W207 Final Project - Detecting Cyber Threat Events with Bi-directional LSTM CNN Model

Sean Sica

04/05/2024

Project Overview

The project, titled "Detecting Cyber Threat Events with Bi-directional LSTM CNN Model," aims to address the critical challenge of rapidly evolving DDoS attacks that are becoming increasingly sophisticated and difficult to detect with traditional security measures. By utilizing advanced machine learning techniques, this initiative seeks to develop an adaptive system capable of identifying and mitigating DDoS threats in real-time network traffic. This approach not only aims to enhance the accuracy of DDoS attack detection but also significantly reduces the incidence of false positives and negatives, thereby improving the security posture of networked systems against such disruptive cyber threats.

Team Members

- Sean Sica

Problem Statement

In today's digital landscape, Distributed Denial of Service (DDoS) attacks represent a formidable challenge for network security, characterized by their ability to evolve rapidly and elude traditional defense systems. These attacks disrupt service availability, causing significant downtime and financial repercussions for affected businesses. Traditional defense mechanisms are often inadequate in effectively handling the complexity and variability of contemporary DDoS tactics. As these attacks continue to grow in sophistication, there is a pressing need for more advanced and dynamic methods of detection and mitigation.

Objective

The primary goal of this project is to develop a state-of-the-art machine learning model that excels in detecting DDoS attacks within network traffic. This model aims to achieve high accuracy with a strategic focus on minimizing both false positives (erroneous threat detection) and false negatives (missed threats). By surpassing the capabilities of traditional detection approaches, the model will provide a more reliable and adaptive solution for combatting the evolving landscape of DDoS threats. This endeavor not only aims to protect network infrastructures but also seeks to reduce the operational and financial impacts of these disruptive cyber incidents.

Approach/Methodology

Exploratory Data Analysis

Initial exploratory data analysis (EDA) was conducted in a Jupyter Notebook environment (Google Colab). This involved studying the CIC-IDS2017 dataset and determining a feasible approach for feature engineering. A variance distribution was computed and plotted to aid with identifying any obvious signs of patterns or discrepancies in the features.

Null Values Each column was checked for the presence of null values. The only column containing null values is "Flow Byte(s)", totaling 1358 instances. The "Flow Bytes(s)" feature represents the rate at which bytes are transmitted over a network during a specific flow. A flow is as a sequence of packets sent from a source to a destination, characterized by attributes like IP addresses, port numbers, and protocol.

Before deciding on how to handle the null values, additional investigation was done in the context of other features. For example, checking the "Flow Duration" might offer insights: flows with very short durations might explain the nulls.

It was discovered that all samples with null "Flow Byte(s)" have a flow duration of 0, suggesting that these flows are instantaneous and do not last any measurable amount of time. In the context of network traffic, a flow duration of 0 could mean that the flow was initiated and terminated within the same timestamp unit for measurement, thus not allowing for a rate (flow bytes) to be calculated.

The total_fwd_packets and total_backward_packets statistics show that, on average, there are about 1-2 forward packets and very few (if any) backward packets. This suggests minimal to no response or acknowledgment packets in these flows, which aligns with the instantaneous nature of these flows.

Given this context, the null flow_bytes/s values *appear* to be a natural consequence of the flow_duration being 0, rather than missing data due to recording errors. Therefore, simply imputing these nulls with a mean or median could introduce bias or inaccuracies. Since these flows are instantaneous and have no duration, it might make sense to set flow_bytes/s for these records to 0. This directly reflects the fact that, over the zero duration, no bytes were transmitted per second.

The research paper accompanying the CIC-IDS2017 dataset was reviewed before making a final determination. The paper states that Flow Inter arrival time (IAT) related features such as Min, Mean, Max and also the Flow Duration are the best common features for DoS detection. Notably, for DDoS attack, backward packet length, average packet size and some inter arrival time related features are heavily influential features.

Here are some observations and potential insights based on the boxplot:

- Variability in Duration: There is significant variability in the duration of flows across different categories. Some types of attacks, like DoS and DDoS, have wider interquartile ranges (IQR), indicating more variability within these categories.
- Long Duration Attacks: The categories like DoS Hulk and Heartbleed show a larger range of flow durations, including some extremely high values, which could indicate prolonged attack activities.
- Short Duration Traffic: On the other hand, categories such as BENIGN, Infiltration, and various Web Attacks have shorter flow durations, as indicated by the lower median and IQR, which might be typical of regular traffic or more stealthy attacks.
- Outliers: There are numerous outliers in almost all categories, especially in DoS Hulk and DDoS. This could indicate the presence of exceptionally long flows that could be due to either malicious activities or non-malicious but unusual network behavior.
- Median and Mean Duration: Categories with higher medians like DoS GoldenEye might imply a consistently higher flow duration characteristic of the attack, while others with lower medians but high variability (like DDoS) might be sporadic or bursty.

Future research on this dataset should consider the potential for feature engineering the flow duration, especially when considered in conjunction with other features like packet size and inter-arrival times. It could be used to engineer additional features that more effectively discriminate between benign and malicious traffic.

Literature Review

A literature review was performed, which included reading the supporting paper published alongside the CIC-IDS2017 dataset. The researchers listed what they determined to be the best selected features and corresponding weight of each label category. Although they did not explicitly explain how they calculated the weights in Table 3 of the paper, they mentioned the use of `RandomForestRegressor`, suggesting that they likely used a feature importance technique.

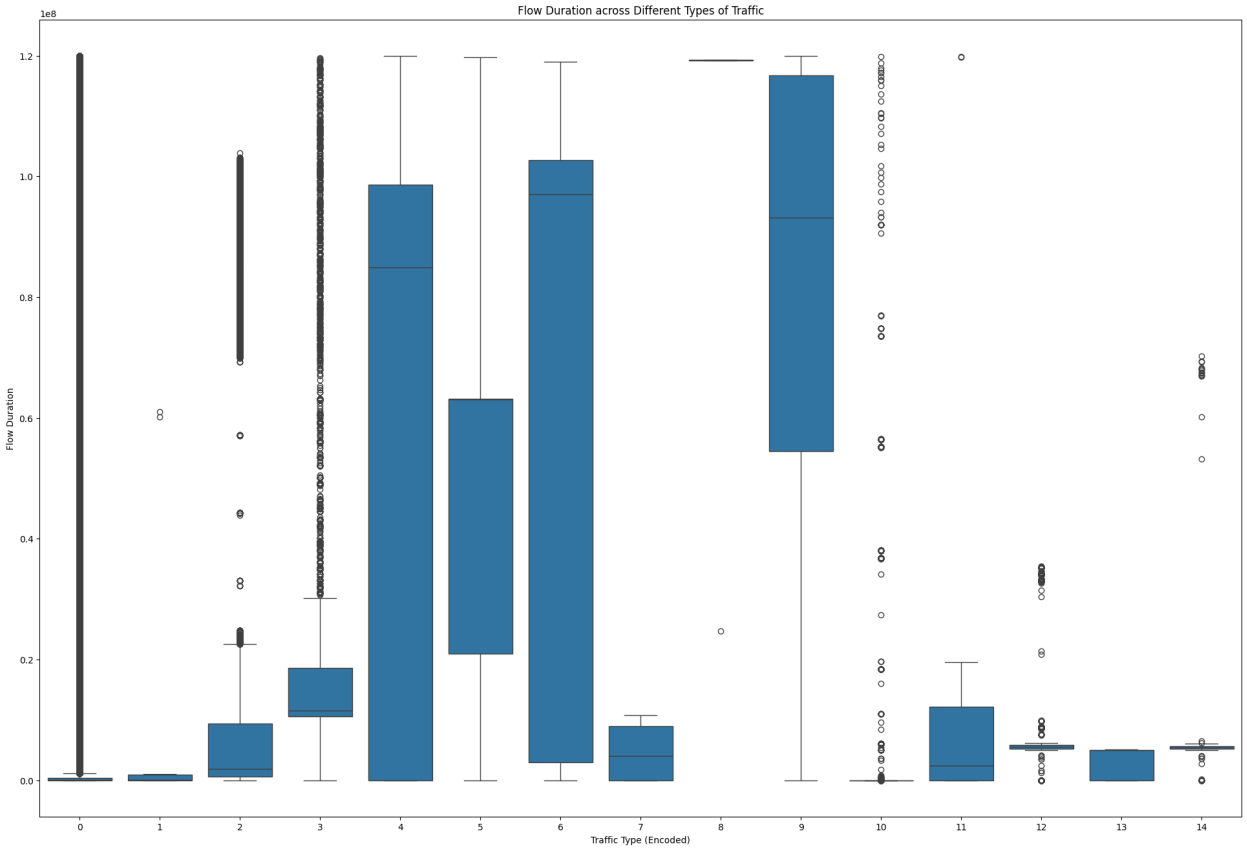


Figure 1: Flow Durations

An attempt was made to recreate their findings using a Random Forest Regression analysis with SciKit-Learn. While the exact results could not be replicated, likely due to differences in preprocessing steps and hyperparameters, the identified most important features were overall consistent with the researchers' findings and provided valuable insights.

