Deep Learning Project : Intrusion Detection System (IDS) Group-2

Guillaume Pealat , Afra Muhammad, Wafa Bouakez, Susmitha Ann Alex

Introduction

Network security traditionally relies on signature-based detection (e.g., Snort, Suricata) or machine learning models analyzing packet headers and payload statistics. However, modern cyberattacks—such as zero-day exploits, polymorphic malware, and adversarial evasion techniques—increasingly bypass these methods by mimicking benign traffic or using obfuscated payloads.

This project proposes **an unconventional use of Natural Language Processing (NLP) and Computer Vision (CV) to detect malicious TCP/IP packets** by interpreting raw network traffic as both structured textual data and visual patterns. Unlike traditional approaches, we treat packet analysis as:

An NLP problem: Analyzing byte sequences as "language" (e.g., ASCII/hex payloads, protocol syntax) using transformers/RNNs to detect anomalies.

A CV problem: Visualizing packets as 2D entropy maps, byte distribution heatmaps to spot structural irregularities.

Why This Approach is Unique

Most intrusion detection systems (IDS) focus on:

Rule-based matching (limited to known threats).

Statistical features (e.g., flow duration, packet size), which miss subtle payload-level attacks.

Our method innovates by:

Tokenizing packets like text: Treating raw bytes (hex/ASCII) as a "language" for NLP models (e.g., BERT for protocol anomaly detection).

Image-based anomaly detection: Converting packets into grayscale/spectrogram-like images and applying CNNs to localize malicious segments.

I. Dataset

1. Raw dataset

The **CSE-CIC-IDS2018** dataset is a comprehensive network intrusion detection dataset developed collaboratively by the **Canadian Institute for Cybersecurity (CIC)** and the **Communications Security Establishment (CSE)**. It is designed to provide a realistic and up-to-date benchmark for evaluating intrusion detection systems (IDS) and machine learning-based security solutions.

It can be found at the following url: IDS2018 dataset

Key Features:

- **Realistic Traffic:** Contains network traffic generated in a simulated environment mimicking real-world user behavior, including normal activities and diverse attack scenarios.
- **Modern Attacks:** Covers common and evolving cyber threats such as **Brute Force, Denial of Service (DoS), Web Attacks, Infiltration, Botnet, and Insider Attacks**.
- **Rich Metadata:** Provides full packet capture (PCAP) files along with extracted network flows (NetFlow) and labeled logs for supervised learning.
- **Balanced Data:** Includes both benign and malicious traffic, enabling the training and testing of anomaly-based and signature-based detection models.
- **Structured Format:** Available in CSV files with 80+ network flow features, including timestamps, protocols, ports, payload sizes, and attack labels.

The raw dataset is extremely large, so to use it we needed to extract the relevant information for our study.

2. Extracted dataset

a. Extracted attacks

We didn't extract all the type of possible attacks. Given the over representation of DoS attacks, we decided to only use 6 possible attacks: DoS-GoldenEye, Infiltration, DoS-SlowLoris, BruteForce-Web, BruteForce-XSS and SQL Injections.

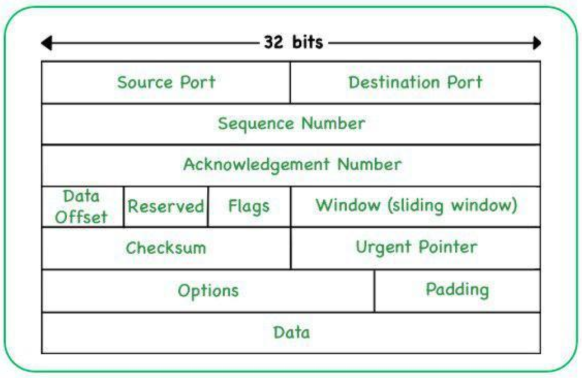
We did so to have a good representation of different attacks, being either an external attack or an internal attack.

The documentation of the CIC-IDS2018 dataset has the benefit of clearly stating the exact time stamp of the attacks. It helped us in labelling the packets properly. For each packet, depending on the IP of the target or source machine, we labelled the packets as Malicious or Benign. We also used the documentation to have a finer labelling. Indeed, for a DoS attack, only the incoming packets are malicious, while on an infiltration attack, we should only look at outgoing packets.

b. Data extracted from the packets

From each packet, we extract information from both the TCP and the IP layer. Based on TCP/IP infrastructure, we can choose which information we want to extract.

<u>TCP Packet</u>

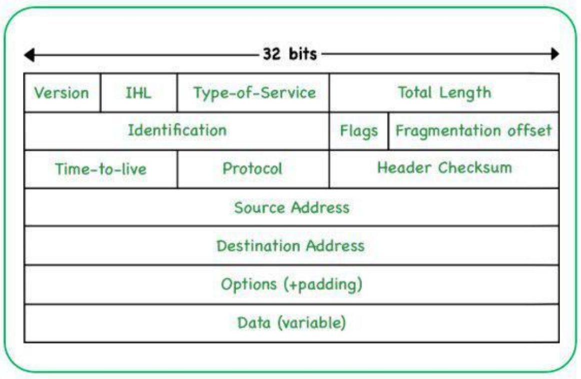


TCP packet architecture

We have decided to extract the following:

- Source & destination ports
- Data offset
- Flags
- Window size

<u>IP Packet</u>



IP Packet architecture

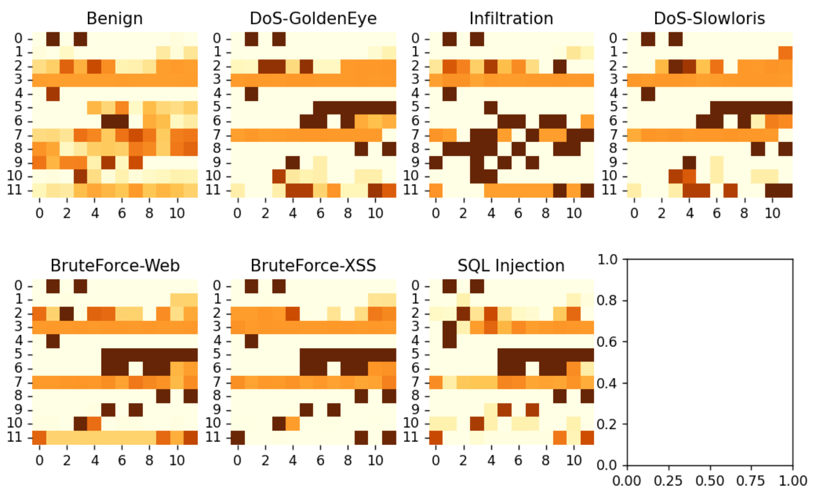
We have decided to extract the following:

- IHL
- Type Of Service
- Total Length
- Identification
- Flags
- Time to live
- Protocol

We do not extract the IP addresses as it is relevant information if we want to make the model to generalise. Indeed, as the attacker machines are always the same, they share a common IP address. Adding it to the data will create a data leakage which will confuse the model (for example the model will recognize the IP address and will decide it is a malicious packet based on this information only).

