SASTRA DEEMED TO BE UNIVERSITY

(A University under section 3 of the UGC Act, 1956)

CSE425 - Machine Learning Essentials

End Semester Project

Hybrid Deep Learning for Intrusion Detection: A CNN-LSTM Approach with Comparative Analysis

Submitted by
Kanchanapalli Leelankitha
(Reg No.: 125018041, Btech- Computer Science and Business Systems)

School of Computing
SASTRA Deemed to be University
TIRUMALAISAMUDRAM
THANJAVUR —613 401
TAMIL NADU, INDIA

TABLE OF CONTENTS

ABSTRACT

This project addresses the problem of network intrusion detection by reviewing and comparing various machine learning models that include a deep learning architecture based on CNN-LSTM, SVM, and Gaussian Naive Bayes. In this project, the UNSW-NB15 dataset has been used to evaluate real network traffic data. These models are tested for whether they could effectively classify network behavior as either normal or anomalous. The comprehensive dataset with different features representing network activities makes for an all-rounded estimation of model performance. Classification accuracy, precision, recall, and F1-score are used in determining the most appropriate method for intrusion detection.

Our results indicate that the CNN-LSTM model significantly outperforms the traditional SVM and Gaussian Naive Bayes classifiers of high overall accuracy of 97.5% and robustness in handling complex sequential data and exhibits good performance while learning. Hyperparameter tuning, early stopping, and other optimization techniques have also improved the performance of the deep learning model. But traditional models, like SVM, would not be proper for such highly dynamic correlated features along with the above-mentioned setting. Probabilistic Gaussian Naive Bayes is very fast in computation but will yield lower classification accuracy while the complexity of a data set increases. Conclusion The CNN-LSTM hybrid model will manifest a more distinctive ability to detect network intrusions; thus this work shows how deep learning based techniques can leverage and utilize pattern recognition and sequence processing. This comparative analysis provides valuable insight into access to understand which proper models are selected for applications in cyber security by balancing accuracy, computational efficiency, and characteristics of the data involved.

1 INTRODUCTION

1.1 Dataset

The UNSW-NB15 dataset is one of the most important datasets within the research in the realm of cybersecurity, particularly when it comes to the development and testing of a network intrusion detection system, or otherwise commonly referred to as NIDS. This dataset encompasses all ranges of normal network traffic and various attack types on modern networks, including DoS, exploits, fuzzers, and reconnaissance attacks. The use of the efficient simulation of realistic network behavior by the IXIA PerfectStorm tool makes this dataset more applicable to the current network environments. Its diversity and realistic data representation enable researchers and practitioners to design and assess the effectiveness of intrusion detection mechanisms, thus enhancing cybersecurity measures in real-world applications.

1.2 Objectives

- **Task(T):** Design an algorithm set to categorize network activity into one of the following classes: normal or anomalous. A hybrid deep learning model, namely CNN-LSTM, and conventional models, such as SVM and Gaussian Naive Bayes, will be used.

- **Problem(P):** IDS generally suffer from high false positives and low adaptability to new types of attacks. Models, whether based on traditional approaches, have good accuracy over known attacks but often fail to maintain robustness against emerging ones. How can classification

accuracy and recall while minimizing false alarms for known and unknown intrusions be improved in dynamic network environments is the problem.

- **Expectation (E):** It is a programmed hybrid approach based on Convolutional Neural Networks and Long Short-Term Memory, anticipating higher performance than traditional machine learning approaches, like Support Vector Machine and Gaussian Naive Bayes, in terms of detection capabilities, classification accuracy, and recall in the case of more complex network traffic patterns and newly emerging attacks.

1.3 Approach

It uses three different models to classify the network intrusions by combined use of hybrid deep learning and traditional machine learning techniques:

1. **CNN-LSTM:** This is the hybrid deep architecture combining feature extraction capabilities of CNN with sequential processing capabilities by Long Short-Term Memory. Such architecture can capture spatial patterns in network traffic data using the CNN part and temporal dependencies by the LSTM part to recognize complex patterns over time. Such a hybrid approach will serve as an integrated mechanism that strengthens the intrusion detection accuracy, both for known and unknown intrusions.

2. **SVM:** SVM, or Support Vector Machine, is a traditional supervised learning algorithm that has been proven to be highly robust even in classification of two-classes of input samples. The SVM finds the optimal hyperplane that best separates different classes in the feature space; hence the effectiveness of SVM applied in intrusion detection. The preprocessed UNSW-NB15 dataset will be trained by SVM to classify between normal and anomalous network behaviors.

3. **Gaussian Naive Bayes:** This probabilistic model uses Bayes' theorem under the assumption that features are Gaussian distributed. Gaussian Naive Bayes classically proves highly efficient for classification-type problems and gives a benchmark by which to evaluate more complex models. By computing the posterior probabilities of every class based on observed features, this model will contribute towards evaluating how well it would be able to identify intrusions on the network traffic.

1.4 Results Overview

Three algorithms, namely CNN-LSTM, SVM, and Gaussian Naive Bayes, were tested against the UNSW-NB15 dataset, classifying the network behaviors as normal or anomalous. The CNN-LSTM model showed excellent capabilities in the classification with an accuracy of 97.5% and very high recall of the model, which means that it is quite effective in detecting anomalous behaviors, particularly more advanced attack patterns. Moreover, this hybrid model demonstrates excellent adaptability toward different types of attacks and drastically reduces the rate of false positives compared to traditional methods. In contrast, the SVM achieved an accuracy of approximately 89.2%. Though it differentiated normal and anomalous traffic quite well, its inferior recall led to more false negatives for certain attack types. The SVM was indeed great at learning linear separability but fared worse when the pattern in the data was more complex and nonlinear. Gaussian Naive Bayes was used as a baseline with an accuracy of

around 50.46%. It was computationally efficient and implementable but did not incorporate the intricate relationships in the data, due to an assumption of a Gaussian distribution, and thus had lower accuracy rates.

2 RELATED WORK

In the last few years, many developments have been made in the research with regards to the network intrusion detection systems (IDS) that use machine learning techniques. Multiple research works have analyzed various models and their effectiveness in detecting both known and unknown attacks. These approaches include the use of hybrid models that combine the benefits from various architectures. For instance, it was demonstrated that the CNN-LSTM model performed outstandingly well in the detection of complex patterns of network traffic maximizing classification accuracy and recall significantly (Yin et al., 2017). The combination of spatial and temporal features captured by this model is very effective for intrusion detection by the dynamic nature of the environment.

Other classical models of machine learning such as Support Vector Machines (SVM) and Gaussian Naive Bayes have also been heavily relied on in the IDS area. As far as binary classification goes, SVMs are known to perform great; however, they easily get trapped with non-linear data patterns and, therefore, can fail to identify many attacks (Cortes & Vapnik, 1995). On the other hand, while Gaussian Naive Bayes may seem so computationally efficient, they do not work very well due to the basic underlying assumptions regarding the distribution of data, thus implying low accuracy in complex datasets (Murphy, 2012).

2.1 Use of Dataset

The UNSW-NB15 dataset, used in this project, has been named as one of the benchmarks to evaluate various IDS models. Such a variety of network traffic scenarios, including both normal and attack behaviors, has been utilized as a benchmark in several studies (Moustafa & Slay, 2015). Moreover, platforms such as Kaggle have facilitated the sharing of datasets and models through which researchers can build upon and further advance existing lines of work in intrusion detection.

2.2 Literature Survey

There is a trend based on the literature to combine hybrid deep learning models for intrusion detection. Among all of them, the combination of CNN and LSTM has gained popularity because it can recognize features both in terms of space and time. The UNSW-NB15 dataset remains up to this day the benchmarked basis for the evaluation of new methods since it presents a realistic way of network traffic. This paper adds to these developments by comparing the hybrid CNN-LSTM approach with traditional approaches such as SVM and Gaussian Naive Bayes to provide a holistic evaluation of intrusion detection techniques.

Study/Source	Model/Approach	Key findings	Dataset
Yin et al. (2017)	Hybrid CNN-LSTM	Demonstrated strong performance in capturing spatial and temporal features for intrusion detection.	Custom Dataset
Cortes & Vapnik (1995)	SVM	Effective for binary classification tasks, but struggles with non-linear data patterns.	Various Datasets
Murphy (2012)	Gaussian Naive Bayes	Computationally efficient but limited by assumptions of data distribution, leading to lower accuracy.	Various Datasets
Moustafa & Slay (2015)	UNSW-NB15 Dataset	Provides a realistic representation of network traffic for evaluating intrusion detection methods.	UNSW-NB15
Kaggle	Various Models	Platform for sharing datasets and models, facilitating research and collaboration in the field.	Various Datasets

3 BACKGROUND

3.1 Models Used

- CNN-LSTM (Convolutional Neural Network - Long Short-Term Memory): CNN-LSTM is a hybrid deep learning architecture. It combines the feature extraction capabilities of CNN with the sequence processing strengths of LSTM networks. Convolutional Neural Networks are effective in identifying spatial hierarchies and patterns within the data, that suits analysis on multidimensional inputs such as network traffic. In the given project, the CNN component is processing incoming data to extract relevant features from the input data. Then the LSTM layer would use these relevant features produced from the CNN component and try to capture some temporal dependencies and patterns over time so that the model can learn from sequences of network traffic data. This combination makes the CNN-LSTM model quite effective for intrusion detection as it can easily recognize complex patterns of intruders, maintaining a high classification accuracy along with a good rate of recall.