Key insights from feature importances:

- Most influential features: `flow_bytes/s`, `total_length_of_fwd_packets`, `bwd_packet_length_std`, and `subflow_fwd_bytes`
- Less influential features: `bwd_urg_flags`, various bulk rate features, and `fwd_avg_bytes/bulk`
- Potential for dimensionality reduction to simplify the model without significantly impacting performance

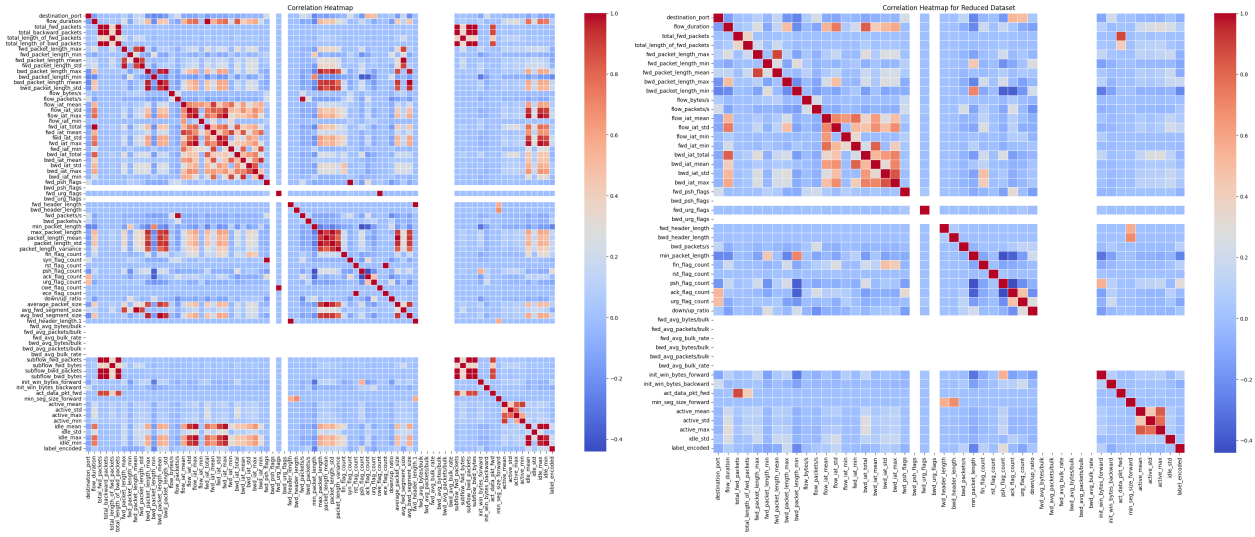
Feature Engineering

Feature engineering was carried out using a robust preprocessing module, structured following the builder software pattern to ensure flexibility and adherence to the DRY (Don't Repeat Yourself) principle. Below is an illustrative example of the preprocessing implementation:

```
# Initialize the preprocessor using the PreprocessorBuilder
preprocessor = PreprocessorBuilder() \
    .with_data_cleaning(fill_method="median") \
    .with_correlated_feature_removal(correlation_threshold=self.correlation_threshold) \
    .with_pca(pca_variance_ratio=self.pca_variance_ratio) \
    .with_one_hot_encoding() \
    .build()  # returns a Preprocessor object

# Utilize the preprocessor in the BasePipeline
data_loader, X_preprocessed, y_encoded, label_mappings = self.initialize(
    preprocessor=preprocessor
)
```

Dimensionality Reduction Initially, a correlation matrix was generated with a threshold of 90% to identify and remove redundant features. This analysis led to the elimination of 31 features that were highly correlated, simplifying the model without significantly affecting its performance. The removed features include 'subflow_bwd_packets', 'bwd_packet_length_mean', and 'fwd_iat_mean', among others.



The feature set was initially reduced from 79 to 48. Subsequently, Principal Component Analysis (PCA) was applied, further reducing the feature count to 26 components while retaining 95% of the variance. For

the final experiments, PCA alone was utilized with a higher correlation threshold of 95%, which effectively reduced the dataset to 24 dimensions:

```
$ dosdetect --bilstm-cnn-correlation-threshold 1.0
```

Following dimensionality reduction, data normalization and label encoding were performed, with labels ranging from 0 to 14 and being one-hot encoded, except for Logistic Regression. These preprocessing steps were managed by the `DataCleaner` module, which is also responsible for handling missing values by filling infinite and NaN values with the median by default. The use of the median fill method was selected with preliminary evidence suggesting its effectiveness; however, further exploratory data analysis is recommended to more thoroughly justify this choice and explore other potential methods.

Baseline Models

After the EDA phase, primitive baseline models were run to set benchmarks for the Bi-LSTM-CNN model. The selected baseline models included:

- K-Nearest Neighbors (KNN)
- Random Forest
- Logistic Regression

The model predictions for KNN and Random Forest were surprisingly accurate, although their generalizability was likely less robust than the Bi-LSTM-CNN model. Training the models on the entire CIC-IDS2017 dataset was found to be unfeasible on consumer hardware (Apple M1 Pro and M3 Pro were specifically tested); the dataset had to be reduced by a factor of 5-10 to achieve training convergence in a reasonable amount of time.

Pipeline Scaffolding

During the development of our project, a substantial amount of effort was dedicated to creating a PyPi (wheel) application equipped with a command-line interface (CLI) that facilitates training, evaluation, and exporting of the Bi-LSTM-CNN model. Initially conceived as a straightforward utility for fast and repeatable model training, this application evolved into a robust tool capable of managing not only the Bi-LSTM-CNN but also baseline models such as KNN, Random Forest, and Logistic Regression from the SciKit-Learn suite.

This CLI application is designed with 23 configurable flags, primarily utilized to adjust or set various hyperparameters. It features an auto-tuning capability that employs KerasTuner to automatically find the best hyperparameters, which are subsequently saved to a file. This aids in replicating model evaluations accurately.

Key flags include:

- `--dataset`: This flag allows users to specify the path to the CIC-IDS2017 dataset, making the application adaptable to other sequential CSV datasets with a 'Label' classification feature.
- `--train-fraction`: Users can train the model on a subset of the original dataset, which proves invaluable during initial development and debugging phases.
- `--log-dir` and `--model-dir`: These flags are used to designate directories for storing logs, exported files (such as model parameters in .keras and .pkl formats), evaluated performances, and model details. By default, all pipelines create a unique directory within `~/.dosdetect` appended with a timestamp suffix.

The CLI further supports individual model configurations through additional flags specific to each model type:

- **Bi-LSTM-CNN Hyperparameters**:
  - Correlation threshold and PCA variance ratio are settable with defaults of 0.9 and 0.95, respectively, to control data preprocessing.
  - The number of epochs and batch size for training are customizable, with defaults of 10 epochs and a batch size of 32.
- **Random Forest Hyperparameters**:

  - Similar to Bi-LSTM-CNN, settings for correlation threshold and PCA variance ratio are provided.
  - The model configuration can be finely tuned with parameters like the number of estimators, maximum depth (with a default of `None` indicating unlimited growth of trees), and a random state.
- **Logistic Regression Hyperparameters**:
  - This model also uses correlation threshold and PCA variance ratio settings.
  - Parameters such as inverse of regularization strength (`C`), maximum number of iterations, and random state enhance the flexibility in model training and regularization strength to prevent overfitting.

Each of these flags is designed to provide granular control over the training and evaluation processes, ensuring that the models can be optimally configured for the best possible performance on the tasks at hand.

Bi-LSTM Model Architecture

The model architecture consists of a combination of Bidirectional Long Short-Term Memory (Bi-LSTM) layers, a Convolutional Neural Network (CNN) layer, and fully connected (Dense) layers. Let's break down each component of the model:

Input Layer The model starts with feeding the input data directly into the first LSTM layer. The input shape is determined by a configurable `input_shape` variable, which should be set according to the dimensions of the input features.

Bidirectional LSTM Layers The model includes two Bidirectional LSTM layers. LSTM is a type of recurrent neural network (RNN) that can effectively capture long-term dependencies in sequential data. The Bidirectional wrapper allows the LSTM to process the input sequence in both forward and backward directions, enhancing the model's ability to capture contextual information. The first Bi-LSTM layer has a configurable number of units (ranging from 64 to 256 with a step of 64) determined by the hyperparameter tuning (hp) or set to a default value of 128. The second Bi-LSTM layer has a similar configuration but with a smaller range of units (32 to 128 with a step of 32) and a default value of 64.

Notably, both Bi-LSTM layers have `return_sequences=True` to ensure that the output shape is compatible with the subsequent Conv1D layer. The output shape of the last LSTM layer would otherwise be (batch_size, units) and the subsequent Conv1D layer expects an input shape of (batch_size, timesteps, features). Toggling return_sequences on ensures that the output shape of the last LSTM layer is (batch_size, timesteps, units), which is compatible with the input shape expected by the Conv1D layer.