<u>Packet summary</u>

At the end of the data extraction, we have 144 bits of information in our dataset for each packet. We can display the information on a 12x12 image which can be segmented by attack type as below:



c. Data extracted for the model using CV

On top of the Header data, we also want to include the payload (if any).

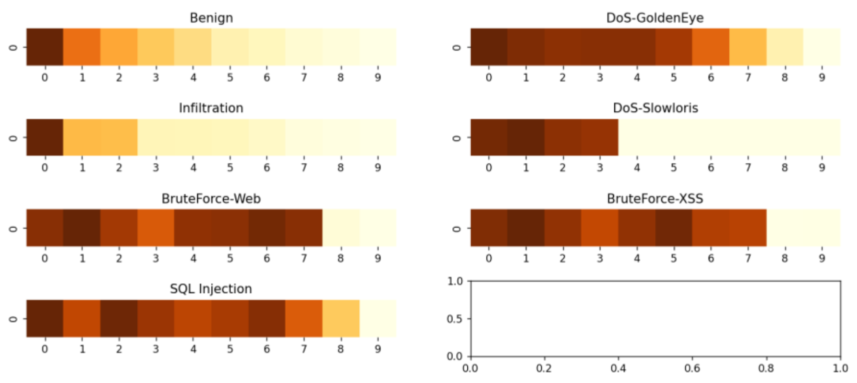
<u>Payload</u>

Contrary to the header, the payload does not have a fixed size. To cater for this fact, we decided to normalize the data using the following algorithm:

- If the payload does not exist, we fix it at a '000.0000' value.
- If it exists, we check if it is longer than 10 bits
    a. If not, we display the payload in a 10 bit long array (filling the missing value by adding a 0
    b. If it exists, we split the payload in quintile and calculate the Shannon entropy for each quintile

Shannon entropy measures the uncertainty or randomness in a dataset, often used to quantify information content. Higher entropy means more unpredictability, while lower entropy indicates more structure or redundancy**.**

Once done, we can plot average values for the entropy of each quintile depending on the attack type:



We see that it holds interesting properties as we can see visually the difference between each of them.

Summary of the data

After this step, we have the following dataset:

Column type	Number of columns	Description
Headers bit	144	Bits for the header
Entropy	10	Quintile entropy
Payload Encoding	1	Detected Encoding
Payload Strings	1	Extracted payload
Attack	1	Attack Name
Label	1	Malicious/Benign Label

This represents 162 columns, but we will not keep them all for the Computer Vision problem. We can drop the 2 columns about Payload Encoding and Payload Strings as they will be only needed for a NLP task. After this step our dataset memory footprint is 3.3GB, which is much smaller than the original packets sizes (but still significant).

Size reduction

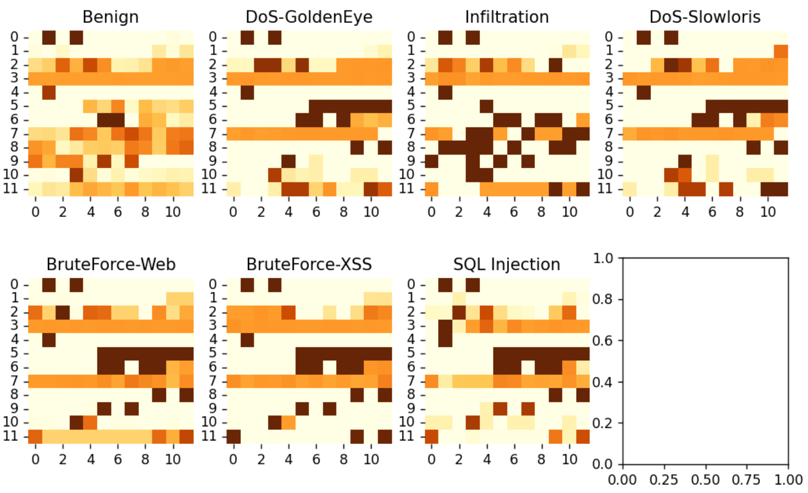
By default, pandas will try to convert to object it knows. But relying on this will result in a very heavy dataset as the types selected will not match the exact reality of the data. To reduce the overall memory footprint of the dataset, we have decided to use the following datatypes:

Column type	Number of columns	Datatype
Headers bit	144	float32
Entropy	10	uint8

Remainer	4	object

Following this change, the dataset's memory footprint goes from 3.3GB to 606MB, hence a reduction of ~82% of the memory used. This saving is very helpful to be able to run the model training on a personal laptop rather than in the cloud.

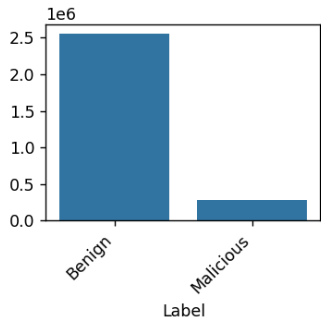
To increase the memory footprint efficiency, we also looked at the Headers bit, wondering if all those data were useful. We first checked if some bit would never change whatever the attack type. The first do a visual inspection by plotting the average bits in the dataset



Finally, we removed all bits with a standard deviation below a threshold (1e^{-10}). We are left with 109 headers bit and 10 entropy values, so 119 data. Unfortunately, this is not a squared number, which would have allowed us to represent the data as a squared image.
To correct this point, we have added a fixed bit (of 0) at the end of the headers and at the end of the entropy, allowing us to create a 11x11 image.

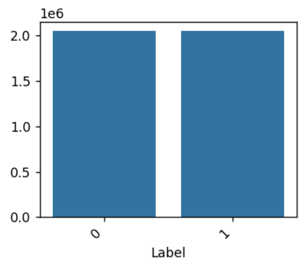
Correcting data imbalances

Even with the most care, Intrusion Detection datasets are heavily imbalanced. The plot below is a representation of this imbalance in our dataset.



To correct for this imbalance, we decided to implement the SMOTE algorithm on the training dataset (not on the test dataset), so we have more data to train our deep learning models.