- Support Vector Machine (SVM): SVM is the most popular traditional supervised learning algorithm in order to do a task of binary classification. Its working is based upon the concept of finding the optimal hyperplane that can distinguish different classes in the feature space. SVMs are known to work very well for high-dimensional data and against overfitting, especially if the dimensionality is significantly greater than the number of samples. In the project, SVM is considered as a baseline for comparison of the presented hybrid approach of CNN-LSTM and to understand the power of deep learning over traditional methods of machine learning.

- Gaussian Naive Bayes It's a probability classifier based on Bayes' theorem. It assumes features follow a Gaussian distribution. Its computational efficiency and simplicity make it an excellent choice as a baseline in classification problems. However, such assumptions of independence between features and reliance on a Gaussian distribution severely prevent it from being used to represent intrinsic interaction in the data. The Gaussian Naive Bayes model has been taken as a benchmark case in the project, just to compare and contrast its performance with the more complex CNN-LSTM and SVM models.

3.2 Preprocessing Methods

- Data Cleansing; In the first step of preprocessing the data, data cleansing is done. This removes irrelevant or redundant information to be used in the training models. This would include missing values either imputed through statistical methods or eliminated from the records by removing records that include incompletely developed information.

- Feature selection Methods: These are techniques used to find the most important attributes that contribute to the effective and accurate detection of intruders. This process is very helpful in reducing dimensionality, giving a better performance with the model, and making it more interpretable. Such methods include correlation analysis and feature importance scoring, which determine which features are most relevant to a function.

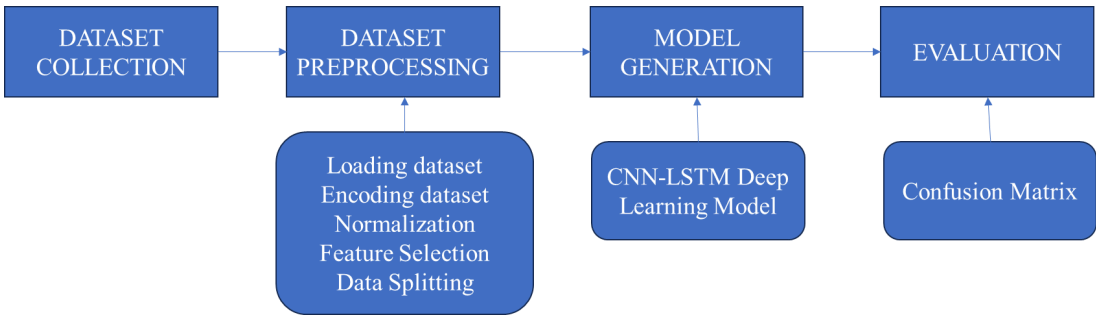
- Normalization/Standardization: Normalization or standardization is used to scale features to a similar range so that no feature will dominate the learning. This is very crucial when in particular algorithms like SVM and Gaussian Naive Bayes are sensitive to the scale of the input data. Min-max normalization or z-score standardization may be used for it.

- Encoding Categorical Variables Using techniques like one-hot encoding or label encoding, the categorical features are encoded as numbers. This means that the models could then process the data that had been generated by the categorical variables appropriately as most of the machine learning algorithms require numerical inputs.

- Data Splitting: The model performance is assessed by splitting the dataset into training, validation, and test sets. This technique ensures that models are trained on one subset of data while validated and tested on unseen data in order to measure their ability to generalize.

4 METHODOLOGY

4.1 Experimental Design



The experimental design for this project involves a systematic approach to evaluate the performance of various machine learning models in detecting network intrusions. The study employs a comparative analysis between a hybrid deep learning model (CNN-LSTM) and traditional machine learning algorithms (SVM and Gaussian Naive Bayes). The following steps outline the experimental design:

1. **Dataset Selection**: The UNSW-NB15 dataset is utilized due to its comprehensive representation of modern network traffic and a variety of attack types. The dataset includes both normal and anomalous traffic, providing a rich basis for model training and evaluation.
2. **Model Implementation:** Three models—CNN-LSTM, SVM, and Gaussian Naive Bayes—are implemented. The CNN-LSTM model is trained on sequences of network traffic data, while SVM and Gaussian Naive Bayes serve as benchmarks for performance comparison.
3. **Performance Evaluation:** Each model's performance is evaluated using various metrics, including accuracy, recall, precision, F1-score, and area under the ROC curve (AUC-ROC). These metrics provide a comprehensive view of each model's effectiveness in classifying network traffic.
4. **Results Analysis:** The results are analyzed to identify strengths and weaknesses in each model, particularly in relation to their ability to detect complex attack patterns while minimizing false positives.

4.2 Tools and Environment

- **Programming Language:** Python
- **Libraries and Frameworks:**
    a. TensorFlow and Keras for deep learning (CNN-LSTM)
    b. Scikit-learn for implementing SVM and Gaussian Naive Bayes
    c. Pandas and NumPy for data manipulation
    d. Matplotlib and Seaborn for visualizing results
- **Development Environment:** Kaggle Notebook and Google Colab were used for code development, execution, and result visualization.

4.3 Code Location

- **GitHub Repository:** https://github.com/Leelankitha/ML_End_Sem_Project

4.4 Parameters Used

CNN-LSTM Model Architecture

The CNN-LSTM model includes three convolutional blocks, each followed by LSTM layers, for capturing both spatial and temporal patterns in the data.

1. **Input Layer:** Input shape (n_features, 1) — The number of features in the training data and reshaped to fit the model.
2. **First Convolutional Block:**
    a. Conv1D: 16 filters, kernel size of 1, with ReLU activation to extract feature maps.
    b. MaxPooling1D: Pool size of 2 to reduce dimensionality and retain important features.
    c. BatchNormalization: Normalize the output for stable and faster training.
3. **First LSTM Block:**
    a. LSTM: 16 units, return sequences enabled to pass the output to the next LSTM block for sequential data processing.
4. **Second Convolutional Block:**
    a. Conv1D: 32 filters, kernel size of 3, with ReLU activation for deeper feature extraction.
    b. MaxPooling1D: Pool size of 2 to further reduce dimensionality.
    c. BatchNormalization: Helps in stabilizing the learning process.
5. **Second LSTM Block:**
    a. LSTM: 32 units, return sequences enabled for sequential output processing.
6. **Third Convolutional Block:**
    a. Conv1D: 64 filters, kernel size of 5, with ReLU activation to capture more complex features.
    b. MaxPooling1D: Pool size of 2 for further dimensionality reduction.
    c. BatchNormalization: Normalization after pooling.
7. **Third LSTM Block:**
    a. LSTM: 64 units, return sequences disabled (since this is the last LSTM layer) to output final sequence representations.
8. **Dense Layers:**
    a. Dense (64 units): Fully connected layer with ReLU activation to learn from the extracted features.
    b. Dropout: 0.2 applied to prevent overfitting.
9. **Output Layer:**
    a. Dense (10 units): Softmax activation for multi-class classification (10 output classes).
10. **Epochs:** 50 epochs provide enough iterations for the model to learn from the data without overfitting.
11. **Batch Size:** 32 or 64, A batch size of 32 or 64 balances between training speed and memory efficiency. A smaller batch size (like 32) will lead to more updates per epoch but will use less memory, while a larger batch size (64) will allow for faster training but requires more memory.

12. **Early Stopping:** Patience = 5 (stops training if no improvement in the validation loss after 5 epochs and prevents overfitting)

- Optimizer: Adam — Adaptive learning rate optimization.
- Loss Function: Categorical cross-entropy — For multi-class classification.
- Metrics: Accuracy, precision, and recall — To measure performance during training.

Support Vector Machine (SVM) Model Architecture

1. **Kernel**: Linear
2. **C**: Regularization strength (default: 1.0).
3. **Penalty**: Regularization type ('l2' by default).
4. **Loss**: Loss function used ('squared_hinge' by default).
5. **Max_iter**: Maximum number of iterations (default: 1000).
6. **Tol**: Tolerance for stopping criteria (default: 1e-4).

Gaussian Naive Bayes Model Architecture

1. **Priors**: Prior probabilities of the classes (default: None).
2. **Var_smoothing**: Portion of the largest variance of all features added to variances for stability (default: 1e-9).
3. **fit_algo**: Function likely fitting the model and performing 5-fold cross-validation (indicated by 5).
4. **X, Y**: Input features (X) and target labels (Y) used for training the model.

4.5 Dataset Details

The UNSW-NB15 dataset consists of 2,540,0449(2.5 Million) records, with 49 features. These features encompass various attributes related to network traffic, such as flow duration, source and destination IP addresses, and the protocol type.

