] provided an extensive overview of machine learning (ML) methods for identifying network intrusions and divided them into three groups: supervised, unsupervised, and hybrid. They examined a number of studies that employed these techniques and discussed some of their drawbacks, such as the requirement for labeled data in supervised learning and the challenge in defining normal behavior in unsupervised learning. They also offered suggestions for future research directions, including enhancing the interpretability of ML models and creating more durable models that can withstand changing attacks.

Meanwhile, On the UNSW-NB15 dataset, Mukherjee et al. (2020) [6] evaluated the effectiveness of various Deep Learning (DL) approaches, including Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM), and Autoencoder. They discovered that LSTM, Autoencoder, and CNNs were the most accurate and had the highest F1-Score. The results showed that LSTM achieved the highest accuracy of 99.47% and outperformed other models. The effectiveness of DL (i.e., Deep Learning) approaches was also evaluated in comparison to that of well-known machine learning algorithms like Decision Tree, Random Forest, Naive Bayes, K-Nearest Neighbor, and Support Vector Machine, and it was discovered that DL approaches outperformed them in terms of accuracy and F1-Score. They also discussed the advantages and limitations of deep learning approaches for network intrusion detection.

Moustafa and Slay (2016) [5] introduced the UNSW-NB15 dataset and used it to conduct a statistical analysis of Network Anomaly Detection Systems. See [5]. They compared the performance of various machine learning algorithms, such as Decision Trees, Random Forest, Naive Bayes, K-Nearest Neighbor, and Support Vector Machine, on the dataset to that of the benchmark KDD99 dataset. They found that machine learning techniques had higher detection and accuracy rates on the UNSW-NB15 dataset. The performance of the classifiers varied depending on the type of attack, they continued, and no single classifier was effective for all types of attacks. To improve detection accuracy all around, they suggested using an ensemble of classifiers.

In summary, these studies provide a comprehensive overview of ML approaches for Network Intrusion Detection, evaluate the performance of various machine learning and deep learning algorithms, propose feature selection methods to improve classifier accuracy, and suggest directions for future research. Our proposed project aims to build on by exploring the potential of Neural Networks and Feature Selection.

Our project aims to develop a Machine Learning-based approach to Network Intrusion Detection, utilizing the UNSW-NB15 dataset. It seeks to build on by exploring the potential of Neural Networks (NNs), such as Neural Network (NN) using MLP Classifier, Convolutional Neural Networks (CNNs) and Recurrent Convolutional Neural Networks (RCNNs), for Network Intrusion Detection. Additionally, the project will investigate the

benefits of feature selection methods in improving the performance of the classifiers. Finally, the project aims to compare the effectiveness of NNs with traditional classifiers, including XGBoost, SVM, Random Forest, Decision Tree, and Logistic Regression.

2 Problem Description

2.1 Methodology

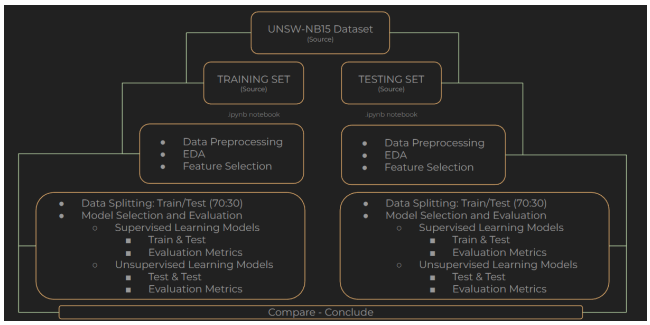


Figure 1: Outline of the Project Methodology

1. **Preprocessing of the Dataset:** Here, We begin with Extraction and Cleaning of the Data. The UNSW-NB15 dataset (both in Training and Testing Datasets) contains missing values and categorical variables, which needed to be addressed before feeding the data into the Machine Learning models. Here We, preprocessed the data by dropping unwanted columns, encoding categorical variables by using LabelEncoder(), and standardizing numerical variables by using StandardScaler(). This is done in both the notebooks of the Training Dataset and Testing Dataset.

   (a) **Data Cleaning:**

   - Dropping Unwanted Columns such as: 'id', 'proto', 'service', 'state'.
   - As they are not useful for the analysis and classification task.
   - Dropping such columns reduces the dimensionality of the dataset and improve the efficiency and effectiveness of the analysis.

   (b) **Data Encoding:**

   - LabelEncoder fits the encoder on unique values of a column and transforms them into numerical representation.
   - Encoding categorical variables allows the data to be used in machine learning models that require numeric inputs.
   - The resulting encoded values are integers ranging from 0 to n-1 in our case 0 to 9 range (1 normal no attack, 9 are attacks), where n (n=10) is the number of unique categories in the column.

   (c) **Data Standardization:**

   - Standardizes the Numerical Columns using StandardScalar().
   - This transformation is useful because many machine learning models assume that the data is standardized, and it can improve the performance of these models.

2. **Exploratory Data Analysis (EDA):** EDA is essential to understand the distribution of the features, correlation among the features, and to identify any outliers in the data. In accordance to our plan we have performed EDA using visualizations such as histograms, heatmaps, and boxplots. This is done in both the notebooks of the Training Dataset and Testing Dataset. These visualizations obtained gives us a more clearer picture on understanding the dataset and it's distribution across various factors.

   (a) **Histograms of Numerical Variables:** Shows the distribution of values for each numerical variable in the dataset.

   (b) **Heatmap of Correlation Matrix:** Visualizes the degree of correlation between different variables in the dataset using a heatmap.