After the SMOTE algorithm step is completed, our dataset is now balanced:



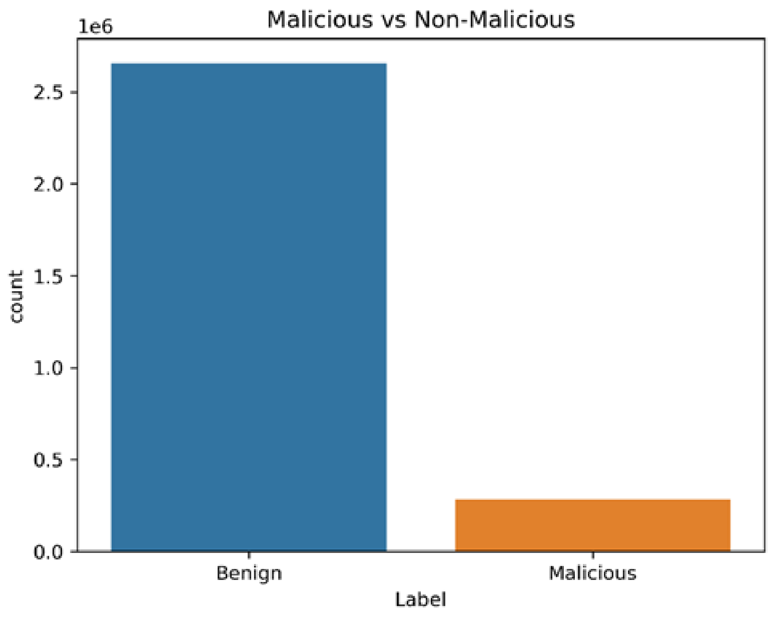
d. Data extracted for the model using NLP (Binary Classification) :

The first phase is dedicated to preparing the data. It begins with parsing .pcap files using PyShark to extract structured HTTP fields such as methods, URLs, headers, and bodies.
These raw payloads are then semantically cleaned and fingerprinted to highlight suspicious patterns (e.g. SQL keywords, encoded content, shell commands).
In parallel, multiple. parquet files -extracted from the same dataset (CIC-IDS2018) and containing both numerical and string columns- are loaded and processed to only keep string columns: PayloadEncoding, PayloadString, Attack and Label which will later be used to extract matching features.
Both datasets are unified into a single field called nlp_text, designed to represent each HTTP request as a clean, semantically meaningful sentence. The combined dataset is exported in .csv and .json formats for further NLP tokenization and training.

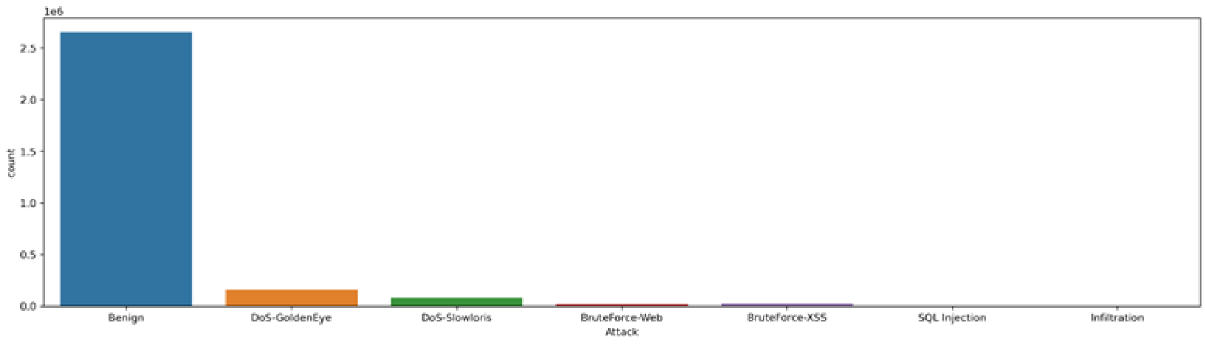
3. Exploratory data analysis

a. Class Distribution Analysis

The initial bar graph highlights a significant class imbalance in the dataset. The number of benign samples is greater than the number of malicious ones. This disparity may impact model performance, particularly in terms of accuracy, recall, and the detection of minority-class events.

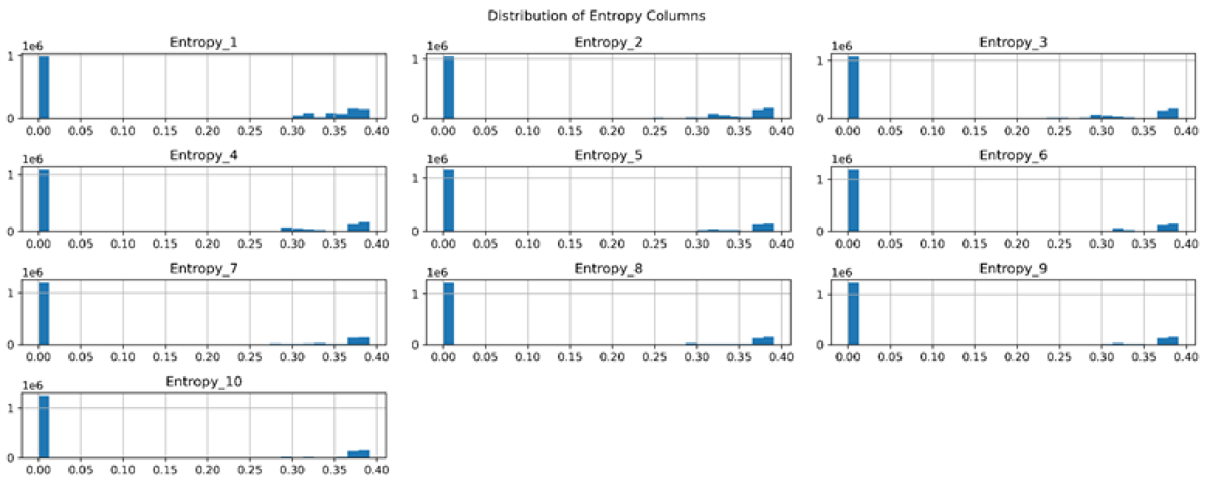


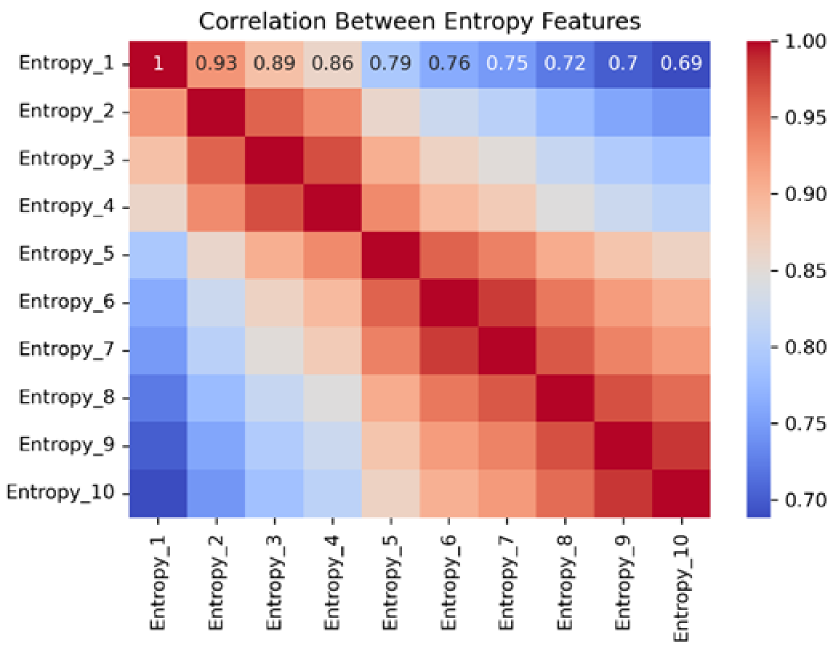
Further analysis of attack types reveals that Denial of Service (DoS) attacks—specifically **DoS-GoldenEye** and **DoS-Slowloris**—dominate the malicious samples. The dataset contains very few instances of **SQL Injection** and **Infiltration** attacks. Overall, malicious payloads constitute only **9.6%** of the entire dataset, reinforcing the imbalance issue.