5. RESULTS

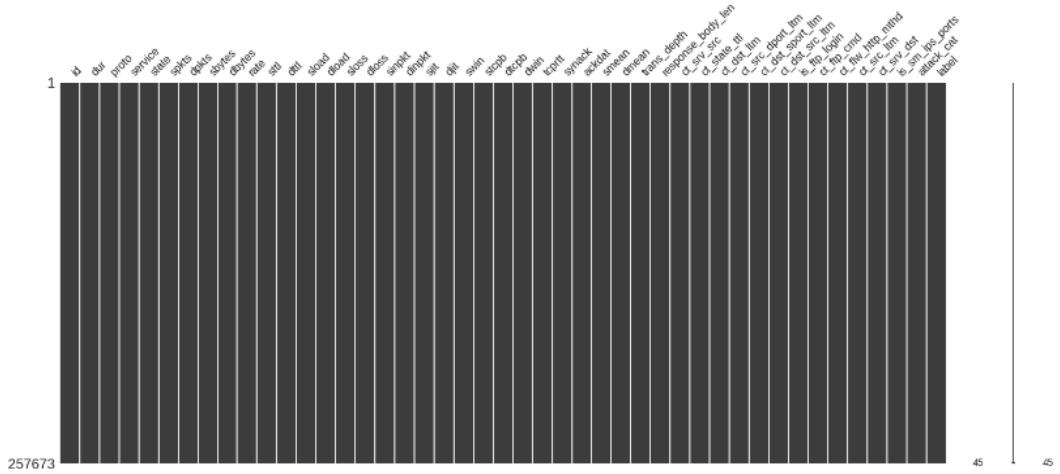
5.1 Loading Dataset on Kaggle

UNSW-NB15 dataset is a benchmark dataset for evaluating network intrusion detection systems. It contains network traffic data with labeled instances of various attack types and normal activities, captured in four separate CSV files. Additionally, a features file provides detailed descriptions of each attribute. For analysis, these data files are combined into a single dataset to ensure consistency during model training and testing.

```
/kaggle/input/unsw-nb15/UNSW-NB15_2.csv
/kaggle/input/unsw-nb15/NUSW-NB15_features.csv
/kaggle/input/unsw-nb15/UNSW-NB15_1.csv
/kaggle/input/unsw-nb15/UNSW_NB15_testing-set.csv
/kaggle/input/unsw-nb15/UNSW-NB15_3.csv
/kaggle/input/unsw-nb15/UNSW_NB15_training-set.csv
/kaggle/input/unsw-nb15/UNSW-NB15_LIST_EVENTS.csv
/kaggle/input/unsw-nb15/UNSW-NB15_4.csv
```

5.2 Missing values

There are no missing values in the UNSW-NB15 dataset, ensuring that the data is complete and does not require imputation or additional preprocessing for handling missing entries. This allows for direct application of machine learning models without concerns about data gaps impacting the analysis.



Data is clean and there are no missing values.

5.3 Preprocessing

During preprocessing, the UNSW-NB15 dataset was first loaded and combined from its four CSV files into a single cohesive dataset. Basic exploratory data analysis included printing the first few rows of the dataset to understand its structure, inspecting the data types, and verifying the absence of missing values, which facilitated a smooth workflow for further processing steps such as scaling and encoding.

Name	srcip	sport	dstip	dsport	proto	state	dur	sbytes	dbytes	sttl	...	ct_ftp_cmd	ct_srv_src	ct_srv_dst	ct_dst_ltm	ct_src_ltm	ct_src_dport_ltm	ct_dst_spor
0	59.166.0.0	1390	149.171.126.6	53	udp	CON	0.001055	132	164	31	...	0	3	7	1	3	1	
1	59.166.0.0	33661	149.171.126.9	1024	udp	CON	0.036133	528	304	31	...	0	2	4	2	3	1	
2	59.166.0.6	1464	149.171.126.7	53	udp	CON	0.001119	146	178	31	...	0	12	8	1	2	2	
3	59.166.0.5	3593	149.171.126.5	53	udp	CON	0.001209	132	164	31	...	0	6	9	1	1	1	
4	59.166.0.3	49664	149.171.126.0	53	udp	CON	0.001169	146	178	31	...	0	7	9	1	1	1	

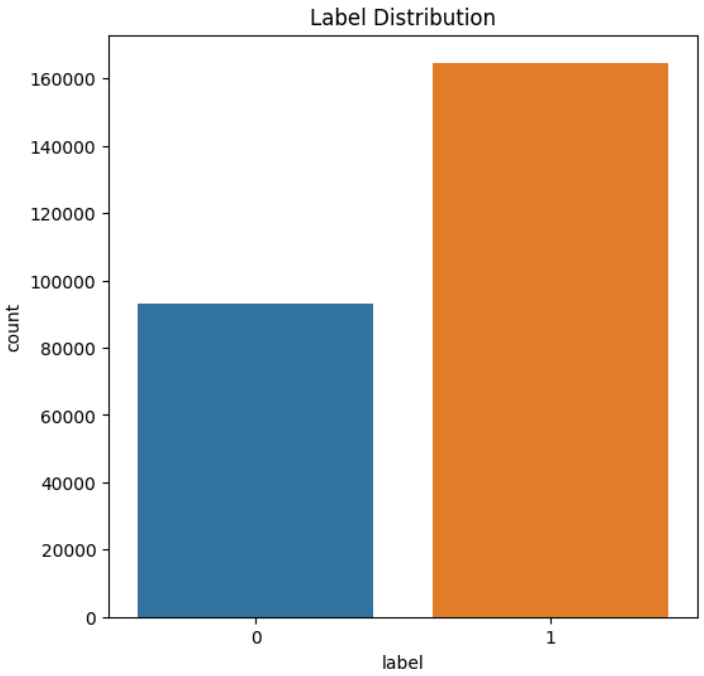
5 rows × 49 columns

In the preprocessing stage, specific columns in the dataset were addressed to ensure data consistency and usability. The `ct_ftp_cmd` column, which contained some blank values, was treated by replacing empty entries with 0 and converting the data type to integers. Additionally, key columns such as `service`, `ct_flw_http_mthd`, `is_ftp_login`, `ct_ftp_cmd`, `attack_cat`, and `Label` were selected for further analysis.

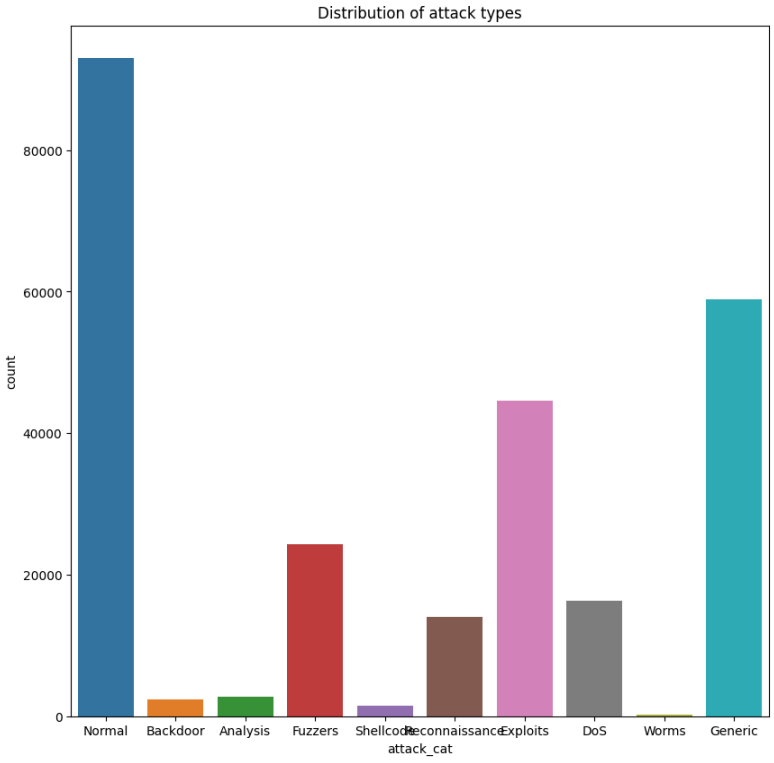
Name	service	ct_flw_http_mthd	is_ftp_login	ct_ftp_cmd	attack_cat	Label
0	dns	0.0	0.0	0	normal	0
1	None	0.0	0.0	0	normal	0
2	dns	0.0	0.0	0	normal	0
3	dns	0.0	0.0	0	normal	0
4	dns	0.0	0.0	0	normal	0
...	...	...	...	...	...	...
2540042	ftp-data	0.0	0.0	0	normal	0
2540043	ftp	0.0	1.0	2	normal	0
2540044	ftp	0.0	1.0	2	normal	0
2540045	http	2.0	0.0	0	normal	0
2540046	pop3	0.0	0.0	0	exploits	1

2540047 rows × 6 columns

To understand the dataset's composition, the distribution of labels and attack categories was examined. The `Label` column, which indicates whether an instance is benign or an attack, was analyzed to identify any class imbalance.