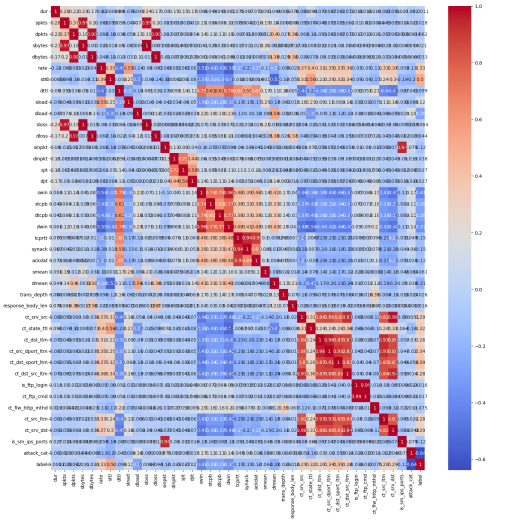


Figure 2: Heatmap of the Dataset

   (c) **Box plots of numerical variables:** Creates box plots for each numerical variable in the dataset, grouped by the different attack categories, to identify differences in the distributions across categories.

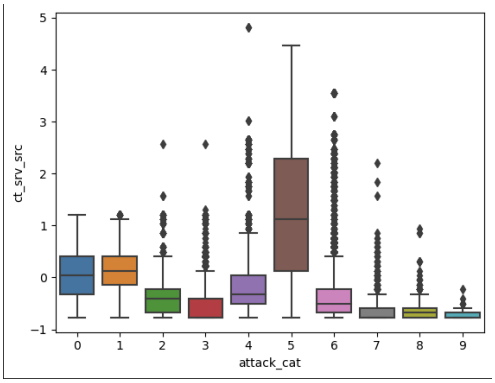


Figure 3: Boxplot of ct_srv _src w.r.t attack _cat

3. **Selection of the Features:** Feature selection aims to select the most relevant features that can improve the performance of the classifiers. We perform the feature selection on the dataset using the SelectKBest and mutual information score.

   - During fitting, mutual information scores are calculated for each feature with respect to the target variable.
   - The SelectKBest object selects the k features (k = 10) with the highest mutual information scores.
   - The mutual information criterion measures the amount of information shared between the feature and target variables, which is a dependable and streamlined approach for recognizing pertinent features.
   - This method is useful for feature selection when there is a non-linear relationship between the features and target variable or when there are complex interactions among the features.
   - The selected features are assigned to the names of the selected features using the get _support() method of the selector object.

```
from sklearn.feature_selection import SelectKBest, mutual_info_classif

X = df.drop(['label'], axis=1)
y = df['label']

# Select features using mutual information score
selector = SelectKBest(mutual_info_classif, k=10)
selector.fit(X, y)

# Get the selected features
selected_features = X.columns[selector.get_support(indices=True)]
X = X[selected_features]
```

Figure 4: Code Snippet - Feature Selection

Through this approach, we can enhance the efficiency of our Machine Learning models by decreasing the size of the input data and eliminating noise and redundancy. We have implemented this approach in both the Training and Testing Dataset notebooks.

4. **Selection and Training of the Models:** Then we to select different types of Neural Networks, such as CNNs, RCNNs, and NNs (MLP), and compare their performance with traditional classifiers such as Random Forest, Decision Tree, XGBoost, SVM and Logistic Regression. We have obtained the performance results for these models. This is done in both the notebooks of the Training Dataset and Testing Dataset.

5. **Evaluation of the Models:** The performance of the classifiers will be evaluated using metrics such as Accuracy, Precision, Recall, F1-Score, ROC Curves Graphs and AUC Score Plots. We have used the classification report and confusion matrix for each of the classifiers to identify the misclassified samples and analyze the results. This is done in both the notebooks of the Training Dataset and Testing Dataset.

2.2 Supervised Traditional Classifiers

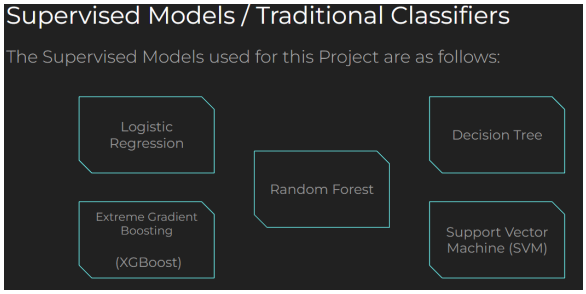


Figure 5: Supervised Models

1. **Logistic Regression Model:**

   As we know that Logistic Regression (LR) is a common statistical technique which used for binary classification problems, where the objective is to predict a binary outcome (i.e., 0 or 1). Using the input features, LR can be used in the context of our project for the network intrusion detection to determine whether a specific network connection is malicious or benign.

   The LR model fits to its purpose as a perfect baseline model for this project as it includes being computationally effective and very simple to apply, which makes it appropriate for evaluating big datasets like the UNSW-NB15 dataset. In thought of a larger scope, it can as well be useful for Security analysts who can grasp the elements that went into the categorization choice thanks to LR's ability to produce results that are easy to interpret.

   Considering how easy it is to understand and use, it is utilized for network intrusion detection.

2. **Decision Tree Model:** Because it can handle both continuous and categorical data, the decision tree model is good choice for the UNSW-NB15 dataset as it's a good fit for a dataset with a variety of feature types. Decision tree models can be helpful in identifying patterns and rules that can differentiate between valid and malicious traffic using our model in the UNSW-NB15 dataset for network intrusion detection. It works by splitting into smaller sub-sets or sub-trees to make the predictions.

   Decision trees also have the reputation of being interpretable models, which implies they can shed light on the decision-making process. Understanding the model's prediction process and spotting significant dataset features can both benefit from this. Last but not least, decision trees are computationally effective, which makes them suitable for sizable datasets like the UNSW-NB15 dataset.