2. Entropy Feature Correlation Analysis

To understand patterns in the distribution of payload entropy, I evaluated the correlation among entropy features across the entire dataset.



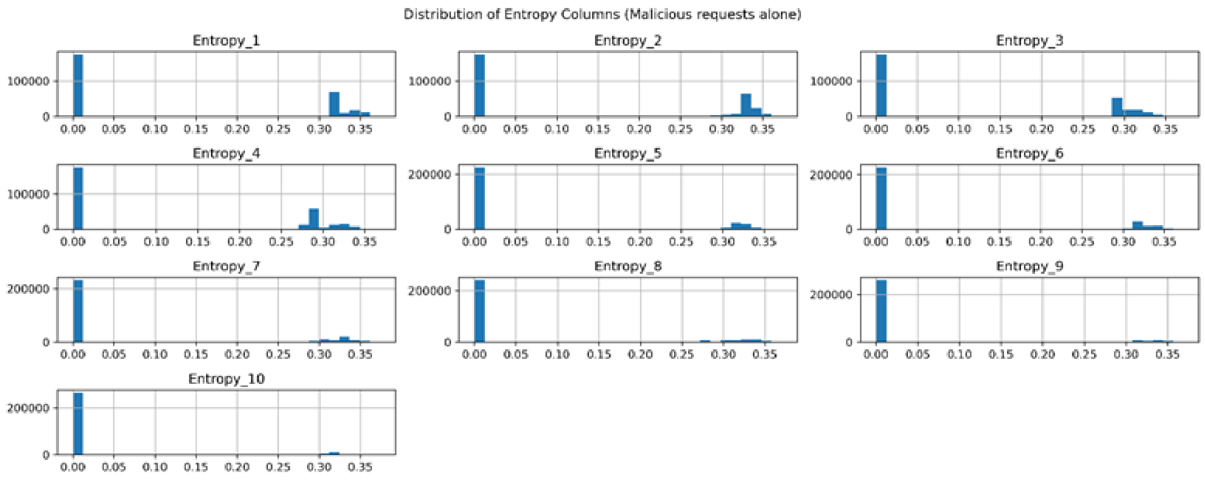


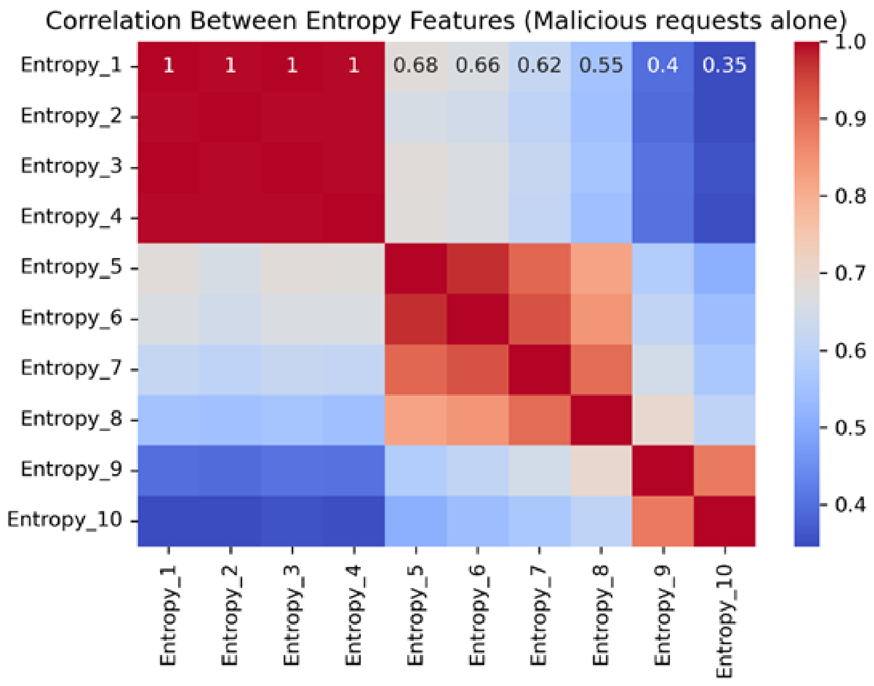
- All entropy-related columns exhibit **positive correlation** with each other.
- The **first 10% and second 10%** of the payload show a moderately strong correlation.
- The **latter segments** of the payload (last 40%) demonstrate **medium to strong correlations** among themselves, suggesting potential redundancy or consistency in entropy structure.

3. Entropy Correlation in Malicious Payloads

Focusing exclusively on malicious samples, I observed the following patterns:

- The **first four entropy segments** show near-perfect correlation. This indicates a strong similarity in the entropy ranges of the first 40% of malicious payloads, implying that malicious data often embeds attack patterns early in the payload.
- The correlation strength gradually decreases across the subsequent entropy segments, suggesting less consistency in entropy distribution in the latter parts of the payload.

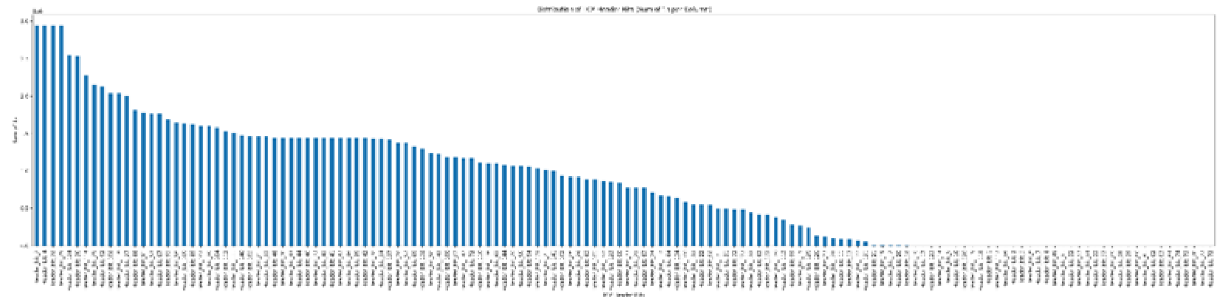




4. Header Bit Analysis

I examined the **Header_Bit** columns to identify patterns specific to malicious traffic.