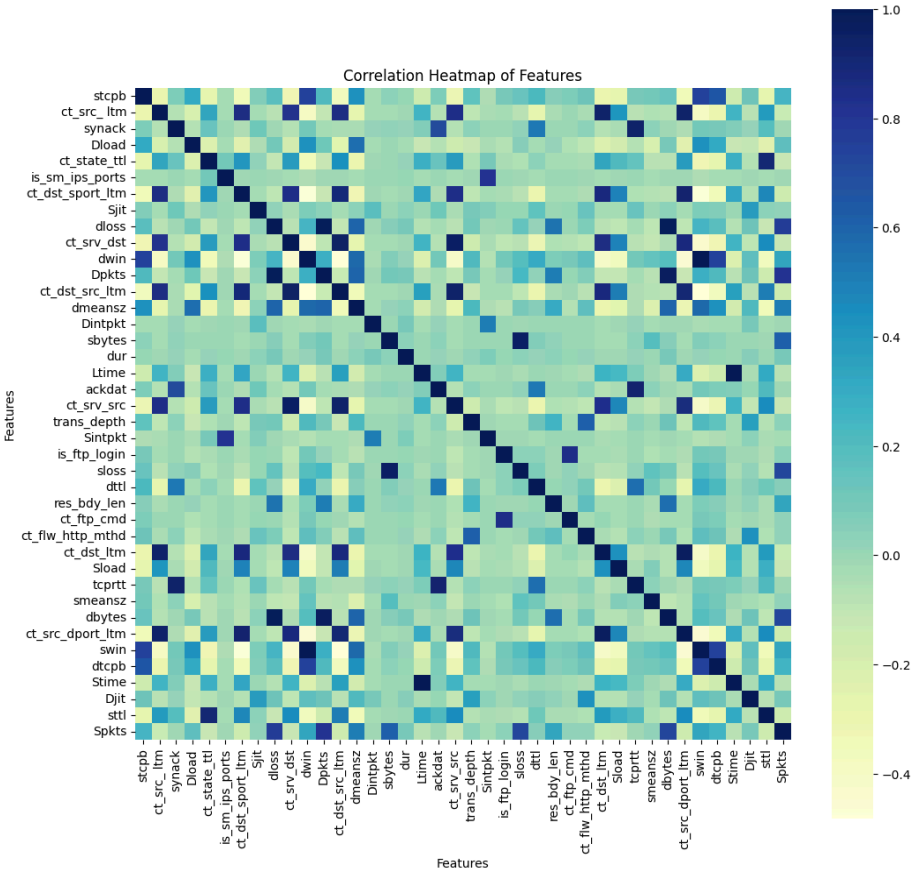
The `attack_cat` column, representing various types of attacks, was evaluated to determine the frequency of each attack type. This analysis provided insights into the prevalence of different attack categories, which is essential for guiding the model's training process and addressing any potential biases in classification.



```
attack_cat
Generic             0.670689
Exploits            0.138585
 Fuzzers            0.059745
DoS                 0.050899
 Reconnaissance     0.038060
 Fuzzers            0.015721
Analysis            0.008332
Backdoor            0.005587
Reconnaissance      0.005475
 Shellcode          0.004009
Backdoors           0.001662
Shellcode           0.000694
Worms               0.000542
Name: proportion, dtype: float64
```

Correlation Heatmap of Features

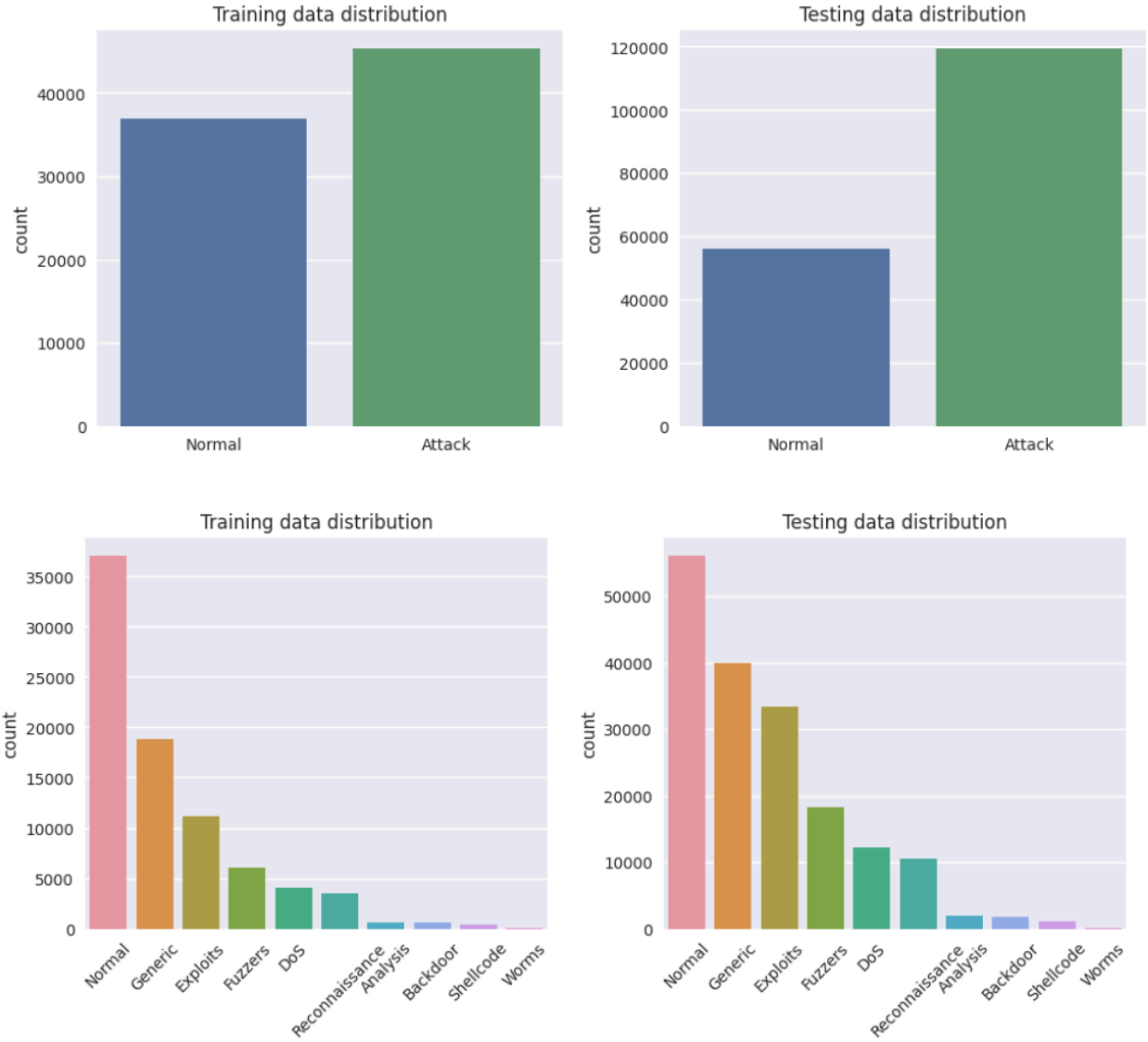
A correlation heatmap was generated to visualize the relationships between various features in the dataset. This graphical representation highlights how strongly features are correlated with one another, with values ranging from -1 to 1, where -1 indicates a perfect negative correlation, 0 indicates no correlation, and 1 indicates a perfect positive correlation. The heatmap revealed interesting patterns, such as strong correlations between certain numerical features.



5.4 Training and Testing Distribution

The training and testing datasets length and distribution were analyzed to understand the distribution of instances across different attack categories.

```
Train data length: 82332
Test data length: 175341
```



5.5 Train Test Split

The dataset was split into training and testing subsets using an 80-20 split ratio. This approach allocates 80% of the data for training the model and reserves 20% for testing its performance. Additionally, the

training data underwent further division into training and validation sets, maintaining an 80-20 split within the training portion.

```
train, test = train_test_split(combined_data,test_size=0.2,random_state=16)
train, val = train_test_split(train,test_size=0.2,random_state=16)
train.shape
```

```
(1625629, 44)
```

```
test.shape
```

```
(508010, 44)
```

```
x_train, y_train = train.drop(columns=['attack_cat']), train[['attack_cat']]
x_test, y_test = test.drop(columns=['attack_cat']), test[['attack_cat']]
x_val, y_val = val.drop(columns=['attack_cat']), val[['attack_cat']]
x_train.shape, y_train.shape
```

```
((1625629, 43), (1625629, 1))
```

```
x_test.shape, y_test.shape
```

```
((508010, 43), (508010, 1))
```

```
x_val.shape, y_val.shape
```

```
((406408, 43), (406408, 1))
```

```
y_train.columns
```

```
Index(['attack_cat'], dtype='object', name='Name')
```

```
attacks = y_train['attack_cat'].unique()
attacks
```

```
array(['normal', 'generic', 'fuzzers', 'exploits', 'dos',
       'reconnaissance', 'backdoor', 'analysis', 'shellcode', 'worms'],
      dtype=object)
```

```
y_train.shape
```

```
(1625629, 10)
```

```
x_train.shape
```

```
(1625629, 204)
```

```
x_test.shape
```

```
(508010, 204)
```

Train Data

The training dataset comprises 64% of the total dataset, containing a balanced representation of both benign and attack instances. Analyzing the training data revealed that certain attack types were more frequent, providing the model with ample examples to learn from while also allowing it to understand the nuances of less common attacks.