   Model Architecture

   - No. of Leaves: 3
   - Depth of Tree: 2
   - Class Names: [Attack, Normal]

3. **Random Forest Model:** We utilized the Random Forest model, a form of ensemble learning method that creates many decision trees and combines them to provide a more precise and stable prediction, to build a model with the UNSWNB-15 dataset.

   Random Forest is a popular choice for network intrusion detection since it can handle large datasets with highdimensional features, handles noise and outliers, and can handle categorical and continuous data. It is an effective option for identifying sophisticated attacks because it can recognize interactions and non-linear correlations between features.

   To conclude, the Random Forest Classifier is a good model for the UNSWNB-15 dataset since it can handle big and complicated datasets and reliably identify network intrusion attacks.

   Model Architecture

   - max _depth: 4
   - n_Estimators: 150
   - No. of Trees: 100

4. **Support Vector Model:**

   Due to the SVM model's capacity for handling high-dimensional feature spaces, which makes it ideal for datasets with numerous features like the UNSW-NB15 dataset, it has been employed in the model with UNSWNB-15 dataset. Using kernel methods, SVM is also renowned for its capacity to handle non-linearly separable data. By doing so, it can translate the original feature space to a higher-dimensional space where it might become linearly separable.

   In the case of non-linearly separable data which is the UNSW-NB15 dataset, SVM helps maps the input data to a higher-dimensional space, where it becomes linearly separable. This mapping is achieved by applying a kernel function to the input data, such as the RBF kernel used in this model.

   The effectiveness of the SVM model depends on the kernel selection, which may affect its performance. We are using the kernel which is by default, the SVC class uses: the RBF kernel, which is a Gaussian radial basis function kernel. This kernel is widely used in practice because it can effectively model complex, non-linear decision boundaries.

   Based on the numerous attributes retrieved from the network traffic in the UNSW-NB15 dataset, SVM can be used to categorize connections as benign or malicious.

   Model Architecture

   - Classifier: SVC()
   - Kernal: Default - RBF

5. **XGBoost Model:** Extreme Gradient Boosting, or XGBoost, is an ensemble learning technique that has gained popularity for its outstanding performance on structured datasets. it is a strong and effective algorithm that can handle a big amount of data with high dimensionality. It operates by first creating a string of decision trees, then sequentially adding each one while learning from the mistakes of the preceding ones.

   Because of its outstanding performance on structured datasets and capacity to handle high dimensional data, XGBoost is a viable algorithm in the context of the UNSWNB-15 dataset. Making it ideal for the classification problem of detecting network intrusions.

2.3 Unsupervised Models

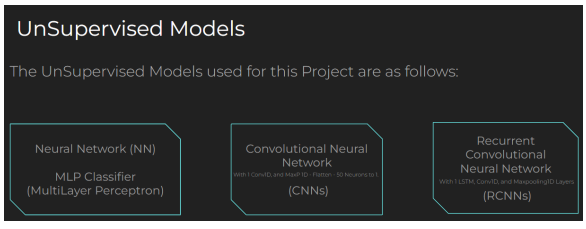


Figure 6: Unsupervised Models

1. **Neural Network (MLP Classifier) Model:** The neural network model employed is with the Multi-Layer Perceptron (MLP) Classifier, which is popularly used in a variety of classification tasks. A built-in function called MLPClassifier can be used to address a range of classification issues. It trains via backpropagation.

   The UNSW-NB15 dataset is a sizable and intricate dataset with several features that are highly correlated with one another.

   It is well known that neural networks, particularly MLP, can handle high-dimensional data with non-linear connections between features.

   MLPClassifier is thus an appropriate model for the UNSW-NB15 dataset.

   Model Architecture

   - Classifier: MLPClassifier()
   - Activation Funtion: Default - ReLU
   - Optimization Algorithm: Default - SGD
   - Hidden Layer: Default - 1

2. **CNN Model:**

   In this scenario, The UNSWNB-15 dataset's network traffic data we have used the Convolutional Neural Network (CNN) Model.

   Because of its capacity to learn hierarchical representations of the input data, which may capture the temporal patterns and dependencies in the dataset, the CNN is an appropriate candidate for the UNSWNB-15 dataset.

   While the max pooling layer aids in reducing the spatial dimensionality of the data, the 1D convolutional layer in the CNN can identify local patterns in the data. Additionally, a non-linear mapping between the extracted features and the output classes is provided by the fully connected layers.

   Model Architecture

   - 1 Conv1D Layer: Filters: 64, KernalSize: 3, Activation: ReLU
   - 1 MaxPooling Layer: PoolSize: 2
   - Flatten: Converts Multi-Dimensional input data to 1Dim array.
   - Dense (50), Activation: ReLU
   - Dense (1), Activation: Sigmoid
   - Optimizer: Adam, Loss: BCE - Binary Cross Entropy
   - Epochs: 10, Batch Size: 32

3. **RCNN Model:**

   In our project, we employed a version of a neural network called the RCNN (Recurrent Convolutional Neural Network), which combines recurrent and convolutional layers.

   The model architecture consists of a 1D convolutional layer, a max pooling layer, an LSTM layer, and a dense layer for binary classification with a sigmoid activation function.

   For the particular classification problems like intrusion detection, the RCNN model has been used to the UNSWNB-15 dataset because it is a well-liked and successful method. The model can recognize both short- and long-term temporal patterns in the data because it combines convolutional layers with recurrent layers. In this dataset, where network traffic sequences can vary in length, the LSTM layer's ability to handle sequences of different lengths is crucial. The model's high accuracy rating can be credited to its capacity to recognize intricate features and recurring patterns in the data.