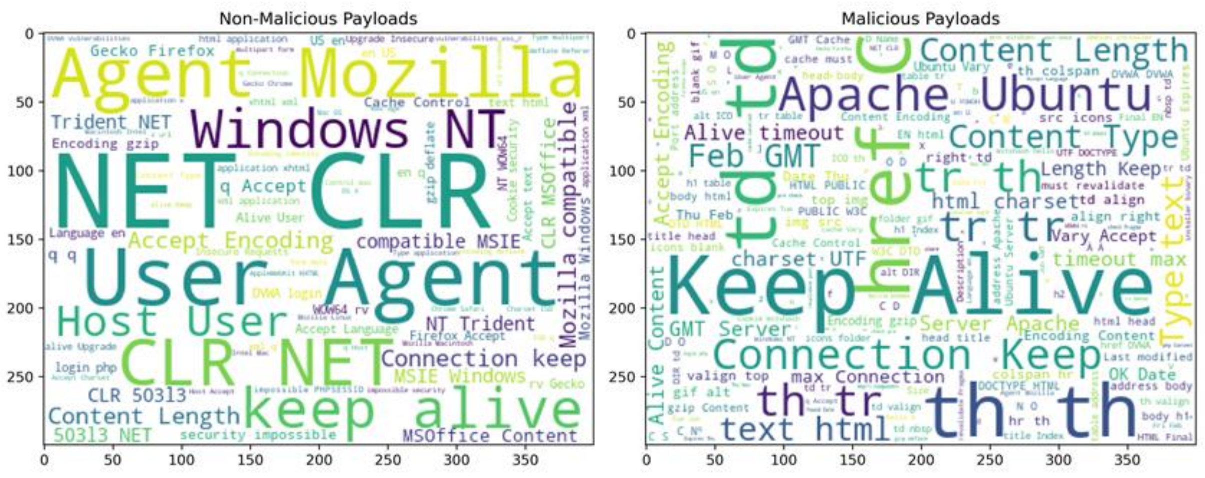
- The bits with the highest activity overall are: **2, 4, 29, 50, 78, 79, 108, 114, and 124**.
- Malicious payloads consistently show activity in specific header bits, indicating that certain bit patterns may be indicative of attack signatures.



5. Word Cloud Analysis

Text-based payload patterns offer further insight into the behavioral differences between benign and malicious samples.

- In **benign payloads**, common terms include **"User-Agent"**, **"NET"**, and **"CLR"**, suggesting normal browser or client software signatures.
- In **malicious payloads**, references to "User-Agent" are largely absent. Instead, the payloads include server-related identifiers such as **"Apache"** and **"Ubuntu"**, which may point to automated scripts or server-side attacks.

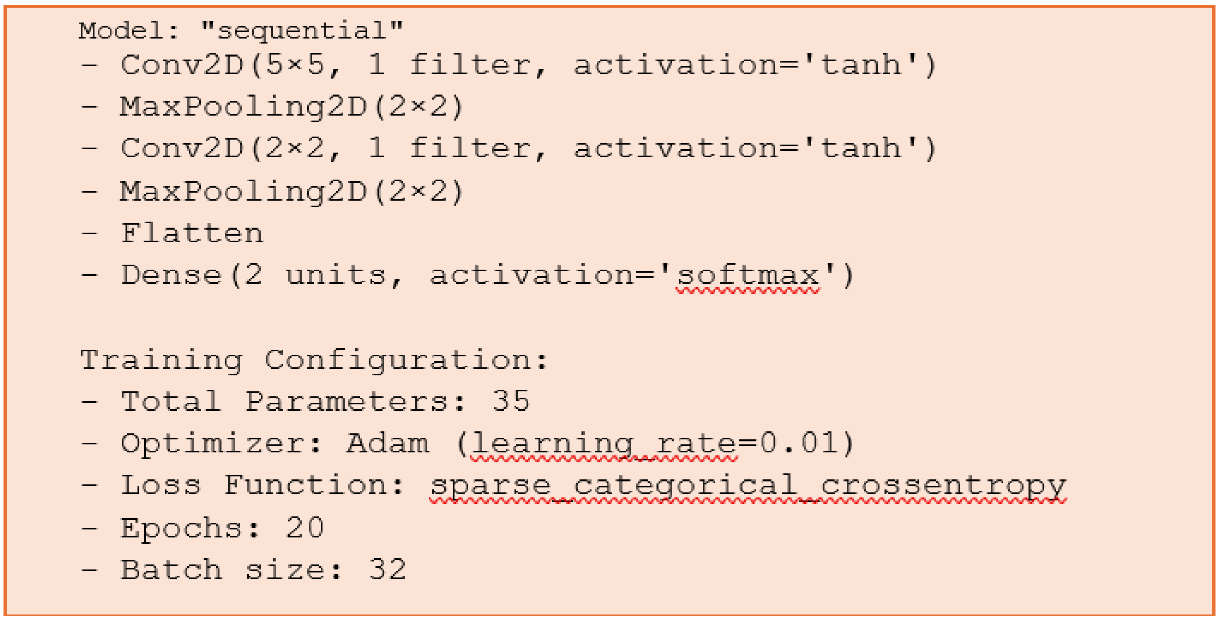


II. Intrusion detection using Computer Vision

1. Simple model

We implemented a lightweight Binary IDS approach inspired by the research paper "Intrusion Detection using TCP/IP Single Packet Header Binary Image". In this method, only the first 144 header bits of each packet are used to create 12×12 binary images, which are then fed into a simple CNN architecture with only 35 trainable parameters.

a. Architecture

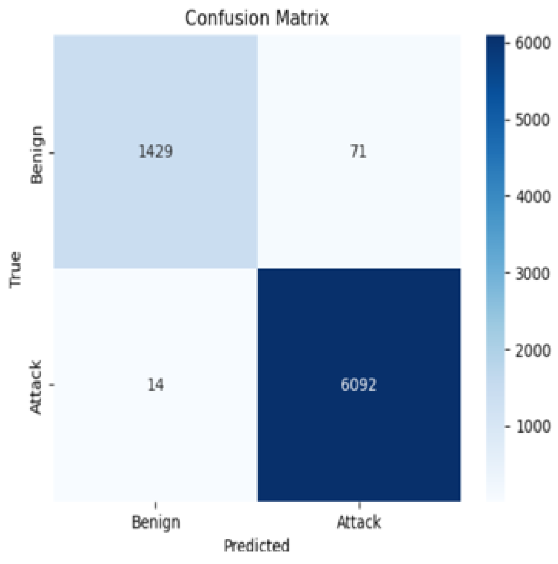


```
Model: "sequential"
- Conv2D(5×5, 1 filter, activation='tanh')
- MaxPooling2D(2×2)
- Conv2D(2×2, 1 filter, activation='tanh')
- MaxPooling2D(2×2)
- Flatten
- Dense(2 units, activation='softmax')

Training Configuration:
- Total Parameters: 35
- Optimizer: Adam (learning_rate=0.01)
- Loss Function: sparse_categorical_crossentropy
- Epochs: 20
- Batch size: 32
```

TensorFlow model architecture

a. Results after 20 epochs



Accuracy	99%
Error Rate	2.88%
Precision	99%
Recall	95%
F1 score	97%

This minimal model achieved strong classification performance (~99% accuracy) while maintaining very low computational cost, demonstrating that accurate detection can be achieved using header-only information.

2. Simple model

We have decided after to keep the simple model, but to add more data to it. Indeed, we thought we could add the entropy of the payload as part of the data. Also, we wanted to create a bigger dataset, so we increased the training set using SMOTE.