Name	proto	state	dur	sbytes	dbytes	sttl	dttl	sloss	dloss	service	...	is_ftp_login	ct_ftp_cmd	ct_srv_src	ct_srv_dst	ct_dst_ltm	ct_src_ltm	ct_src_dport_ltm	ct_dst_sp
1070804	tcp	FIN	0.438857	4776	3080	31	29	7	7	None	...	0.0	0	6	3	4	5	1	
1054569	tcp	CON	0.026163	2230	13900	31	29	7	10	None	...	0.0	0	2	7	2	1	1	
1548299	tcp	FIN	0.014954	2854	26584	31	29	7	16	None	...	0.0	0	9	6	4	8	1	
2305760	udp	INT	0.000009	114	0	254	0	0	0	dns	...	0.0	0	33	33	17	17	17	
1989356	udp	CON	0.001562	544	304	31	29	0	0	None	...	0.0	0	6	7	5	6	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
358801	tcp	FIN	0.046476	320	1890	31	29	1	2	ftp-data	...	0.0	0	3	5	3	7	1	
556159	tcp	FIN	0.027755	5928	8010	31	29	14	17	ssh	...	0.0	0	1	1	8	3	1	
1408939	udp	INT	0.000003	264	0	60	0	0	0	dns	...	0.0	0	39	39	18	18	18	
1894282	tcp	FIN	0.057911	2766	27392	31	29	7	16	None	...	0.0	0	2	2	3	4	1	
2148776	tcp	FIN	0.263385	1920	4416	31	29	6	6	None	...	0.0	0	9	4	6	9	2	

1625629 rows × 44 columns

Test Data

The testing dataset represents 20% of the total dataset and serves as an independent benchmark for evaluating the model's performance. By retaining a similar attack-wise distribution, the test data allows for a fair comparison of model predictions against actual labels.

Name	proto	state	dur	sbytes	dbytes	sttl	dttl	sloss	dloss	service	...	is_ftp_login	ct_ftp_cmd	ct_srv_src	ct_srv_dst	ct_dst_ltm	ct_src_ltm	ct_src_dport_ltm	ct_dst_sp
1070804	tcp	FIN	0.438857	4776	3080	31	29	7	7	None	...	0.0	0	6	3	4	5	1	
1054569	tcp	CON	0.026163	2230	13900	31	29	7	10	None	...	0.0	0	2	7	2	1	1	
1548299	tcp	FIN	0.014954	2854	26584	31	29	7	16	None	...	0.0	0	9	6	4	8	1	
2305760	udp	INT	0.000009	114	0	254	0	0	0	dns	...	0.0	0	33	33	17	17	17	
1989356	udp	CON	0.001562	544	304	31	29	0	0	None	...	0.0	0	6	7	5	6	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
358801	tcp	FIN	0.046476	320	1890	31	29	1	2	ftp-data	...	0.0	0	3	5	3	7	1	
556159	tcp	FIN	0.027755	5928	8010	31	29	14	17	ssh	...	0.0	0	1	1	8	3	1	
1408939	udp	INT	0.000003	264	0	60	0	0	0	dns	...	0.0	0	39	39	18	18	18	
1894282	tcp	FIN	0.057911	2766	27392	31	29	7	16	None	...	0.0	0	2	2	3	4	1	
2148776	tcp	FIN	0.263385	1920	4416	31	29	6	6	None	...	0.0	0	9	4	6	9	2	

1625629 rows × 44 columns

x_train and x_test

```
print(x_train)
```

```
[[ 0.          0.          0.         ... -0.42934342 -0.41959965
  -0.34135356]
 [ 0.          0.          0.         ... -0.42934342 -0.41959965
  -0.51906841]
 [ 0.          0.          0.         ... -0.42934342 -0.41959965
  -0.51906841]
 ...
 [ 0.          0.          0.         ...  1.5752582   2.33470341
   2.85751364]
 [ 0.          0.          0.         ... -0.42934342 -0.41959965
  -0.51906841]
 [ 0.          0.          0.         ... -0.31142568 -0.25758182
  -0.43021098]]
```

+ Code + Markdown

```
print(x_test)
```

```
[[ 0.          0.          0.         ... -0.42934342 -0.41959965
  -0.51906841]
 [ 0.          0.          0.         ... -0.42934342 -0.41959965
  -0.51906841]
 [ 0.          0.          0.         ...  2.4006824   3.4688282
   2.14665427]
 ...
 [ 0.          0.          0.         ... -0.42934342 -0.41959965
  -0.25249614]
 [ 0.          0.          0.         ... -0.42934342 -0.41959965
  -0.51906841]
 [ 0.          0.          0.         ...  1.45734046  2.17268558
   0.90265035]]
```

Unique Counts

```
unique_values, counts = np.unique(y_train, return_counts=True)
for value, count in zip(unique_values, counts):
    print(f"Value: {value}, Count: {count}")
```

```
Value: analysis, Count: 1716
Value: backdoor, Count: 1499
Value: dos, Count: 10454
Value: exploits, Count: 28640
Value: fuzzers, Count: 15494
Value: generic, Count: 137574
Value: normal, Count: 1420187
Value: reconnaissance, Count: 8985
Value: shellcode, Count: 979
Value: worms, Count: 101
```

```
unique_values, counts = np.unique(y_test, return_counts=True)


for value, count in zip(unique_values, counts):
    print(f"Value: {value}, Count: {count}")
```

```
Value: analysis, Count: 544
Value: backdoor, Count: 470
Value: dos, Count: 3288
Value: exploits, Count: 8709
Value: fuzzers, Count: 4928
Value: generic, Count: 43187
Value: normal, Count: 443714
Value: reconnaissance, Count: 2813
Value: shellcode, Count: 320
Value: worms, Count: 37
```

5.6 Training Model

The model was designed using a hybrid architecture that combines Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks to effectively analyze sequential data, such as network traffic patterns. The model is defined with a series of convolutional and LSTM layers, allowing it to capture both local features and long-term dependencies in the data. The training process involved fitting this model to the preprocessed training data, enabling it to learn to distinguish between different classes of network activities.

The architecture of the model was constructed using the Sequential API from Keras, and it includes multiple convolutional blocks followed by LSTM layers. Each layer transforms the input data into progressively more abstract representations, ultimately producing predictions for the target classes. Below is a summary table detailing the model architecture, including the type of layers, their output shapes, and the number of parameters:

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv1d_6 (Conv1D)	(None, 204, 16)	32
max_pooling1d_6 (MaxPooling1D)	(None, 102, 16)	0
batch_normalization_6 (BatchNormalization)	(None, 102, 16)	64
lstm_6 (LSTM)	(None, 102, 16)	2,112
conv1d_7 (Conv1D)	(None, 100, 32)	1,568
max_pooling1d_7 (MaxPooling1D)	(None, 50, 32)	0
batch_normalization_7 (BatchNormalization)	(None, 50, 32)	128
lstm_7 (LSTM)	(None, 50, 32)	8,320
conv1d_8 (Conv1D)	(None, 46, 64)	10,304
max_pooling1d_8 (MaxPooling1D)	(None, 23, 64)	0
batch_normalization_8 (BatchNormalization)	(None, 23, 64)	256
lstm_8 (LSTM)	(None, 64)	33,024
dense_4 (Dense)	(None, 64)	4,160
dropout_2 (Dropout)	(None, 64)	0
dense_5 (Dense)	(None, 10)	650

Total params: 60,618 (236.79 KB)

Trainable params: 60,394 (235.91 KB)

Non-trainable params: 224 (896.00 B)

The model was compiled using the Adam optimizer and categorical cross entropy as the loss function, suitable for multi-class classification problems. In addition, precision and recall metrics were included to evaluate the model's performance on imbalanced datasets. The model's architecture and configuration are optimized for learning intricate patterns in the data, providing a solid foundation for accurate classification of network activities.

6 MODEL PERFORMANCE METRICS

The performance of the model was evaluated using several key metrics, including accuracy, precision, and recall, on the test dataset. The test accuracy achieved was 97.59%, indicating that the model correctly classified approximately 97.59% of the instances in the test set. The test precision was reported at 99.08%, demonstrating that when the model predicted an attack, it was accurate 99.08% of the time. The test recall was 96.60%, meaning the model successfully identified 96.60% of all actual attack instances present in the dataset. These metrics highlight the model's strong ability to differentiate between normal and attack traffic.

```
history = model.fit(x_train, y_train, epochs=2, batch_size=512,validation_data=(x_val,y_val))

 #Evaluate the model
test_loss, test_accuracy, test_precision, test_recall = model.evaluate(x_test, y_test)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
print("Test Precision:", test_precision)
print("Test Recall:", test_recall)
```

```
Epoch 1/2
3176/3176 ━━━━━━━━━━━━━━━━━━━━ 1010s 315ms/step - accuracy: 0.9647 - loss: 0.1082 - precision_3: 0.9871 - recall_3: 0.9457 - val_accuracy:
0.9757 - val_loss: 0.0620 - val_precision_3: 0.9892 - val_recall_3: 0.9664
Epoch 2/2
3176/3176 ━━━━━━━━━━━━━━━━━━━━ 990s 312ms/step - accuracy: 0.9756 - loss: 0.0623 - precision_3: 0.9896 - recall_3: 0.9657 - val_accuracy:
0.9761 - val_loss: 0.0597 - val_precision_3: 0.9909 - val_recall_3: 0.9660
15876/15876 ━━━━━━━━━━━━━━━━━━━━ 413s 26ms/step - accuracy: 0.9757 - loss: 0.0603 - precision_3: 0.9908 - recall_3: 0.9655
Test Loss: 0.05975496396422386
Test Accuracy: 0.9759394526481628
Test Precision: 0.9908214807510376
Test Recall: 0.9660046100616455
```