   Model Architecture

   - 1 Conv1D Layer: Filters: 64, KernalSize: 3, Activation: ReLU
   - 1 MaxPooling Layer: PoolSize: 2
   - LSTM Layer: 100 Units
   - Dense (1), Activation: Sigmoid
   - Optimizer: Adam, Loss: BCE - Binary Cross Entropy
   - Epochs: 10, BatchSize: 32

3 Results

SuperVised Models		Logistic Regression	Decision Tree	SVM	XGBoost	Random Forest
TRAINING SET	Accuracy:	0.89637	1.0	0.99971	1.0	1.0
	F1-Score:	0.90	1.00	1.00	1.00	1.00
TESTING SET	Accuracy:	0.93125	1.0	0.99902	1.0	1.0
	F1-Score:	0.93	1.00	1.00	1.00	1.00

UnSuperVised Models		NN (MLP Classifier)	CNN	RCNN
TRAINING SET	Accuracy:	0.99994	0.99996	1.0
	F1-Score:	1.00	1.00	1.00
TESTING SET	Accuracy:	0.99987	0.999878	0.99991
	F1-Score:	1.00	1.00	1.00

Figure 7: Result of All Models across both the sets.

The table above shows the outcomes/results of models used in our project using supervised and unsupervised machine learning models on the UNSW-NB15 dataset.

From the results which we have obtained, we can see that the all of models, whether supervised or unsupervised, were able to achieve high accuracy and F1-scores for both the training and testing sets, as can be seen from the data.

MLP Classifier achieved an accuracy of 0.99994 on the training set for unsupervised models. The test set, and 0.99987 on it. On the training set, CNN had an accuracy of 0.99996, and on the testing set, it had an accuracy of 0.999878. On the training set, RCNN had perfect accuracy, and on the testing set, it had an accuracy of 0.99991.

For both the training and testing sets, all unsupervised models had perfect F1 scores. Overall, the accuracy of the unsupervised models outperformed that of the supervised models, showing that these models were more effective at identifying network intrusions. RCNN was discovered to be the model that performed the best overall, achieving 0.99991 accuracy on the testing set and perfect accuracy on the training set.

The models' high accuracy and F1 scores show that the feature selection method employed in the project was successful in choosing pertinent features for the models. The findings imply that deep learning models, such as RCNN, can perform better for network intrusion detection than conventional machine learning classifiers.

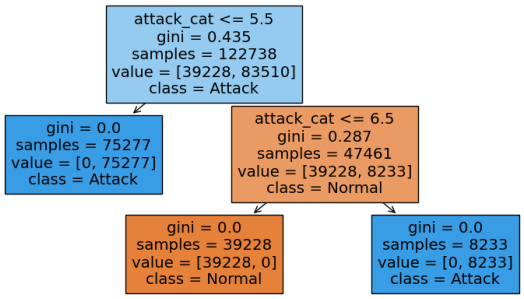


Figure 8: Decision Tree Plot - Training Set

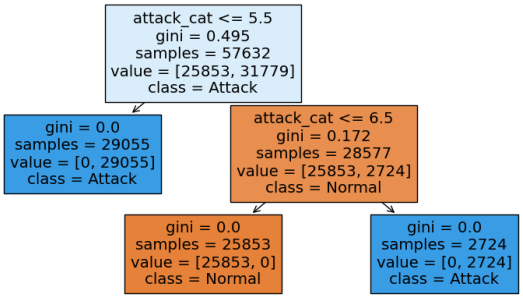


Figure 9: Decision Tree Plot - Testing Set

Among the supervised models, all of them except Logistic Regression achieved perfect accuracy on the training set, while Logistic Regression achieved an accuracy of 0.89637. For the testing set, all supervised models achieved near-perfect accuracy, with the exception of SVM which achieved an accuracy of 0.99902. The F1-scores for all supervised models were perfect for both the training and testing sets ranging between 0.90 and 1.00.

A better level of accuracy and F1-scores of 1.00 for both the training and testing sets were attained by the unsupervised models, which also included NN (MLP Classifier), CNN, and RCNN.