Convolutional Layer Following the Bi-LSTM layers, a Conv1D layer is added to perform convolutional operations on the sequential data. The Conv1D layer applies a set of filters (configurable or default to 64) with a kernel size of 3 to extract local patterns and features from the Bi-LSTM outputs. The activation function used in this layer is ReLU (Rectified Linear Unit), which introduces non-linearity and helps the model learn complex patterns.

Flatten Layer The Flatten layer is used to convert the 3-dimensional output of the Conv1D layer into a 1-dimensional vector. This step is necessary to prepare the data for the subsequent fully connected layers.

Dense Layers The model includes two fully connected (Dense) layers. The first Dense layer has a configurable number of units (ranging from 32 to 128 with a step of 32) determined by hyperparameter tuning or set to a default value of 64. It uses the ReLU activation function to introduce non-linearity. The second Dense layer is the output layer, and its number of units depends on the type of classification task. Notably, support for both binary classification and multi-class classification. For binary classification, it has a single unit with a sigmoid activation function. For multi-class classification, it has `self.num_classes` units with a softmax activation function.

Dropout Layer Between the two Dense layers, a Dropout layer is added to help prevent overfitting. Dropout randomly sets a fraction of the input units to 0 during training, which helps the model generalize better to unseen data. The dropout rate is configurable (ranging from 0.1 to 0.5 with a step of 0.1) or set to a default value of 0.5.

This Bi-LSTM-CNN model combines the strengths of recurrent neural networks (Bi-LSTM) for capturing long-term dependencies and convolutional neural networks (CNN) for extracting local patterns. The Bi-LSTM layers process the input sequence in both directions, the Conv1D layer captures local features, and the Dense layers perform the final classification. The model also includes hyperparameter tuning options to optimize the architecture for specific tasks. This combination of layers allows the model to effectively learn and classify sequential data, making it suitable for various applications such as text classification, sentiment analysis, or time series prediction.

Pipeline Diagram

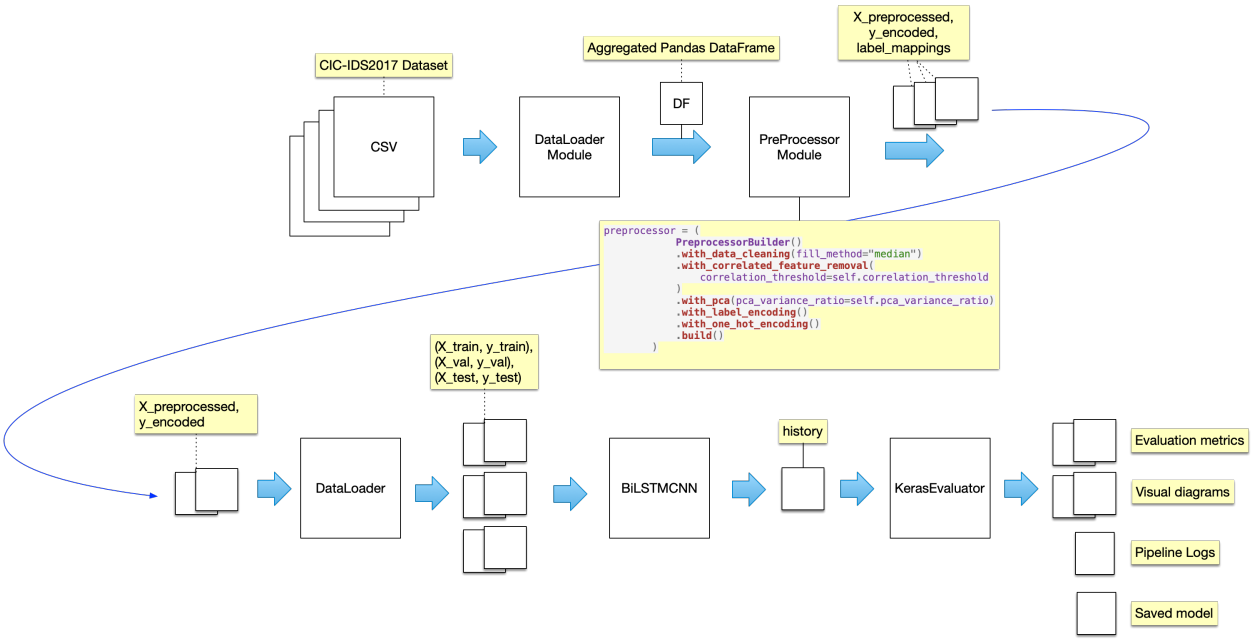


Figure 2: Bi-LSTM-CNN Pipeline Workflow

Model Diagram

Datasets

This study utilized the CICIDS2017 dataset, a comprehensive collection designed to mirror a variety of real-world network traffic and attack scenarios, including Distributed Denial of Service (DDoS) attacks. This dataset is pivotal for training and evaluating our machine learning models, ensuring that they are tested against realistic network behaviors and attack patterns.

CIC-IDS2017 Dataset Overview:

- **Format:** The data is available in two formats:
  - **CSV:** Total size is approximately 880 MB, making it manageable for processing and analysis.
  - **PCAP:** Total size is 52.25 GB, providing extensive raw network traffic data that is ideal for in-depth forensic analysis.
- **Samples:** The dataset contains a total of 2,830,743 data points.
- **Features:** There are 79 features in each sample, encompassing a wide range of traffic attributes and derived statistics.

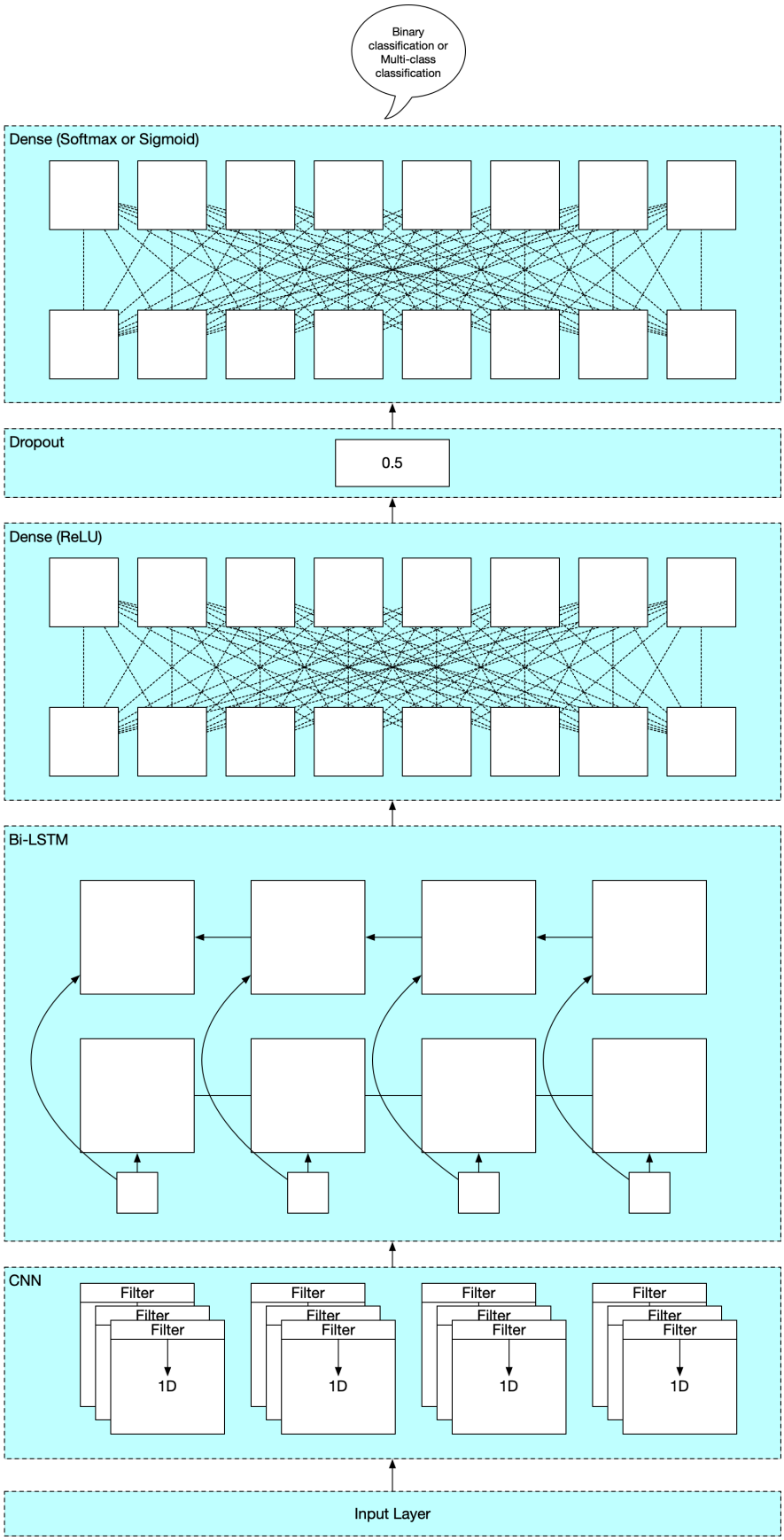


Figure 3: Bi-LSTM-CNN Model Diagram

- **Data Types:** The features are primarily of integer (`int64`) and floating-point (`float64`) types, facilitating detailed numerical analysis.