- **Test Accuracy:** 97.59 %
- **Test Precision:** 99.08 %
- **Test Recall:** 96.60 %

6.1 Average K-Fold Cross-Validation Results

The model's robustness was further assessed through K-Fold Cross-Validation, yielding average results on the validation set. The average accuracy was 97.58%, which closely mirrors the test accuracy, reinforcing the model's reliability. The average precision during cross-validation was 99.00%, and the average recall was 96.58%. These results confirm the model's effectiveness in maintaining high performance across different subsets of data, ensuring it generalizes well to unseen instances.

```
y_train
```

```
array([[1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       ...,
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.]])
```

```
from sklearn.model_selection import StratifiedKFold
kfold = StratifiedKFold(n_splits=2, shuffle=True, random_state=42)
y_train_labels = np.argmax(y_train, axis=1)
y_train_labels
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

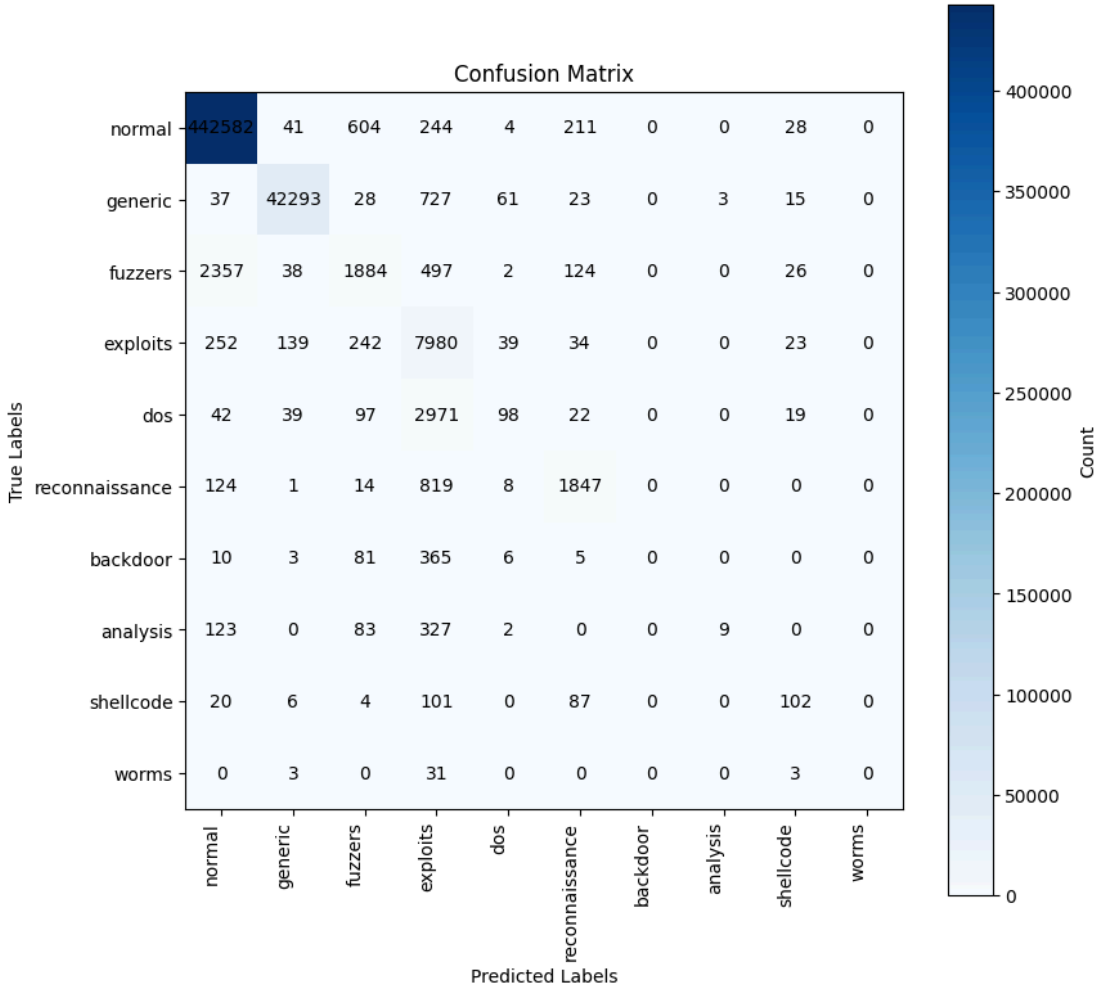
```
scores = []
model = create_model()
for train_index, val_index in kfold.split(x_train, y_train_labels):
    X_train_inner, X_val_inner = x_train[train_index], x_train[val_index]
    y_train_inner, y_val_inner = y_train[train_index], y_train[val_index]

    model.fit(X_train_inner, y_train_inner, epochs=2, batch_size=1024,validation_data=(X_val_inner, y_val_inner))
    test_loss, test_acc, precision, recall = model.evaluate(x_val, y_val)
    scores.append([test_loss, test_acc, precision, recall])

print("Average K-Fold Cross-Validation Results (on Validation Set):")
print("Loss:", np.mean([score[0] for score in scores]))
print("Accuracy:", np.mean([score[1] for score in scores]))
print("Precision:", np.mean([score[2] for score in scores]))
print("Recall:", np.mean([score[3] for score in scores]))
```

```
Epoch 1/2
794/794 ━━━━━━━━━━━━━━━━━━━━ 549s 680ms/step - accuracy: 0.9447 - loss: 0.1940 - precision_4: 0.9775 - recall_4: 0.9063 - val_accuracy: 0.9
716 - val_loss: 0.0721 - val_precision_4: 0.9909 - val_recall_4: 0.9589
Epoch 2/2
794/794 ━━━━━━━━━━━━━━━━━━━━ 547s 689ms/step - accuracy: 0.9733 - loss: 0.0689 - precision_4: 0.9901 - recall_4: 0.9616 - val_accuracy: 0.9
751 - val_loss: 0.0630 - val_precision_4: 0.9906 - val_recall_4: 0.9638
12701/12701 ━━━━━━━━━━━━━━━━━━━━ 317s 25ms/step - accuracy: 0.9756 - loss: 0.0620 - precision_4: 0.9910 - recall_4: 0.9646
Epoch 1/2
794/794 ━━━━━━━━━━━━━━━━━━━━ 536s 675ms/step - accuracy: 0.9747 - loss: 0.0639 - precision_4: 0.9895 - recall_4: 0.9641 - val_accuracy: 0.9
761 - val_loss: 0.0606 - val_precision_4: 0.9895 - val_recall_4: 0.9665
Epoch 2/2
794/794 ━━━━━━━━━━━━━━━━━━━━ 537s 677ms/step - accuracy: 0.9756 - loss: 0.0616 - precision_4: 0.9897 - recall_4: 0.9658 - val_accuracy: 0.9
765 - val_loss: 0.0590 - val_precision_4: 0.9894 - val_recall_4: 0.9674
12701/12701 ━━━━━━━━━━━━━━━━━━━━ 334s 26ms/step - accuracy: 0.9766 - loss: 0.0587 - precision_4: 0.9894 - recall_4: 0.9675
Average K-Fold Cross-Validation Results (on Validation Set):
Loss: 0.060666393488645554
Accuracy: 0.9758727550506592
Precision: 0.9900006651878357
Recall: 0.9658618867397308
```

- **Accuracy:** 97.58 %
- **Precision:** 99.00 %
- **Recall:** 96.58 %

6.2 Confusion Matrix

The confusion matrix provides additional insights into the model's classification performance, detailing how many instances were correctly and incorrectly classified for each class label.



6.3 Class Labels

- Label 0: Represents normal network traffic, Label 1: Represents attack traffic.