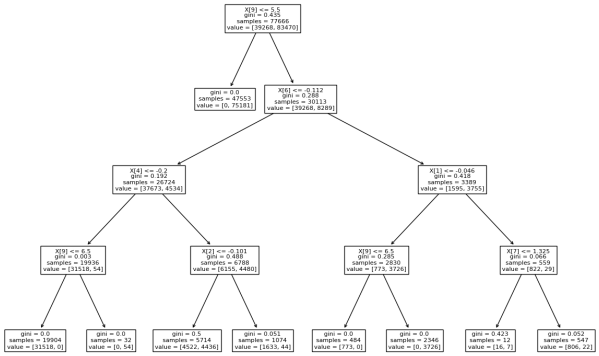


Figure 10: Tree Plot Created using Random Forest Training Set

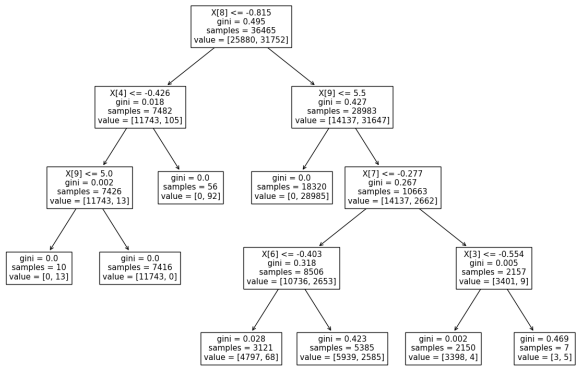
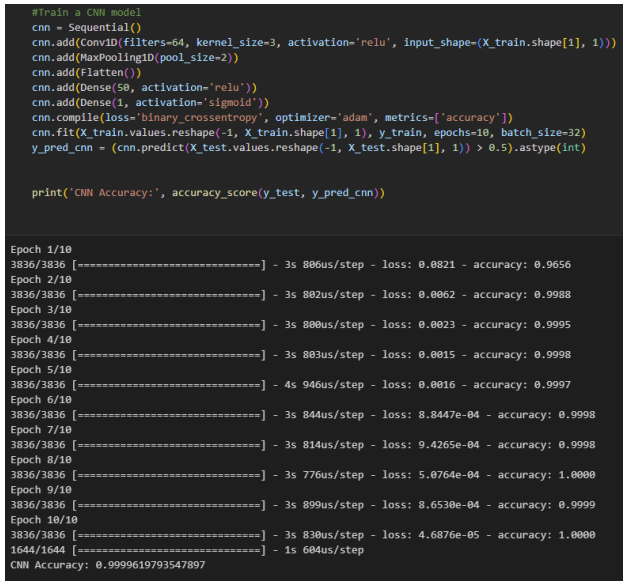


Figure 11: Tree Plot Created using Random Forest Testing Set



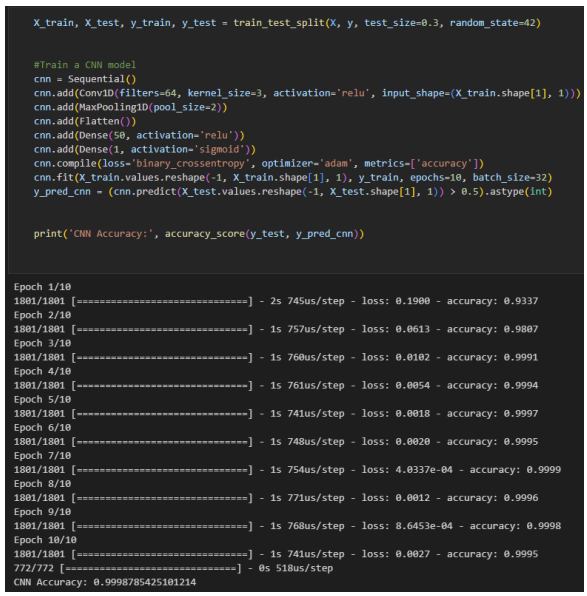
```
#Train a CNN model
cnn = Sequential()
cnn.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)))
cnn.add(MaxPooling1D(pool_size=2))
cnn.add(Flatten())
cnn.add(Dense(50, activation='relu'))
cnn.add(Dense(1, activation='sigmoid'))
cnn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
cnn.fit(X_train.values.reshape(-1, X_train.shape[1], 1), y_train, epochs=10, batch_size=32)
y_pred_cnn = (cnn.predict(X_test.values.reshape(-1, X_test.shape[1], 1)) > 0.5).astype(int)


print('CNN Accuracy:', accuracy_score(y_test, y_pred_cnn))
```

```
Epoch 1/10
3836/3836 [==============================] - 3s 806us/step - loss: 0.0821 - accuracy: 0.9656
Epoch 2/10
3836/3836 [==============================] - 3s 802us/step - loss: 0.0062 - accuracy: 0.9988
Epoch 3/10
3836/3836 [==============================] - 3s 800us/step - loss: 0.0023 - accuracy: 0.9995
Epoch 4/10
3836/3836 [==============================] - 3s 803us/step - loss: 0.0015 - accuracy: 0.9998
Epoch 5/10
3836/3836 [==============================] - 4s 946us/step - loss: 0.0016 - accuracy: 0.9997
Epoch 6/10
3836/3836 [==============================] - 3s 844us/step - loss: 8.8447e-04 - accuracy: 0.9998
Epoch 7/10
3836/3836 [==============================] - 3s 814us/step - loss: 9.4265e-04 - accuracy: 0.9998
Epoch 8/10
3836/3836 [==============================] - 3s 776us/step - loss: 5.0764e-04 - accuracy: 1.0000
Epoch 9/10
3836/3836 [==============================] - 3s 899us/step - loss: 8.6530e-04 - accuracy: 0.9999
Epoch 10/10
3836/3836 [==============================] - 3s 830us/step - loss: 4.6876e-05 - accuracy: 1.0000
1644/1644 [==============================] - 1s 604us/step
CNN Accuracy: 0.9999619793547897
```

Figure 12: Output of CNN for Training Set

Overall, the unsupervised models fared better than the supervised models due to their ability to obtain flawless accuracy and F1-scores, which shows that they were able to learn and generalize the patterns and characteristics of the UNSW-NB15 dataset without having to be trained on labeled data.