Data Split Ratios:

To facilitate the training, validation, and testing of our models, we partitioned the data into distinct sets using the following ratios:

- **Training Set:** 60% of the data (`train_size=0.6`), used for fitting the models.
- **Validation Set:** 20% of the data (`val_size=0.2`), used for tuning the models' hyperparameters and making adjustments to model configurations.
- **Testing Set:** 20% of the data (`test_size=0.2`), used to evaluate the final performance of the models and ensure that they generalize well to new, unseen data.

What is Considered Success/Failure?

- **Success:** Achieving a detection accuracy of 95% or higher with a low rate of false positives and negatives, thus outperforming traditional detection mechanisms.
- **Failure:** Inability to significantly outperform baseline methods or achieve a practical balance between detection accuracy and false positive rate.

Evaluation Parameters and Experimental Setup

Environment

The preliminary development of our models was performed on a machine equipped with an Apple Silicon M1 Pro chip, featuring:

- An 8-core CPU (6 performance cores and 2 efficiency cores)
- A 14-core GPU

We utilized the GPU capabilities of the machine for training all our Keras-based Bi-LSTM-CNN models due to their computational intensity. Conversely, our baseline models, developed using SciKit Learn, were trained on the CPU, as SciKit Learn does not natively support GPU training.

For final production-level training, we transitioned to a more robust setup:

- **Platform:** AWS EC2
- **Instance Type:** g5.12xlarge
  - **GPU:** 4 units
  - **vCPU:** 48 cores
  - **Memory:** 192 GiB
- **AMI:** Deep Learning OSS Nvidia Driver AMI GPU TensorFlow 2.15 (Ubuntu 20.04), version 20240410

Bi-LSTM-CNN Results

The Bi-LSTM-CNN pipeline, optimized for DDoS attack detection, was executed using CUDA on our robust AWS setup. The process commenced at 15:08:21 on April 15, 2024, and concluded at 16:24:00, totaling approximately 1 hour and 15 minutes.

Pipeline Configuration: The pipeline was configured with auto-tuning enabled and utilized the full dataset (train fraction set to 1.0). The correlation threshold was set to 1.0, with a PCA variance ratio of 0.95. Training involved 10 epochs and a batch size of 32, which were adequate given our computational resources and dataset characteristics.

Model Hyperparameters:

```
{
    "input_shape": [25, 1],
    "num_classes": 15,
```

```
    "optimizer": {
        "name": "Adam",
        "learning_rate": "0.001",
        "beta_1": 0.9,
        "beta_2": 0.999,
        "epsilon": 1e-07,
        "jit_compile": true
    },
    "loss": "categorical_crossentropy",
    "metrics": ["loss", "accuracy"]
}
```


Inference Performance: The model's performance metrics are detailed in the table below, showing its ability to classify various attack types with high accuracy. Notably, the model excelled in distinguishing benign traffic and DDos attacks, while showing some limitations in identifying less frequent attack patterns like Bot and Heartbleed, where both precision and recall were significantly lower.

Class	Accuracy	Precision	Recall	F1 Score
Benign	97.99%	99.03%	98.47%	98.75%
Bot	99.93%	0.00%	0.00%	0.00%
DDos	99.91%	99.81%	98.26%	99.03%
Dos GoldenEye	99.94%	96.47%	88.31%	92.21%
Dos Hulk	99.26%	93.79%	97.39%	95.56%
DoS Slowhttptest	99.95%	86.06%	91.45%	88.67%
DoS Slowloris	99.96%	97.69%	80.40%	88.20%
FTP-Patator	99.97%	92.14%	97.84%	94.90%
Heartbleed	100.00%	0.00%	0.00%	0.00%
Infiltration	100.00%	0.00%	0.00%	0.00%
PortScan	99.10%	89.36%	95.29%	92.23%
SSH-Patator	99.96%	86.66%	97.64%	91.82%
Web Attack: Brute Force	99.95%	100.00%	3.15%	6.12%
Web Attack: Sql Injection	100.00%	0.00%	0.00%	0.00%
Web Attack: XSS	99.97%	0.00%	0.00%	0.00%

Baseline Model Evaluation

KNN Results The evaluation of the K-Nearest Neighbors (KNN) model was conducted efficiently, commencing on April 15, 2024, at 14:49:20 and concluding at 14:57:36, with a total duration of approximately 8 minutes and 16 seconds. The configuration for the KNN pipeline did not utilize auto-tuning, and it engaged the entire dataset (train fraction set to 1.0), with a correlation threshold of 0.9 and a principal component analysis (PCA) variance ratio of 0.95. The number of neighbors was set at five, aiming to achieve a balance between sensitivity to local data structures and overfitting.

The model exhibited strong performance across various classes, particularly excelling in identifying benign traffic, DDoS, and Dos Hulk attacks with notable accuracy and precision. The detailed performance metrics are presented below, indicating the model's effectiveness in differentiating between normal and malicious traffic:

Class	Accuracy	Precision	Recall	F1 Score
0 (Benign)	99.44%	99.78%	99.52%	99.65%
1 (Bot)	99.95%	71.56%	59.01%	64.68%
2 (DDos)	99.99%	99.92%	99.91%	99.91%
3 (Dos GoldenEye)	99.99%	97.20%	98.86%	98.02%

Class	Accuracy	Precision	Recall	F1 Score
4 (Dos Hulk)	99.95%	99.68%	99.72%	99.70%
5 (DoS Slowhttptest)	99.99%	98.24%	98.41%	98.33%
6 (DoS Slowloris)	99.99%	98.62%	98.88%	98.75%
7 (FTP-Patator)	100.00%	99.81%	99.32%	99.57%
8 (Heartbleed)	100.00%	100.00%	80.00%	88.89%
9 (Infiltration)	100.00%	33.33%	16.67%	22.22%
10 (PortScan)	99.58%	94.54%	98.04%	96.26%
11 (SSH-Patator)	99.99%	95.67%	98.82%	97.22%
12 (Web Attack: Brute Force)	99.97%	70.43%	76.66%	73.41%
13 (Web Attack: Sql Injection)	100.00%	0.00%	0.00%	0.00%
14 (Web Attack: XSS)	99.97%	48.42%	30.46%	37.40%

Logistic Regression Results The Logistic Regression model's evaluation was conducted over a brief period, beginning on April 15, 2024, at 15:32:55, and concluding shortly after at 15:34:52. This resulted in a pipeline execution time of approximately 2 minutes and 3 seconds. The model was configured with specific parameters, focusing on a full dataset utilization (train fraction set to 1.0), a correlation threshold of 0.9, and a principal component analysis (PCA) variance ratio of 0.95. The regularization strength parameter (C) was set at 1.0 with a maximum of 100 iterations, without specifying a random state, to ensure repeatability.

The performance of the Logistic Regression model across various classes demonstrates a varied effectiveness in identifying different attack types, especially highlighting its strengths and limitations in classifying benign traffic and several specific attacks. The detailed performance metrics, formatted as percentages, are as follows:

Class	Accuracy	Precision	Recall	F1 Score
0 (Benign)	95.99%	97.73%	97.28%	97.50%
1 (Bot)	99.93%	0.00%	0.00%	0.00%
2 (DDos)	99.55%	98.96%	90.92%	94.77%
3 (Dos GoldenEye)	99.79%	84.05%	52.09%	64.32%
4 (Dos Hulk)	98.85%	91.49%	94.69%	93.07%
5 (DoS Slowhttptest)	99.88%	71.76%	62.52%	66.82%
6 (DoS Slowloris)	99.88%	95.98%	45.34%	61.58%
7 (FTP-Patator)	99.71%	48.84%	3.90%	7.22%
8 (Heartbleed)	100.00%	0.00%	0.00%	0.00%
9 (Infiltration)	100.00%	0.00%	0.00%	0.00%
10 (PortScan)	98.56%	79.84%	99.12%	88.44%
11 (SSH-Patator)	99.79%	0.00%	0.00%	0.00%
12 (Web Attack: Brute Force)	99.94%	0.00%	0.00%	0.00%
13 (Web Attack: Sql Injection)	100.00%	0.00%	0.00%	0.00%
14 (Web Attack: XSS)	99.97%	0.00%	0.00%	0.00%

Random Forest Results The evaluation of the Random Forest model spanned a timeframe from 15:02:56 to 15:58:07 on April 15, 2024, totaling approximately 55 minutes and 11 seconds. The model was configured to use the entire dataset (train fraction set to 1.0), a correlation threshold of 0.9, and a principal component analysis (PCA) variance ratio of 0.95. The Random Forest algorithm was implemented with 100 estimators, and no maximum depth was specified to allow the trees to expand until all leaves are pure or until all leaves contain less than the minimum split samples.