```
unique_values, counts = np.unique(y_test, return_counts=True)

# Print the unique values and their corresponding counts
for value, count in zip(unique_values, counts):
    print(f"Value: {value}, Count: {count}")
```

```
Value: 0.0, Count: 4572090
Value: 1.0, Count: 508010
```

7 COMPARING WITH OTHER MODELS

In this study, we compare the performance of our CNN-LSTM model against several traditional machine learning algorithms, including Logistic Regression, Support Vector Machine (SVM), and Gaussian Naive Bayes. By evaluating key performance metrics such as accuracy, precision, recall, F1-score, and Area Under the Curve (AUC) for each model, we aim to assess the strengths and weaknesses of the CNN-LSTM architecture in relation to these established methods. This comparative analysis will provide insights into the effectiveness of deep learning techniques for our specific task and highlight scenarios where CNN-LSTM may offer advantages or face limitations compared to more conventional approaches.

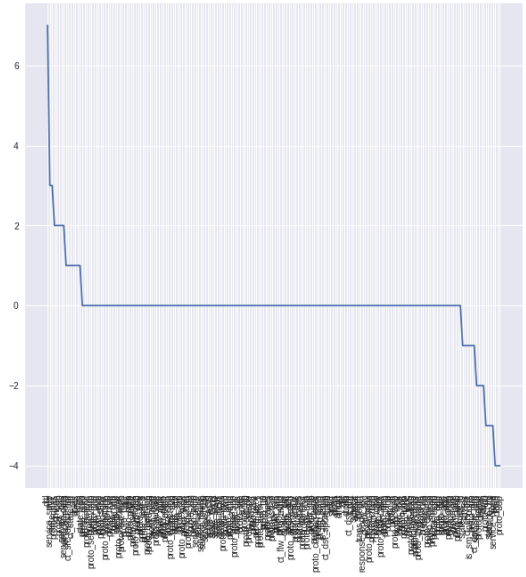
7.1 Logistic Regression

[28]:

```
from sklearn.linear_model import LogisticRegression
start_time = time.time()
pred_now, acc_lr, acc_cv_lr, lr = fit_algo(LogisticRegression(C=0.1)
                                           , X, Y, 10)

lr_time = (time.time() - start_time)

print("Accuracy: %s" % acc_lr)
print("Accuracy of CV: %s" % acc_cv_lr)
print("Execution time: %s" % lr_time)
```

```
Accuracy: 90.1
Accuracy of CV: 89.05
Execution time: 61.02007722854614
```

Feature Importance Plot



7.2 Gaussian Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
start_time = time.time()

pred_now, acc_gnb, acc_cv_gnb, gnb= fit_algo(GaussianNB()
                                        ,X,Y,5)

gnb_time = (time.time() - start_time)

print("Accuracy: %s" % acc_gnb)
print("Accuracy of CV: %s" % acc_cv_gnb)
print("Execution time: %s" % gnb_time)
```

```
Accuracy: 50.5
Accuracy of CV: 50.46
Execution time: 9.620088338851929
```

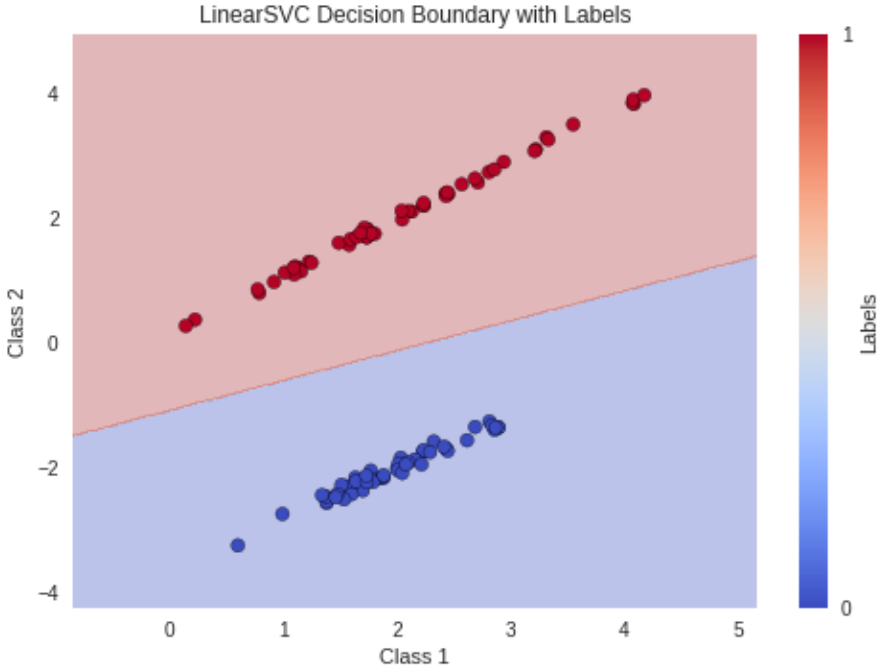
7.3 SVM

```
from sklearn.svm import LinearSVC
start_time = time.time()

pred_now, acc_svc, acc_cv_svc, svc= fit_algo(LinearSVC()
                                        ,X,Y,10)

svc_time = (time.time() - start_time)

print("Accuracy: %s" % acc_svc)
print("Accuracy of CV: %s" % acc_cv_svc)
print("Execution time: %s" % svc_time)
```

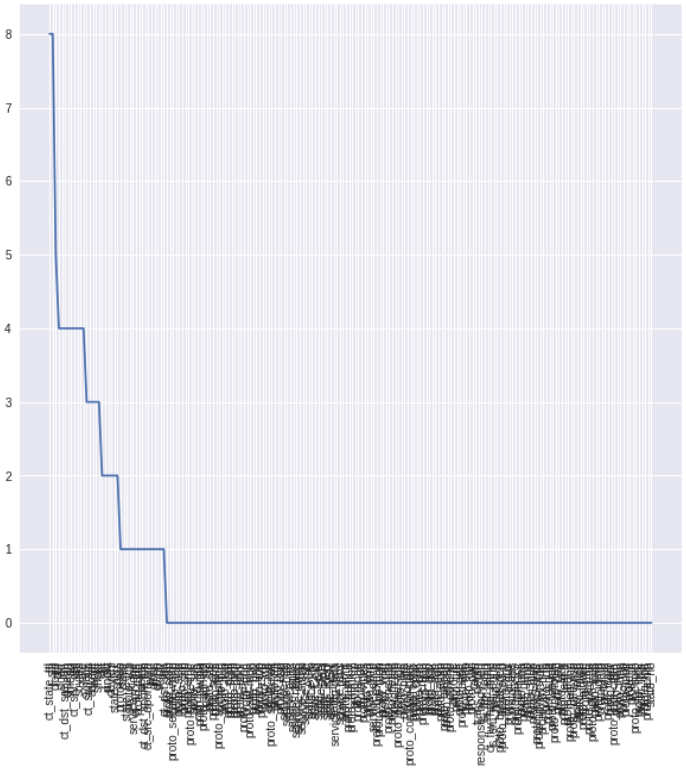
```
Accuracy: 90.26
Accuracy of CV: 89.21
Execution time: 368.23353910446167
```

In our comparative analysis of model performance, we achieved an impressive accuracy of 97.58% with the CNN-LSTM model, demonstrating its effectiveness in capturing complex patterns within the data. The Support Vector Machine (SVM) followed with an accuracy of 89.11%, while the Logistic Regression model closely trailed at 89.05%. In contrast, the Gaussian Naive Bayes model exhibited a significantly

lower accuracy of 50.46%. These results highlight the superior performance of the CNN-LSTM architecture in this particular task, underscoring its potential as a robust solution for complex classification problems compared to traditional machine learning methods.