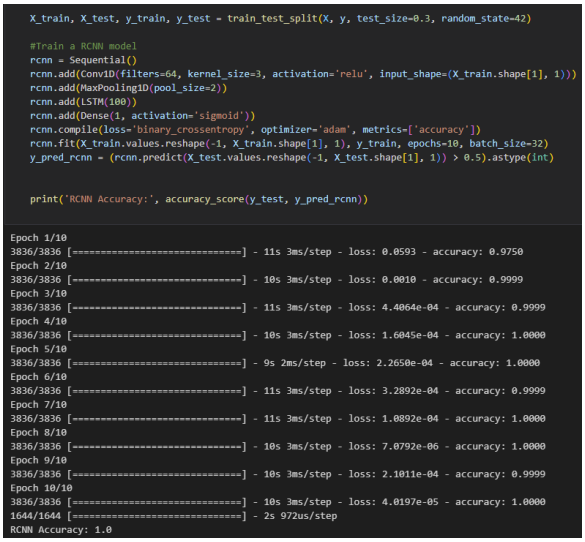
Unsupervised models may learn from the raw data without needing any labeled information, but labeling huge datasets can be a laborious and time-consuming operation.



```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)


#Train a CNN model
cnn = Sequential()
cnn.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)))
cnn.add(MaxPooling1D(pool_size=2))
cnn.add(Flatten())
cnn.add(Dense(50, activation='relu'))
cnn.add(Dense(1, activation='sigmoid'))
cnn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
cnn.fit(X_train.values.reshape(-1, X_train.shape[1], 1), y_train, epochs=10, batch_size=32)
y_pred_cnn = (cnn.predict(X_test.values.reshape(-1, X_test.shape[1], 1)) > 0.5).astype(int)


print('CNN Accuracy:', accuracy_score(y_test, y_pred_cnn))
```

```
Epoch 1/10
1801/1801 [==============================] - 2s 745us/step - loss: 0.1900 - accuracy: 0.9337
Epoch 2/10
1801/1801 [==============================] - 1s 757us/step - loss: 0.0613 - accuracy: 0.9807
Epoch 3/10
1801/1801 [==============================] - 1s 760us/step - loss: 0.0102 - accuracy: 0.9991
Epoch 4/10
1801/1801 [==============================] - 1s 761us/step - loss: 0.0054 - accuracy: 0.9994
Epoch 5/10
1801/1801 [==============================] - 1s 741us/step - loss: 0.0018 - accuracy: 0.9997
Epoch 6/10
1801/1801 [==============================] - 1s 748us/step - loss: 0.0020 - accuracy: 0.9995
Epoch 7/10
1801/1801 [==============================] - 1s 754us/step - loss: 4.0337e-04 - accuracy: 0.9999
Epoch 8/10
1801/1801 [==============================] - 1s 771us/step - loss: 0.0012 - accuracy: 0.9996
Epoch 9/10
1801/1801 [==============================] - 1s 768us/step - loss: 8.6453e-04 - accuracy: 0.9998
Epoch 10/10
1801/1801 [==============================] - 1s 741us/step - loss: 0.0027 - accuracy: 0.9995
772/772 [==============================] - 0s 518us/step
CNN Accuracy: 0.9998785425101214
```

Figure 13: Output of CNN for Testing Set



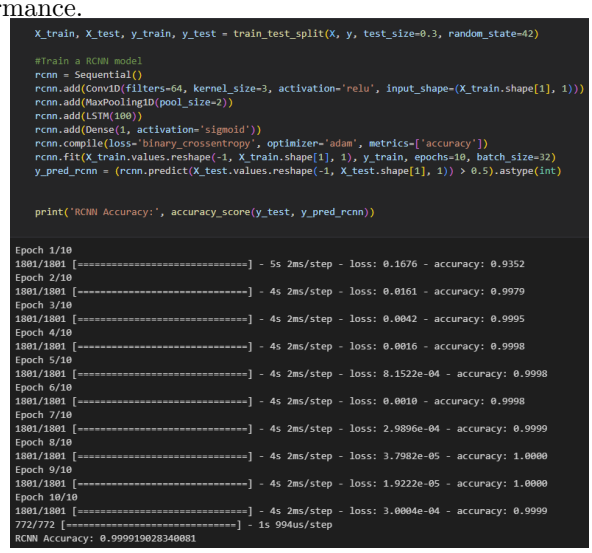
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

#Train a RCNN model
rcnn = Sequential()
rcnn.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)))
rcnn.add(MaxPooling1D(pool_size=2))
rcnn.add(LSTM(100))
rcnn.add(Dense(1, activation='sigmoid'))
rcnn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
rcnn.fit(X_train.values.reshape(-1, X_train.shape[1], 1), y_train, epochs=10, batch_size=32)
y_pred_rcnn = (rcnn.predict(X_test.values.reshape(-1, X_test.shape[1], 1)) > 0.5).astype(int)


print('RCNN Accuracy:', accuracy_score(y_test, y_pred_rcnn))
```

```
Epoch 1/10
3836/3836 [==============================] - 11s 3ms/step - loss: 0.0593 - accuracy: 0.9750
Epoch 2/10
3836/3836 [==============================] - 10s 3ms/step - loss: 0.0010 - accuracy: 0.9999
Epoch 3/10
3836/3836 [==============================] - 11s 3ms/step - loss: 4.4064e-04 - accuracy: 0.9999
Epoch 4/10
3836/3836 [==============================] - 10s 3ms/step - loss: 1.6045e-04 - accuracy: 1.0000
Epoch 5/10
3836/3836 [==============================] - 9s 2ms/step - loss: 2.2650e-04 - accuracy: 1.0000
Epoch 6/10
3836/3836 [==============================] - 11s 3ms/step - loss: 3.2892e-04 - accuracy: 0.9999
Epoch 7/10
3836/3836 [==============================] - 11s 3ms/step - loss: 1.0892e-04 - accuracy: 1.0000
Epoch 8/10
3836/3836 [==============================] - 10s 3ms/step - loss: 7.0792e-06 - accuracy: 1.0000
Epoch 9/10
3836/3836 [==============================] - 10s 3ms/step - loss: 2.1011e-04 - accuracy: 0.9999
Epoch 10/10
3836/3836 [==============================] - 10s 3ms/step - loss: 4.0197e-05 - accuracy: 1.0000
1644/1644 [==============================] - 2s 972us/step
RCNN Accuracy: 1.0
```

Figure 14: Output of RCNN for Training Set

The CNN and RCNN models, as well as the network traffic data in the UNSW-NB15 dataset, were created expressly to deal with sequential data. These models were well suited for this kind of dataset since they were able to capture the temporal patterns and dependencies in the data.

In addition, the RCNN model included both convolutional and recurrent layers, enabling it to concurrently extract spatial and temporal characteristics from the data, outperforming other models in terms of performance.