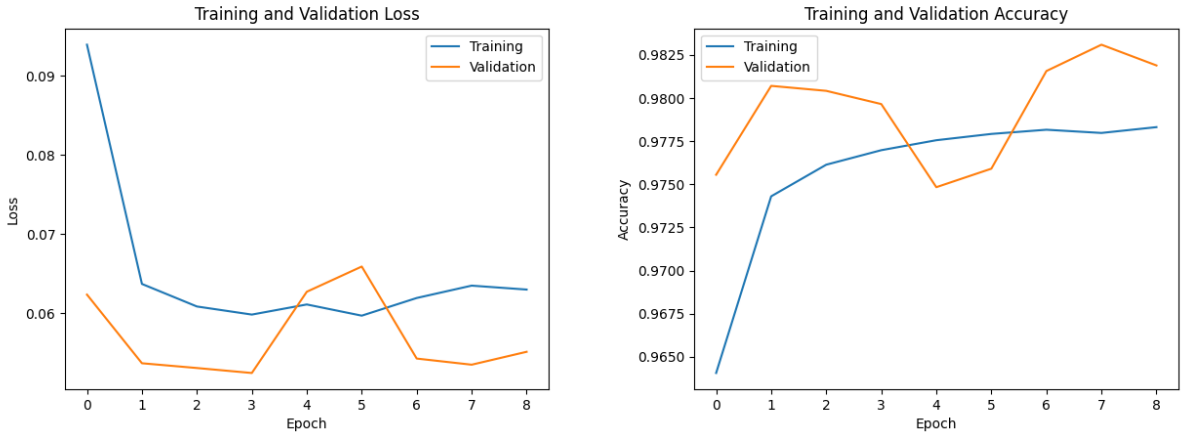
The Random Forest model demonstrated strong performance across various classes, as shown in the detailed metrics table below, formatted as percentages. The model excelled particularly in identifying benign traffic and several types of network attacks, achieving near-perfect scores in accuracy, precision, recall, and F1 score for most categories.

Class	Accuracy	Precision	Recall	F1 Score
0 (Benign)	99.80%	99.88%	99.87%	99.87%
1 (Bot)	99.96%	80.86%	60.49%	69.21%
2 (DDos)	99.99%	99.96%	99.93%	99.95%
3 (Dos GoldenEye)	99.98%	98.98%	96.82%	97.89%
4 (Dos Hulk)	99.97%	99.80%	99.82%	99.81%
5 (DoS Slowhttptest)	99.99%	98.77%	98.77%	98.77%
6 (DoS Slowloris)	99.99%	99.47%	98.10%	98.78%
7 (FTP-Patator)	100.00%	99.88%	99.75%	99.81%
8 (Heartbleed)	100.00%	100.00%	80.00%	88.89%
9 (Infiltration)	100.00%	50.00%	16.67%	25.00%
10 (PortScan)	99.89%	98.79%	99.32%	99.05%
11 (SSH-Patator)	99.99%	99.66%	98.99%	99.32%
12 (Web Attack: Brute Force)	99.97%	72.29%	79.81%	75.86%
13 (Web Attack: Sql Injection)	100.00%	50.00%	14.29%	22.22%
14 (Web Attack: XSS)	99.97%	47.17%	33.11%	38.91%

Results

The focus of this study is to evaluate the efficacy of the advanced Bi-LSTM-CNN model in detecting DDoS and other network attacks, and to ascertain whether it outperforms traditional baseline models such as Random Forest, KNN, and Logistic Regression. Our experimental findings highlight distinct patterns and insights regarding each model's capabilities and limitations.

Bi-LSTM-CNN Model: This sophisticated model, leveraging both LSTM and CNN layers to analyze network traffic data, demonstrated high precision and accuracy, particularly in detecting benign and common attack types like DDos and Dos Hulk. The model achieved exceptional performance with F1 scores frequently surpassing 95% in these categories, indicating its robustness in handling sequences and spatial features effectively. However, the model struggled with lower-frequency attack types such as Bot and Heartbleed, where both precision and recall were notably poor. This suggests a potential need for further tuning, possibly through data augmentation or more specialized feature extraction techniques.



KNN Model: The KNN model displayed excellent accuracy and was particularly effective for detecting major attack types like DDoS and Dos Hulk with high precision and recall. However, it fell short in detecting more complex or less frequent attack types such as SQL Injection and XSS attacks, where it exhibited very low recall rates. This indicates that while KNN is robust for more straightforward classification tasks, its performance degrades without features that distinctly separate all classes in the multi-dimensional feature space.