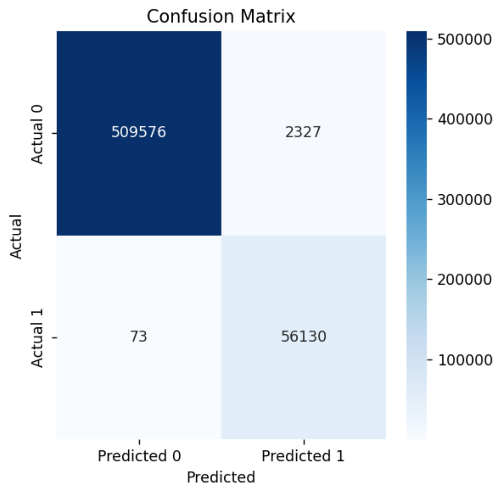
We implemented the following architecture:

a. Architecture

```
(model): Sequential(
   (0): Sequential(
            (0): Conv2d(1, 1, kernel_size=(5, 5), stride=(1, 1), padding=(1, 1))
            (1): Tanh()
            (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1)
        )
   (1): Sequential(
           (0): Conv2d(1, 1, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
           (1): Tanh()
        (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ...)
      )
  (2): Flatten(start_dim=1, end_dim=-1)
  (3): Linear(in_features=4, out_features=1, bias=True)
```

Pytorch model architecture

b. Results after 7 epochs



Accuracy	99.58%
Error Rate	9.88%
Precision	99.87%
Recall	96.02%
F1 score	97.91%

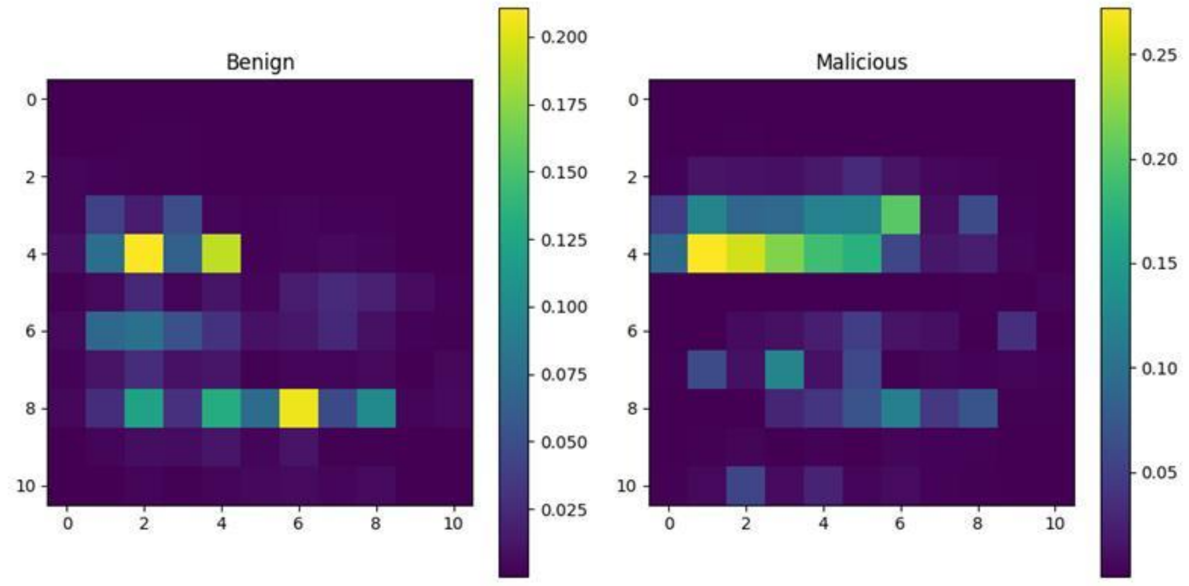
The results are quite good, but we believe we can still improve the model.

c. Attention map

To go deeper in our understanding of the different models, we have implemented an attention map:

Grad-CAM (Gradient-weighted Class Activation Mapping) is a visualization technique for interpreting convolutional neural networks (CNNs). It highlights the regions of an input image that were most influential for the model's prediction by using gradients flowing into the final convolutional layer.

After implementing this, we could run it on our test set to see where the model focuses its attention.



Attention map of the simple model

3. BinaryIDSModel Complex

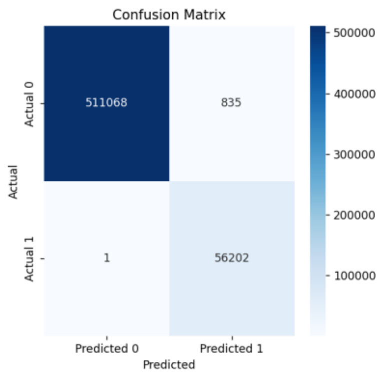
Finally, we implemented a more complex model to add more capabilities of detection.

a. Architecture

```
(model): Sequential(
   (0): Sequential(
         (0): Conv2d(1, 16, kernel_size=(5, 5), stride=(1, 1), padding=(1, 1))
         (1): ReLU()
         (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1)
         )
   (1): Sequential(
        (0): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU()
        (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1)
         )
   (2): Flatten(start_dim=1, end_dim=-1)
   (3): Dropout(p=0.2, inplace=False)
   (4): Linear(in_features=128, out_features=64, bias=True)
   (5): ReLU()
   (6): Linear(in_features=64, out_features=1, bias=True)
 )
```

Pytorch model architecture

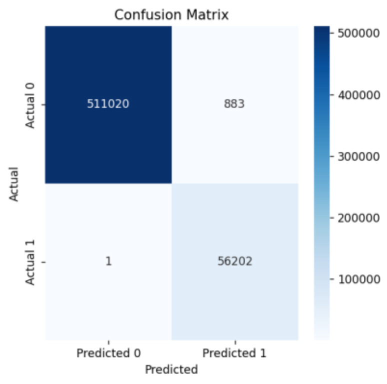
b. Results



Accuracy	99.41%
Error Rate	9.87%
Precision	99.75%
Recall	94.58%
F1 score	97.10%

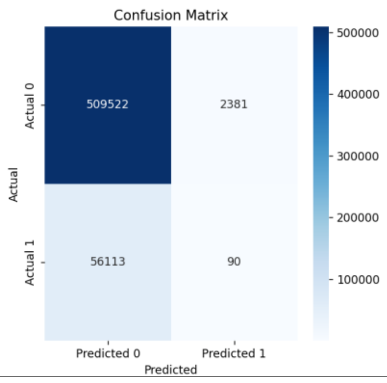
Our model has an overall extremely good performance. Moreover, the errors appear where it less matters: indeed, it is better to classify a packet as suspicious when it is not. The contrary is problematic, as some attacks would go undetected.