```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

#Train a RCNN model
rcnn = Sequential()
rcnn.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)))
rcnn.add(MaxPooling1D(pool_size=2))
rcnn.add(LSTM(100))
rcnn.add(Dense(1, activation='sigmoid'))
rcnn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
rcnn.fit(X_train.values.reshape(-1, X_train.shape[1], 1), y_train, epochs=10, batch_size=32)
y_pred_rcnn = (rcnn.predict(X_test.values.reshape(-1, X_test.shape[1], 1)) > 0.5).astype(int)


print('RCNN Accuracy:', accuracy_score(y_test, y_pred_rcnn))
```

```
Epoch 1/10
1801/1801 [==============================] - 5s 2ms/step - loss: 0.1676 - accuracy: 0.9352
Epoch 2/10
1801/1801 [==============================] - 4s 2ms/step - loss: 0.0161 - accuracy: 0.9979
Epoch 3/10
1801/1801 [==============================] - 4s 2ms/step - loss: 0.0042 - accuracy: 0.9995
Epoch 4/10
1801/1801 [==============================] - 4s 2ms/step - loss: 0.0016 - accuracy: 0.9998
Epoch 5/10
1801/1801 [==============================] - 4s 2ms/step - loss: 8.1522e-04 - accuracy: 0.9998
Epoch 6/10
1801/1801 [==============================] - 4s 2ms/step - loss: 0.0010 - accuracy: 0.9998
Epoch 7/10
1801/1801 [==============================] - 4s 2ms/step - loss: 2.9896e-04 - accuracy: 0.9999
Epoch 8/10
1801/1801 [==============================] - 4s 2ms/step - loss: 3.7982e-05 - accuracy: 1.0000
Epoch 9/10
1801/1801 [==============================] - 4s 2ms/step - loss: 1.9222e-05 - accuracy: 1.0000
Epoch 10/10
1801/1801 [==============================] - 4s 2ms/step - loss: 3.0004e-04 - accuracy: 0.9999
772/772 [==============================] - 1s 994us/step
RCNN Accuracy: 0.999919028340081
```

Figure 15: Output of RCNN for Testing Set

We did notice that in supervised learning, despite giving good accuracy. The Logistic regression model seems to have the a lower accuracy (i.e., 89% in Training, (93% in Testing) when compared to other supervised learning models/traditional classifiers.

There could be a various reasons for why the LR model on the UNSW-NB15 dataset performed less accurately than other supervised and unsupervised models.

One possibility is that the dataset is unbalanced, with the attack class having significantly less samples than the normal class.

The model in this situation might be biased in favor of the majority class, which would reduce the accuracy for the minority class. Another possibility is that the model's training characteristics were insufficiently informative to adequately represent the complexity of the data. The non-linear correlations between the features and the target variable may not be captured by logistic regression because it is a linear model.

More, Evaluation Metrics: Precision-Recall Curve, ROC Curve, AUC Scores Histogram, Classification Reports, and Confusion Matrix Are Included/Shown in the Project Code (.ipynb) Files for more comprehensive evaluation of the model's performance. Few of them are shown here as follows:

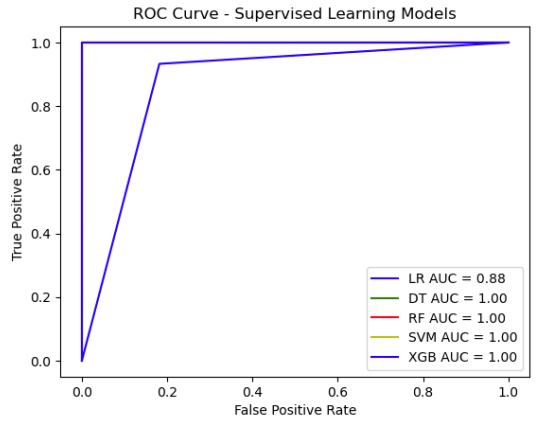


Figure 16: ROC Curve - Supervised Models - Training Set

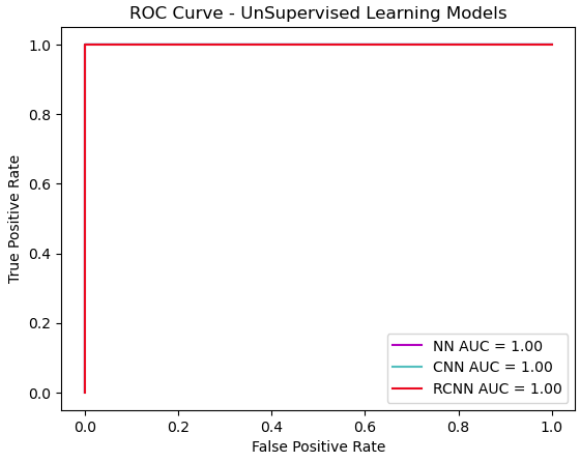


Figure 17: ROC Curve - Unsupervised Models - Training Set

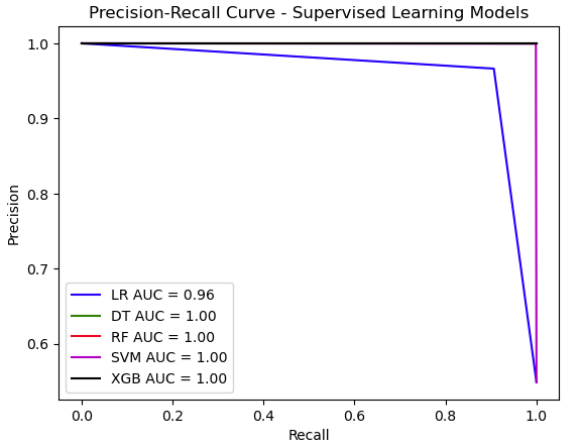


Figure 18: Precision ReCall Curve - Supervised Models - Testing Set

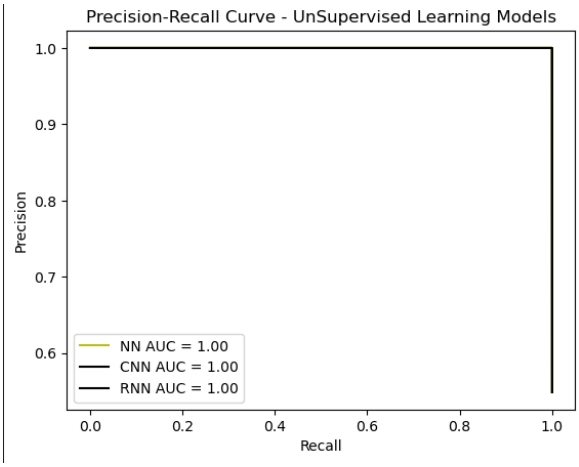


Figure 19: Precision Recall Curve - Unsupervised Models - Testing Set