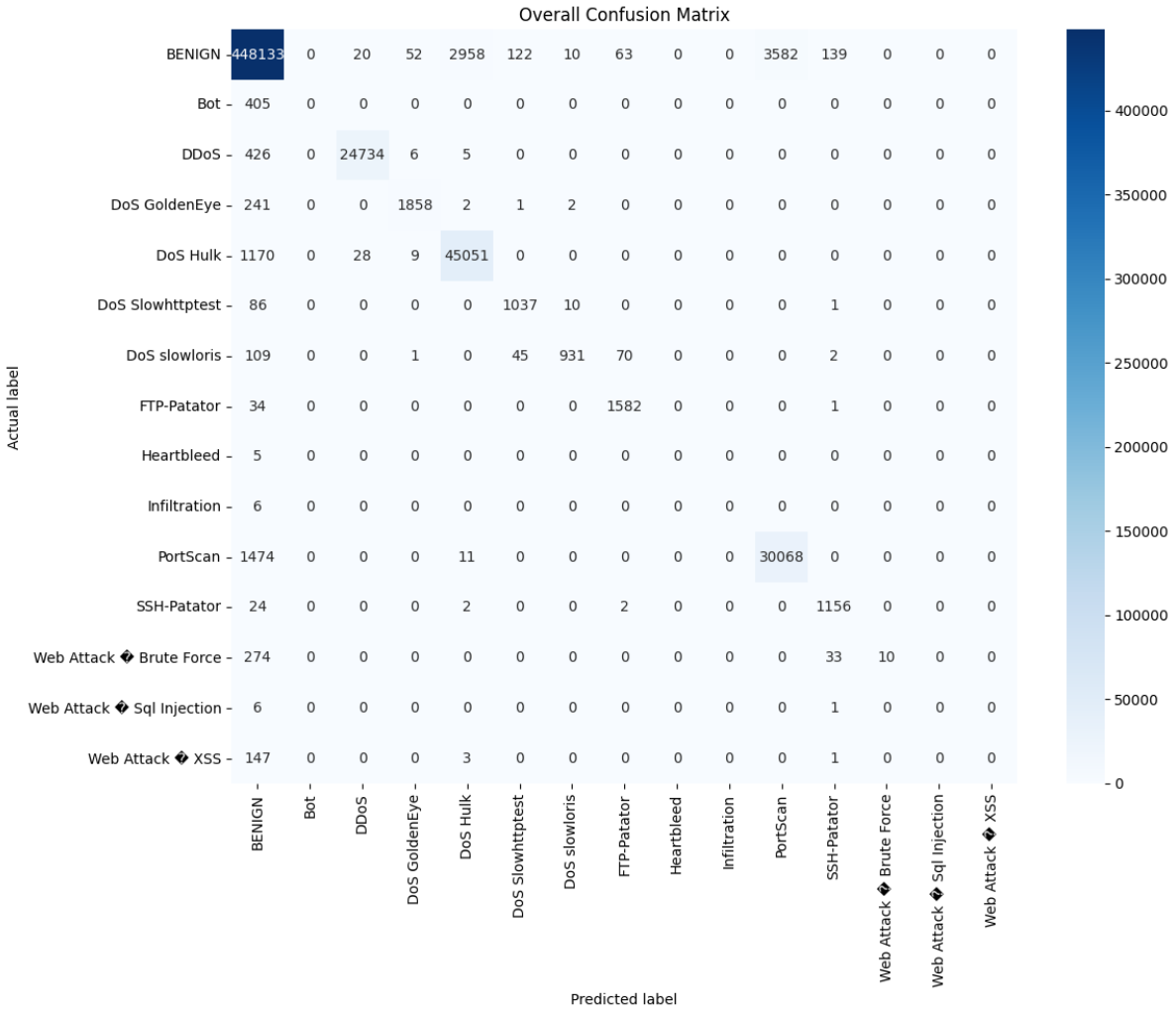


Figure 4: Confusion Matrix - Bi-LSTM-CNN

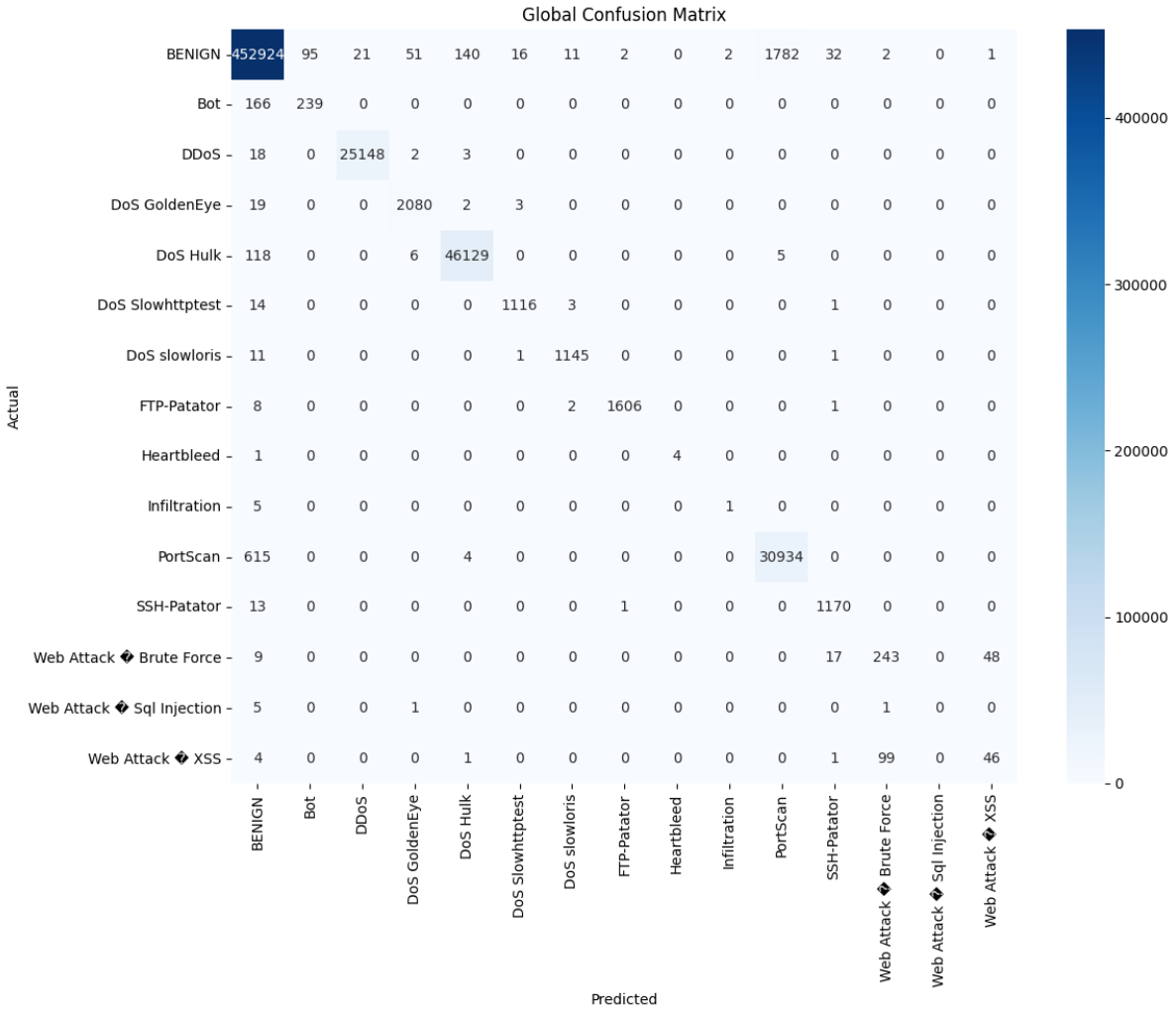


Figure 5: Confusion Matrix - KNN

Random Forest Model: This model achieved impressive results across most categories, with particularly strong performance in identifying benign traffic and several types of DDoS attacks. The ensemble nature of Random Forest, which combines decisions from multiple decision trees, helps in reducing overfitting and improving the generalizability across diverse attack vectors. However, similar to the Bi-LSTM-CNN, it demonstrated weaknesses in identifying infrequent attacks like Heartbleed and Infiltration, highlighting potential areas for improvement in feature engineering or class imbalance handling.

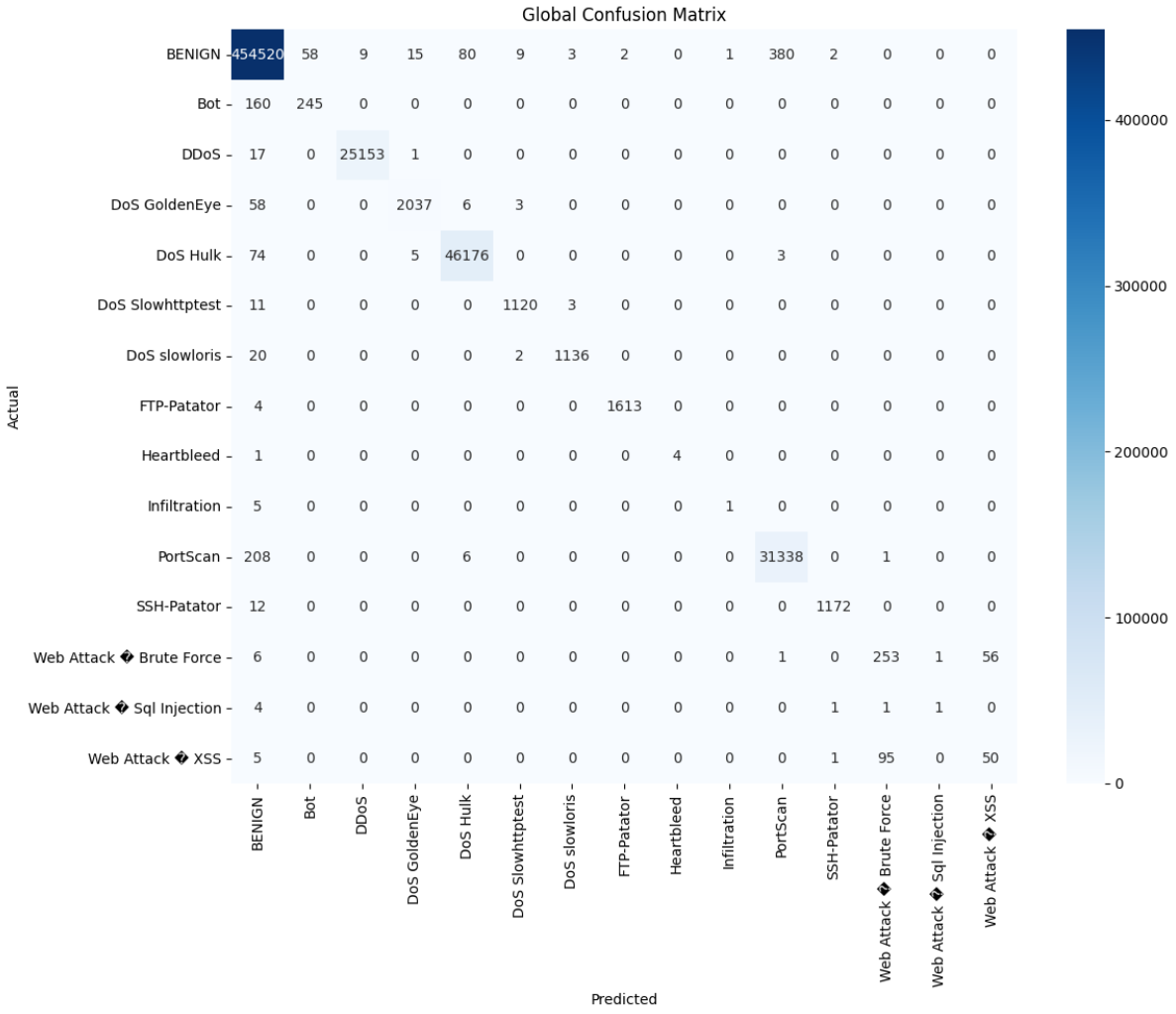


Figure 6: Confusion Matrix - Random Forest

Logistic Regression Model: Logistic Regression provided a good baseline but did not achieve the high levels of accuracy seen in the other models. It performed well in detecting straightforward attack patterns, such as those in DDoS and Dos Hulk, but was less effective for attacks that exhibit subtler signature variations. Its primary limitation was evident in its inability to handle classes with fewer examples or more complex decision boundaries, leading to very low or zero recall in several minor attack categories.