Moreover, the convergence of the model is extremely quick. We can plot the same confusion matrix for the test dataset after 1 epoch:



Accuracy	99.84%
Error Rate	9.89%
Precision	100.00%
Recall	98.45%
F1 score	99.22%

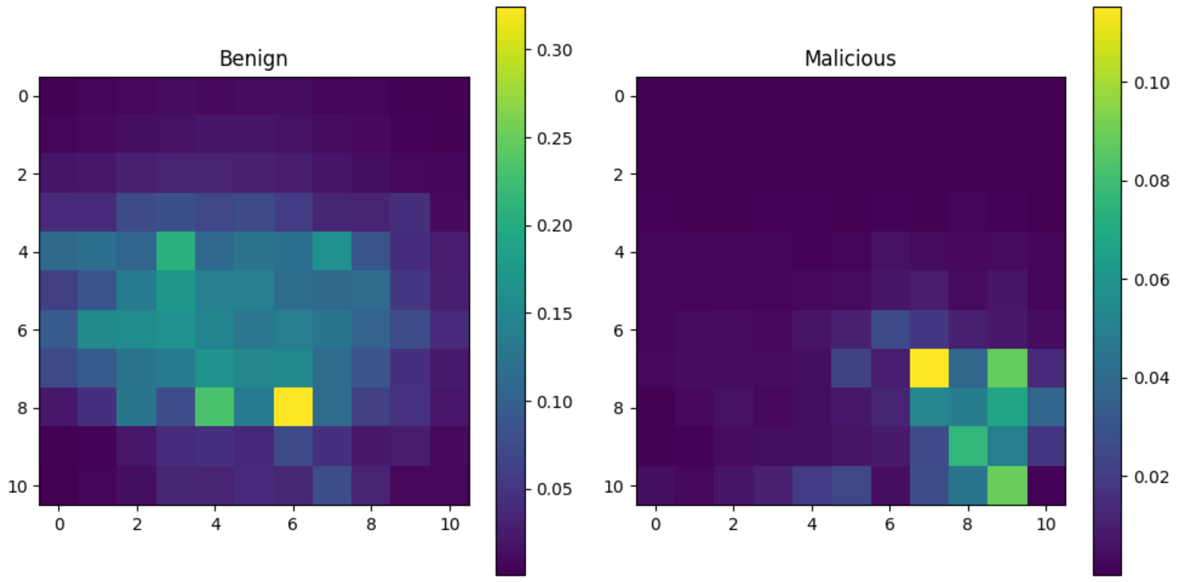
The user could argue this is a leakage of previous calibration. To make sure this is not the case, models are initialized with random weights before any training. We can check that a model with random weights does not have good performance:



Accuracy	89.70%
Error Rate	0.02%
Precision	0.16%
Recall	3.64%
F1 score	0.31%

c. Attention map

Once again, we are interested in understanding where the model focuses its attention. We can use the attention map for this purpose.



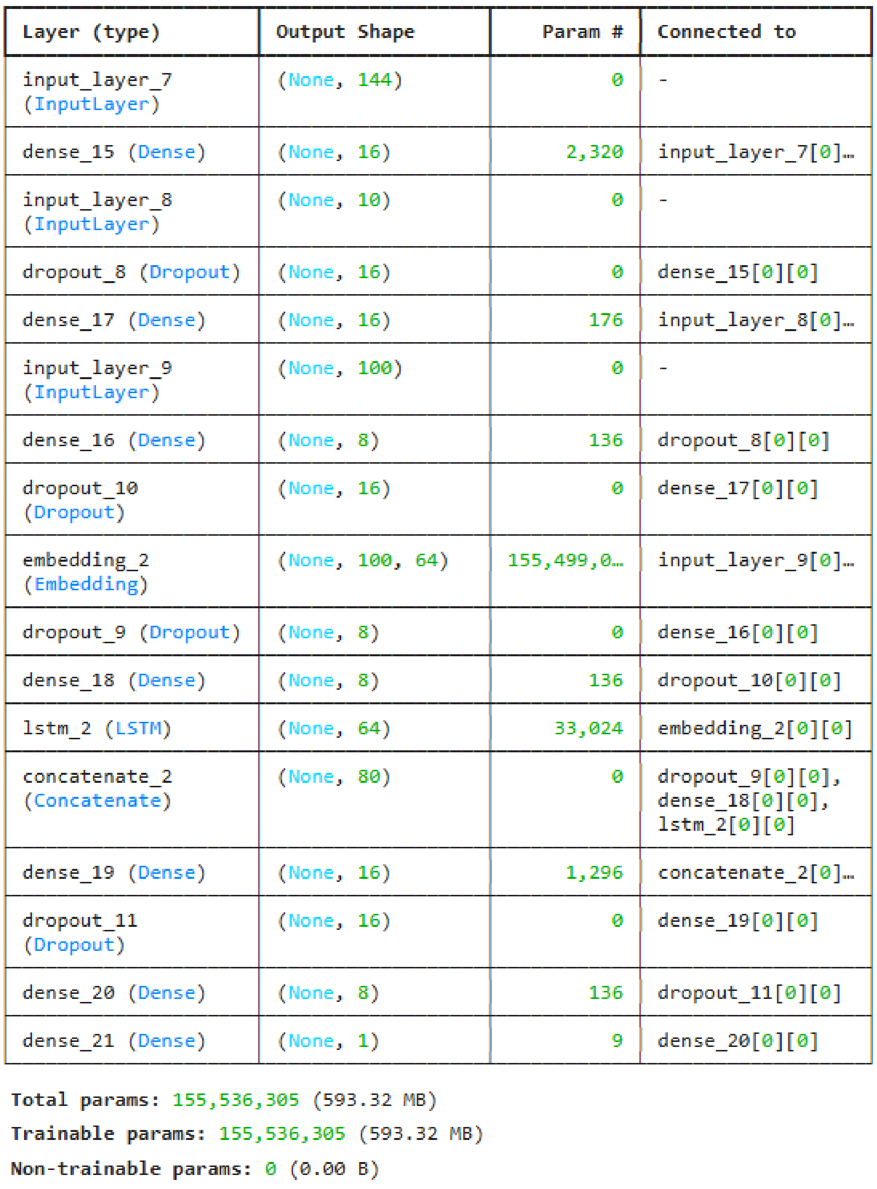
Attention map of the BinaryIDSModel

If we compare with the previous attention map, we can clearly see that the model identifies 2 distinct zones to focus on, to decide if a package is malicious or benign.