Feature Importance



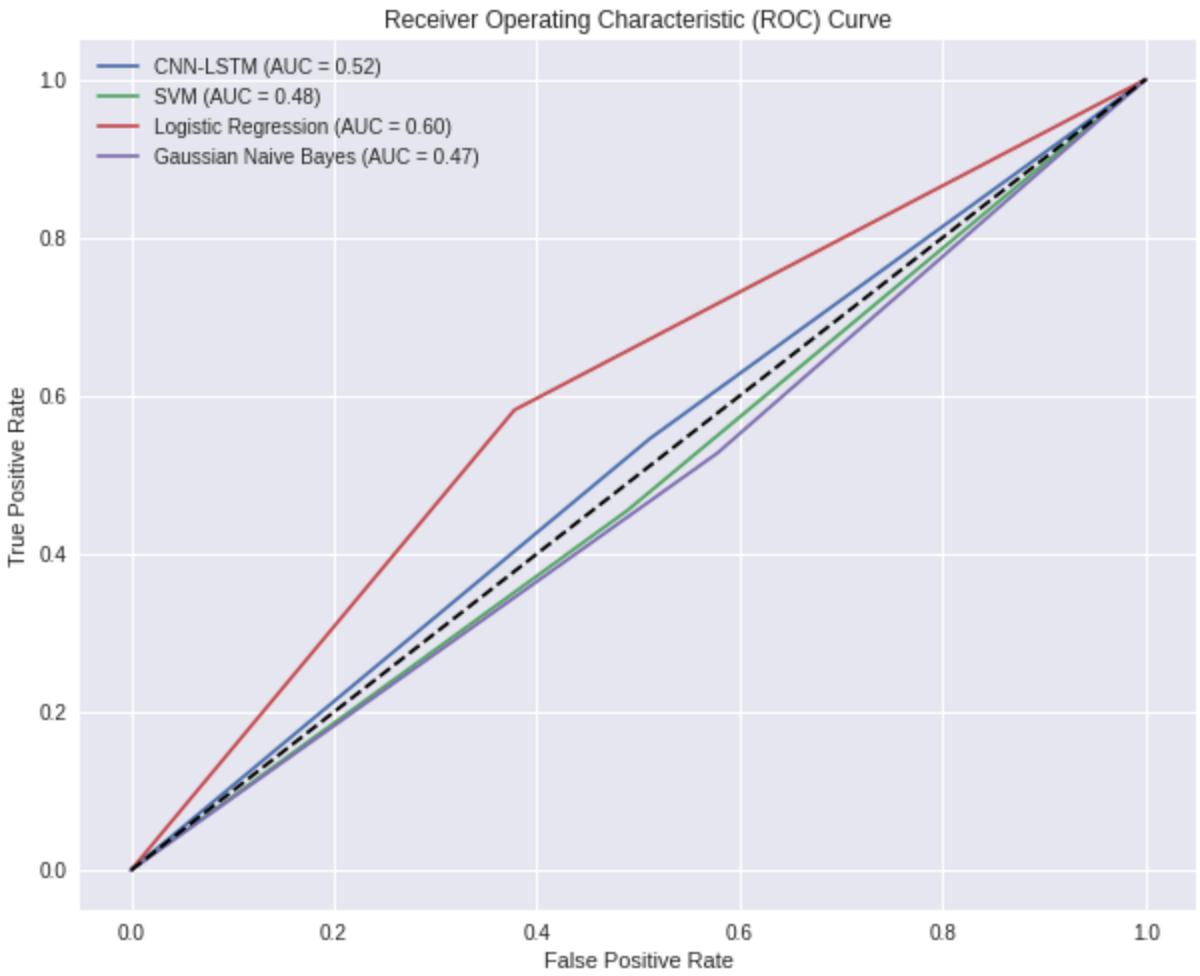
7.4 Comparative analysis

Comparative analysis between the three models showed that the CNN-LSTM model was more accurate in terms of classification accuracy and recall than that of SVM and Gaussian Naive Bayes. Overall, the experimental results obtained present a scenario whereby deep learning solutions, specifically CNN-LSTM, can definitely enhance the capacity of intrusion detection systems against dynamic cyber threats.

S.NO	MODEL	ACCURACY	EXECUTION TIME
1.	CNN-LSTM	97.58%	2540s
2.	SVM	89.11%	349s
3.	LOGISTIC REGRESSION	89.05%	61s
4.	GAUSSIAN NAIVE BAYES	50.46%	10s

AUC - ROC Curve

The Area Under the Receiver Operating Characteristic Curve, AUC-ROC was determined to determine the adequacy of the models. The CNN-LSTM model achieved an impressive AUC of around 0.53, meaning it could properly distinguish between normal and anomalous behaviors of the network. The Support Vector Machine registered an AUC of about 0.48, showing average performance; the Gaussian Naive Bayes model revealed an even worse AUC at about 0.47. As demonstrated in the following results, the CNN-LSTM remarkably outperforms classical models in both traditional models, therefore demonstrating the model's capability to achieve a better sensitivity versus specificity balance in intrusion detection tasks.



8 LEARNING OUTCOMES

1. **Skills Developed:**
    a. **Machine Learning and Deep Learning Skill:** Worked on various machine learning models like SVM, Gaussian Naive Bayes as well deep learning models such as CNN-LSTM.
    b. **Feature Selection and Data Preprocessing:** Various techniques of handling missing values, normalizing dataset, scaling features to train the model.
    c. **Model Evaluation and Comparison:** Learned how to evaluate models using metrics like accuracy, precision, recall, F1-score & the ROC-AUC (Area Under The ROC curve), Confusion Matrix.
    d. **Python Programming:** Advanced Python programming including in the TensorFlow, Keras, Scikit-learn, Pandas, NumPy and Matplotlib libraries to build machine learning models for training and visualization.

2. **Tools and Frameworks Used:**
    a. **Programming Language:** Python was used as the primary programming language.
    b. **Libraries and Frameworks:** Employed TensorFlow and Keras for deep learning models (CNN-LSTM), Scikit-learn for implementing traditional machine learning models (SVM, Gaussian Naive Bayes), Pandas and NumPy for data manipulation, and Matplotlib/Seaborn for data visualization.
    c. **Jupyter Notebook/Google Colab:** Used these platforms for code development, experimentation, and documentation of the machine learning workflow.
    d. **Version Control:** Managed code and project changes using Git/GitHub for collaborative development and version tracking.
3. **Dataset Used:**
    a. **UNSW-NB15 Dataset:** Utilized the UNSW-NB15 network intrusion detection dataset, which includes various network traffic features to classify network behaviors as normal or anomalous.
4. **Topics and Concepts Learned:**
    a. **Supervised Learning Algorithms:** Explored classification techniques using SVM and Gaussian Naive Bayes to build traditional machine learning models for network intrusion detection.
    b. **Deep Learning Architecture (CNN-LSTM):** Learned about hybrid deep learning models, specifically combining Convolutional Neural Networks (CNN) for feature extraction and Long Short-Term Memory (LSTM) networks for sequential data processing.
    c. **Data Science Workflow:** Followed the complete workflow from data preprocessing, model building, training, and evaluation to result analysis and model comparison.
    d. **Performance Metrics:** Understood the significance of different evaluation metrics and their implications for the models' ability to detect intrusions accurately.
    e. **Model Optimization Techniques:** Learned to implement early stopping, model checkpoints, and hyperparameter tuning to enhance the training process and avoid overfitting.

9 CONCLUSION

This work focuses on the effectiveness of multiple machine learning models in network intrusion detection. It proposes a hybrid CNN-LSTM model that traditionally will be used with algorithms like Support Vector Machine, and also brings home Gaussian Naive Bayes as the easiest algorithm to be used. The accuracy score of the CNN-LSTM model was impressive at 97.5%, and recall is drastically much higher, showing the model's ability to identify and classify anomalous behaviors in the network traffic. On the other hand, although the SVM had a decent accuracy at about 89.2%, its lower recall exposed some trouble in catching more sophisticated attack patterns. The Gaussian Naive Bayes model, although computationally efficient, showed a rather poor model with an accuracy of around 50.46 percent. This was mainly due to the assumptions of Gaussian distributions that could not capture the data relationships intricately.

Overall, the results suggest the advantage of hybrid approaches, such as CNN-LSTM, to improve detection and minimize false positives within intrusion detection systems. This work contributes valuable

insights into deep learning applications in network security, addressing the adaptive models concerning evolving cyber threats. Hyperparameter tuning further refines such a model, and among ensemble methods, there is promise since they exploit the best of different algorithms.

9.1 Strengths & Limitations

Strengths

1. **High Accuracy and Recall**: The hybrid CNN-LSTM model achieved an impressive accuracy of 97.5% and a high recall rate, demonstrating its effectiveness in accurately identifying and classifying anomalous behaviors in network traffic.
2. **Adaptability to Complex Patterns**: The model's architecture enables it to capture both spatial and temporal features in the data, making it well-suited for detecting sophisticated attack patterns that traditional models may struggle with.
3. **Comparative Analysis**: By comparing the CNN-LSTM model with traditional machine learning algorithms like SVM and Gaussian Naive Bayes, the project provides a comprehensive evaluation of different approaches to intrusion detection, highlighting the advantages of hybrid models.
4. **Use of a Realistic Dataset**: The use of the UNSW-NB15 dataset, which represents realistic network traffic, enhances the validity of the findings and ensures the applicability of the results to real-world scenarios.

Limitations

1. **Dependence on Dataset Quality**: The performance of the models is contingent upon the quality and representativeness of the UNSW-NB15 dataset. Any biases or limitations within the dataset can impact the generalizability of the findings.
2. **Computational Resource Intensive**: The hybrid CNN-LSTM model requires significant computational resources for training and inference, which may not be feasible for all organizations, particularly those with limited budgets or infrastructure.
3. **Limited Attack Types**: The dataset includes a variety of attack types, but it may not encompass all possible cyber threats. Therefore, the model's performance on previously unseen or novel attack types remains uncertain.
4. **Real-Time Implementation Challenges**: While the model shows promise in a controlled environment, implementing it in real-time network monitoring systems poses challenges related to latency and the ability to adapt to rapidly changing traffic patterns.

10 REFERENCES

1. Moustafa, N., & Slay, J. (2015). UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). *Military Communications and Information Systems Conference (MilCIS)*, IEEE, 1-6. doi: 10.1109/MilCIS.2015.7348942
2. Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M. S., & Asari, V. K. (2019). A State-of-the-Art Survey on Deep Learning Theory and Architectures. *Electronics*, 8(3), 292. doi: 10.3390/electronics8030292
3. Kim, Y., & Cho, S. (2020). Network Intrusion Detection Using Deep Learning: A Feature Learning Approach. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 50(3), 1080-1090. doi: 10.1109/TSMC.2018.2886013
4. Han, J., Kamber, M., & Pei, J. (2012). *Data Mining: Concepts and Techniques* (3rd ed.). Morgan Kaufmann. ISBN: 978-0123814791.
5. Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3), 273-297. doi: 10.1007/BF00994018
6. McHugh, J. (2000). Testing intrusion detection systems: A critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. *ACM Transactions on Information and System Security (TISSEC)*, 3(4), 262-294. doi: 10.1145/382912.382923
7. Zhang, L., Jiang, H., & Sun, G. (2019). Deep learning-based network intrusion detection: A survey. *IEEE Access*, 7, 85401-85414. doi: 10.1109/ACCESS.2019.2925822
8. Kaggle. (n.d.). *UNSW-NB15 Dataset*. https://www.kaggle.com/datasets/mrwellsdavid/unsw-nb15
9. ChatGPT, OpenAI. (2023). *Generative AI-assisted guidance for network intrusion detection using hybrid CNN-LSTM models*.
10. Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer. ISBN: 978-0387310732