Comparisons

The following are the ROC AUC curves for each of the models:

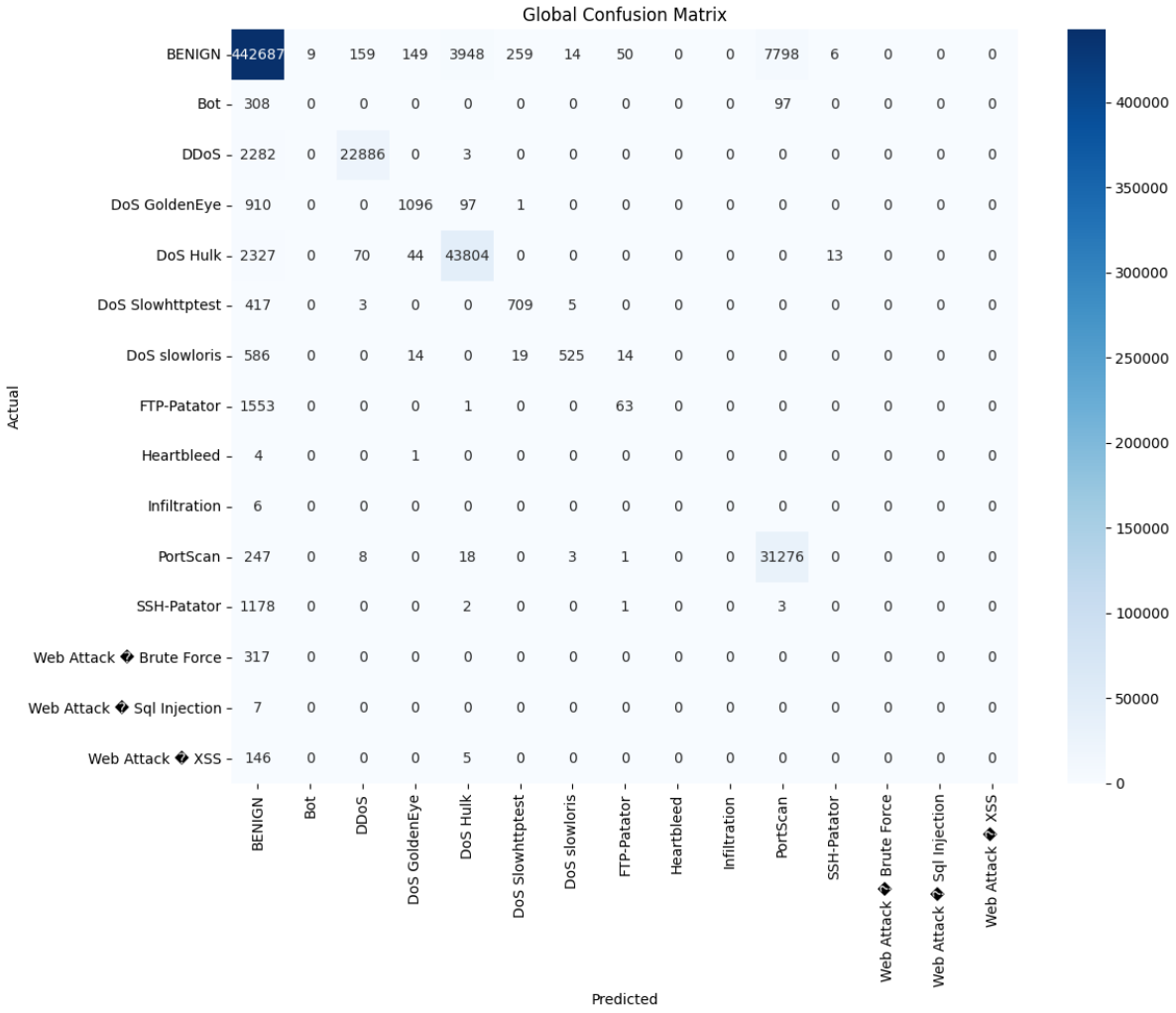
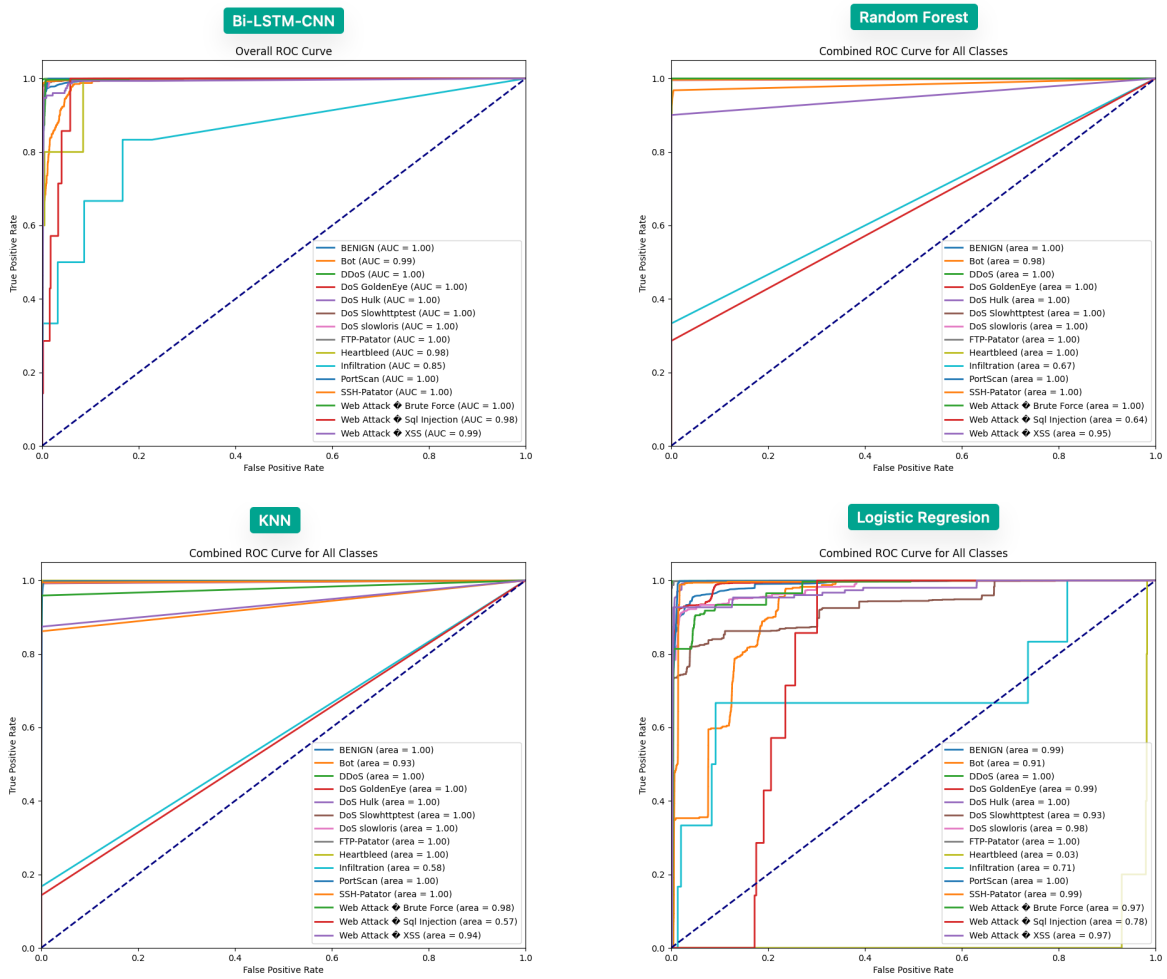
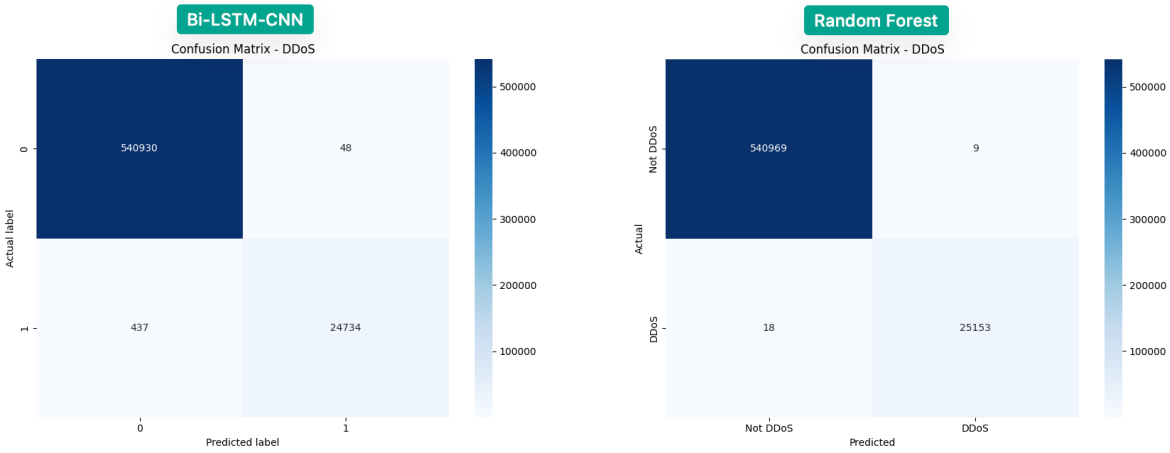
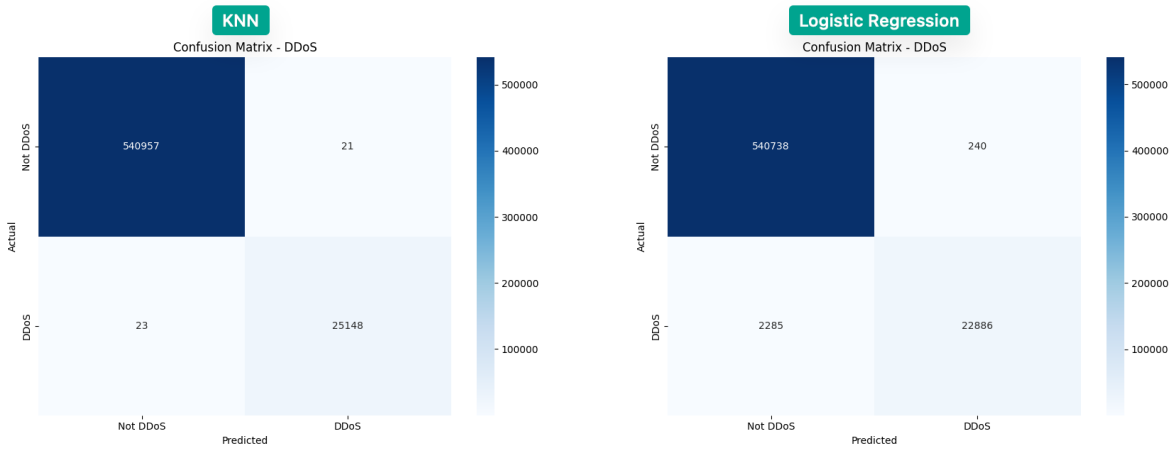