```
Logistic Regression Accuracy: 0.8963747314791932
Decision Tree Accuracy: 1.0
Random Forest Accuracy: 1.0
SVM Accuracy: 0.9997148451609223
XGBoost Accuracy: 1.0
```

Figure 20: Overall Accuracy - Supervised Models - Training Set

```
Neural Network Accuracy: 0.9999429690321845
CNN Accuracy: 0.9999619793547897
RCNN Accuracy: 1.0
```

Figure 21: Overall Accuracy - Unsupervised Models - Training Set

```
Logistic Regression Accuracy: 0.931255060728745
Decision Tree Accuracy: 1.0
Random Forest Accuracy: 1.0
SVM Accuracy: 0.9990283400809716
XGBoost Accuracy: 1.0
```

Figure 22: Overall Accuracy - Supervised Models - Testing Set

```
Neural Network Accuracy: 0.9998785425101214
CNN Accuracy: 0.9998785425101214
RCNN Accuracy: 0.999919028340081
```

Figure 23: Overall Accuracy - Unsupervised Models - Testing Set

Thus, using the UNSW-NB15 dataset on these models, the results show the efficacy of both supervised and unsupervised models for network intrusion detection. In conclusion, However, the CNN and RCNN models performed better than the supervised models, indicating that unsupervised learning to be a promising area for further study.

4 Conclusion

In our project, We used the UNSW-NB15 dataset using a variety of supervised and unsupervised machine learning models to categorize the network traffic as either normal or malicious. We used MLP classifier, CNN, and RCNN as unsupervised models, as well as logistic regression, decision tree, SVM, XGBoost, and random forest as supervised models.

The proposed approach for identifying network intrusions using machine learning was successful since it showed that neural networks, particularly RCNNs, outperformed conventional classifiers in this task.

The exploring of several neural network types and the application of feature selection techniques to enhance model performance comprised the project's core contribution. These results imply that neural networks have considerable promise for network intrusion detection and network security enhancement.

This project's one of the key contribution is that it uses the UNSW-NB15 dataset to give a thorough comparison of supervised and unsupervised machine learning models for network intrusion detection.

The significance of choosing relevant features to enhance the performance of machine learning models is one of the key lessons acquired from this project. By removing unnecessary or redundant features from the dataset and reducing its dimensionality, feature selection can improve model performance and speed up model training.

The findings suggest that in identifying network threats, unsupervised models may be superior to supervised ones.

4.1 Future Work

The possible avenues for Future work are as follows:

To enhance the effectiveness of network intrusion detection systems, it would be intriguing to investigate more sophisticated deep learning models like Generative Adversarial Networks (GANs).

The integration of the suggested approach into a real-time network intrusion detection system would also be a worthwhile future study because it would enable the real-time detection of network intrusions, which is essential for enhancing the security of computer networks.

The problem of imbalanced datasets in network intrusion detection can be addressed by using GANs to provide synthetic data that can be used to train machine learning models.

To increase the detection of network attacks, look into different unsupervised models or create hybrid models that blend supervised and unsupervised methods. Studying the effect of various feature engineering strategies on model performance is another potential direction for future research. The further study could be expanded to incorporate additional datasets and assess the generalizability of the models.

References

[1] Mamoun Alazab, Michael Hobbs, and Jemal Abawajy. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, 60:84–102, 2016.

[2] Sebastian Garcia, Markus Grill, Lukas Stiborek, Juan Manuel Zuniga, and Mariemma I Aguayo-Torres. An empirical comparison of botnet detection methods. *Computers & Security*, 45:100–123, 2014.

[3] Md Rafiul Islam and Kazi Mohammed Ahmed. Machine learning approaches for network intrusion detection: A comprehensive survey. *IEEE Access*, 7:27459–27484, 2019.

[4] Ahmed Moustafa, Jill Slay, and Gregory Creech. Unsw-nb15: A comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *Military Communications and Information Systems Conference (MilCIS)*, pages 1–6. IEEE, 2015.

[5] Nour Moustafa and Jill Slay. The evaluation of network anomaly detection systems: Statistical analysis of the unsw-nb15 dataset and the comparison with the kdd99 dataset. *Information Security Journal: A Global Perspective*, 25(1-3):1–14, 2016.

[6] Sourav Mukherjee, Ananda Roy Chowdhury, Shukla Das, Sayan Chakraborty, and Mita Nasipuri. A comparative study of deep learning approaches for network intrusion detection. *Future Generation Computer Systems*, 107:1063–1077, 2020.

Github Link for the Project - CodeBase and it's Corresponding files: ML Project Github Respository.