Figure 7: Confusion Matrix - Logistic Regression



Here are the confusion matrices for DDoS detection. Random Forest performed the best, with only 9 false positives and 18 false negatives. KNN was the second most effective, with 21 false negatives and 23 false positives. The Bi-LSTM-CNN model ranked third, with 48 false positives and 437 false negatives. Logistic Regression was the least effective, with 240 false positives and 2285 false negatives.





Success and Failure Criteria:

- **Success** is defined as achieving a detection accuracy of 95% or higher with a low rate of false positives and negatives, thereby surpassing the performance of traditional detection mechanisms. By this criterion, the Bi-LSTM-CNN model cannot be considered an unequivocal success, as it did not consistently outperform the baseline models. However, this does not imply that the model was unsuccessful. It still demonstrated high accuracy and could potentially be more robust than the baseline models, although additional testing would be necessary to substantiate such claims.
- **Failure** would be indicated by an inability to significantly outperform the baseline models or to achieve a practical balance between detection accuracy and false positive rates. Here, the Bi-LSTM-CNN's challenges with less frequent attacks may suggest areas of failure, although its overall performance still indicates a strong potential against baseline models.

In conclusion, while the Bi-LSTM-CNN model demonstrated superior capabilities in several aspects of network attack detection, its performance variability across different attack types suggests that there is still room for improvement. Enhancing the model's ability to detect rarer attack types without compromising accuracy on more common attacks could further solidify its advantage over traditional models. Meanwhile, the baseline models, particularly Random Forest, provide robust alternatives that continue to perform competitively in many scenarios.

Constraints

Computational Resources

One of the primary constraints encountered was the limitation imposed by available hardware for training complex models. Initial training attempts on consumer-grade hardware, specifically the Apple M1 Pro, proved unsuccessful due to prolonged durations exceeding 12 hours. To address this, a g5.12xlarge EC2 instance was provisioned, leveraging CUDA to utilize GPUs, which significantly accelerated the training and optimization of the Bi-LSTM-CNN model. The training time was reduced from over 12 hours to less than 1 hour. However, this solution introduced financial constraints, as operating the g5.12xlarge instance incurs a cost of approximately $5.60 per hour when used on-demand.

Data Skewness

The dataset used in this study exhibits significant class imbalance, with a heavy bias towards benign (non-malicious) traffic. This skewness is representative of real-world network conditions, where non-malicious traffic predominates. However, this imbalance poses a substantial challenge, as it predisposes the resultant models to favor benign classifications disproportionately. Such bias can undermine the effectiveness of the

model in identifying and classifying malicious activities accurately, necessitating strategies to appropriately manage and mitigate the effects of this skewness.

Standards

This study adheres to the highest ethical standards in artificial intelligence, particularly emphasizing the protection of privacy and compliance with data protection regulations during all phases of data handling and model training. The dataset utilized, CIC-IDS2017, comprises synthetic network traffic generated in a controlled laboratory setting, ensuring no real-world user data or activities are implicated. This approach not only mitigates ethical concerns but also aligns with best practices in data security and privacy. However, the potential biases inherent in the models derived from this synthetic dataset underscore the necessity for cautious interpretation of the results. The models and methodologies developed through this project are primarily intended for academic and exploratory purposes. Given this, we explicitly advise against deploying these models in production environments without extensive further validation. This recommendation is aimed at preventing any unforeseen consequences that might arise from differences between the controlled experimental conditions and real-world network environments. Moving forward, it is critical to maintain these ethical standards, particularly as the technology develops and potentially integrates into broader, more diverse operational contexts.

Limitations of the Study

This study, while comprehensive in its approach to detecting DDoS attacks using a Bi-LSTM-CNN model, encounters several limitations that could affect the generalizability and efficiency of the results:

1. **Dataset Dependence:** The study relies heavily on the CIC-IDS2017 dataset, which, while robust, represents a controlled environment rather than real-world network traffic variability. This dependence might limit the model's performance when applied to different or more recent network traffic patterns not represented in the dataset.
2. **Overfitting Potential:** Despite the promising results achieved by the baseline models (KNN, LR, and RF), the high performance could be indicative of overfitting, especially given the complex nature of the network traffic data. While the study reports minimal differences between training and validation accuracies, the risk of overfitting in real-world applications remains a concern and warrants further validation.
3. **Lack of Attention Mechanism:** The model architecture does not include an Attention mechanism, which has been shown to improve performance in tasks requiring focus on specific features within sequences. This omission, due to time constraints and current expertise levels, could be limiting the model's ability to dynamically adapt to new types of DDoS attacks.
4. **Hardware Limitations:** Initial training attempts on consumer-grade hardware like the Apple M1 Pro led to significant delays and occasional system freezes, suggesting that the computational demands of the model exceed what is feasible without specialized hardware. Although switching to AWS hardware mitigated these issues, this dependency on high-performance computing resources could limit the accessibility and scalability of deploying such models in less resource-rich environments.
5. **Unresolved Freezing Issue:** The frequent freezing of the Python runtime during prolonged training periods remains unresolved. Although not a direct limitation of the model architecture, it points to potential stability issues in the training process that could impact the practical deployment of the model.
6. **Future Implementations:** The study did not implement unsupervised learning techniques or real-time network environment tests, which could further enhance the model's detection capabilities and robustness. Additionally, the potential for using the model in production environments without extensive further validation could risk deploying a system that is not fully optimized for real-world operations.

By addressing these limitations in future work, the study can enhance the robustness and applicability of the proposed model, potentially leading to more reliable and efficient DDoS detection systems.

Future Work

This study lays the groundwork for several promising avenues that could significantly enhance the detection and generalizability of DDoS attack models:

1. **Integration of Unsupervised Learning Techniques:** Exploring unsupervised learning methods could potentially reduce reliance on extensive feature engineering and preprocessing. By incorporating techniques such as autoencoders or clustering, the model could improve its ability to detect novel attack patterns that were not labeled in the training data.
2. **Real-World Environment Testing:** To validate the efficacy and robustness of the model, deploying it in a live network environment would provide invaluable insights. This step would involve adapting the data ingestion pipeline to process raw network packet captures (PCAPs) directly, utilizing tools like 'tshark' for real-time data conversion and analysis.
3. **GPU Acceleration for SciKit Learn Models:** Refactoring the existing baseline models to utilize GPU resources could substantially decrease training and inference times. Implementing API mapping libraries, such as SciKeras, would bridge the compatibility gap between SciKit Learn and GPU-supported backends like TensorFlow/Keras.
4. **Visualization of CNN Feature Maps:** Developing a sub-module for the visualization of convolutional neural network feature maps would allow for a better understanding of model behavior and decision-making processes. These visualizations could help identify specific patterns or features that the model uses to distinguish between different types of network traffic, providing opportunities for further optimization.
5. **Commodification of Machine Learning Pipelines:** By packaging the machine learning pipelines into a hostable RPC interface, such as a REST API or GraphQL API, the model could be more easily integrated into existing network management frameworks. Streamlining deployment on platforms like AWS or RapidAPI would make the solution more accessible and practical for real-world applications.
6. **Ensemble Models for Specific Attack Types:** Considering the development of specialized models for each type of attack detected in the CIC-IDS2017 dataset could lead to more tailored and effective solutions. An ensemble approach, where each model is optimized for a particular attack type, may yield better performance than a general model across all types.
7. **Open Source Contribution:** Publishing the `dosdetect` application on PyPi would facilitate wider use and collaboration. Implementing continuous integration and continuous deployment (CI/CD) pipelines could ensure the reliability and maintainability of the application, promoting ongoing improvements and community engagement.

By pursuing these directions, the project can advance the state of the art in DDoS detection, making the technologies developed more robust, efficient, and applicable to a wider range of real-world scenarios.