III. Intrusion detection using NLP based multi-input architecture for Binary Classification (Benign VS Malicious)

Git Hub Repository: https://github.com/susmitha-ann/DeepLearning/tree/main

a. Architecture

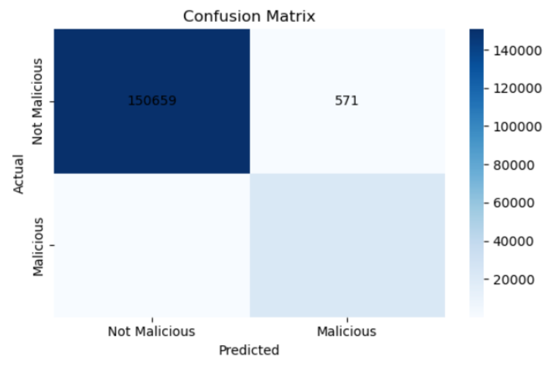


```
┌─────────────────────┬───────────────────┬────────────┬───────────────────┐
│ Layer (type)        │ Output Shape      │    Param # │ Connected to      │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ input_layer_7       │ (None, 144)       │          0 │ -                 │
│ (InputLayer)        │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dense_15 (Dense)    │ (None, 16)        │      2,320 │ input_layer_7[0]… │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ input_layer_8       │ (None, 10)        │          0 │ -                 │
│ (InputLayer)        │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dropout_8 (Dropout) │ (None, 16)        │          0 │ dense_15[0][0]    │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dense_17 (Dense)    │ (None, 16)        │        176 │ input_layer_8[0]… │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ input_layer_9       │ (None, 100)       │          0 │ -                 │
│ (InputLayer)        │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dense_16 (Dense)    │ (None, 8)         │        136 │ dropout_8[0][0]   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dropout_10          │ (None, 16)        │          0 │ dense_17[0][0]    │
│ (Dropout)           │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ embedding_2         │ (None, 100, 64)   │ 155,499,0… │ input_layer_9[0]… │
│ (Embedding)         │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dropout_9 (Dropout) │ (None, 8)         │          0 │ dense_16[0][0]    │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dense_18 (Dense)    │ (None, 8)         │        136 │ dropout_10[0][0]  │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ lstm_2 (LSTM)       │ (None, 64)        │     33,024 │ embedding_2[0][0] │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ concatenate_2       │ (None, 80)        │          0 │ dropout_9[0][0],  │
│ (Concatenate)       │                   │            │ dense_18[0][0],   │
│                     │                   │            │ lstm_2[0][0]      │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dense_19 (Dense)    │ (None, 16)        │      1,296 │ concatenate_2[0]… │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dropout_11          │ (None, 16)        │          0 │ dense_19[0][0]    │
│ (Dropout)           │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dense_20 (Dense)    │ (None, 8)         │        136 │ dropout_11[0][0]  │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dense_21 (Dense)    │ (None, 1)         │          9 │ dense_20[0][0]    │
└─────────────────────┴───────────────────┴────────────┴───────────────────┘
 Total params: 155,536,305 (593.32 MB)
 Trainable params: 155,536,305 (593.32 MB)
 Non-trainable params: 0 (0.00 B)
```

Why a multi-input neural network model was considered:

1. We are working with 3 heterogeneous input types: structured, numerical, and text. A single model can't handle all 3 well unless you separate inputs.
2. A simple model (like logistic regression or one big Dense net) might overfit fast or miss nuances across modalities.
3. Multi-branch architectures are common in cybersecurity ML; e.g., detecting phishing, botnet traffic, malware signatures where multiple signals are fused.

b. **Results**



Accuracy	98.83%
Precision	98%
Recall	99%
F1 score	98%

IV. Intrusion detection using NLP for Binary Classification (Benign VS Malicious)

Once the textual dataset was prepared, it was tokenized using Hugging Face's DistilBERT tokenizer, converting each nlp_text into input IDs and attention masks suitable for transformer models.

The final classification task was framed as binary: distinguishing between benign and malicious HTTP requests.

Due to a strong class imbalance (with ~80% malicious samples), class weighting was applied to mitigate bias during training, using a custom model.

The model architecture used was DistilBERTForSequenceClassification, fine-tuned using standard binary cross-entropy loss.

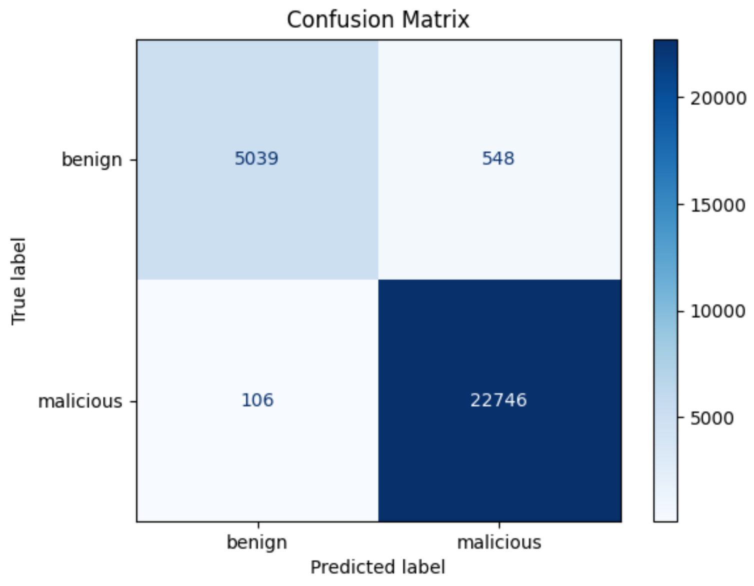
Evaluation metrics included accuracy, precision, recall, F1-score, and ROC-AUC.

Visual diagnostics such as confusion matrix and ROC curve were generated. The model achieved strong performance, reaching 98% accuracy, and was particularly effective at detecting malicious traffic with minimal false negatives an essential quality in the IDS context.

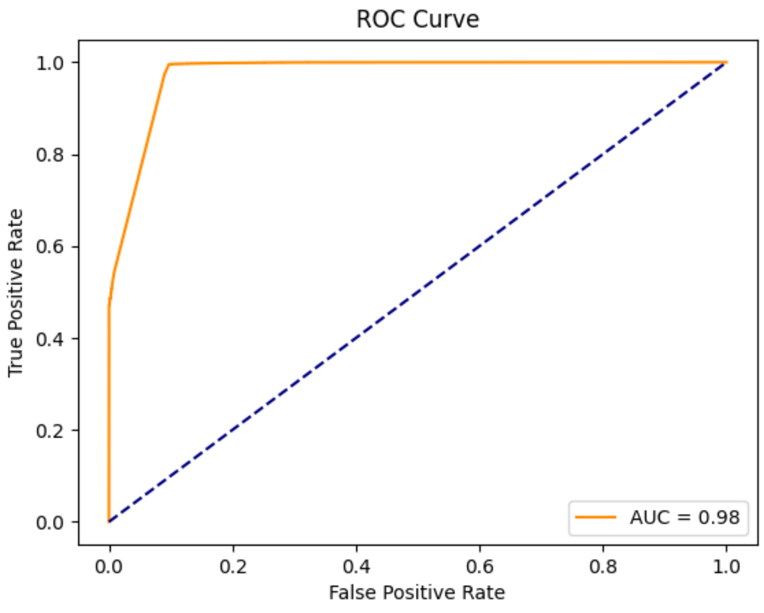
Outputs, visualizations, and generated files are all stored within the project structure. More detailed implementation steps can be found in the README.md file of the ***Github repository*** :

https://github.com/wafa26/nlp-ids-binary-classifier

- **Confusion Matrix :**



- **ROC Curve :**



Git Repository Link

GitHub - gpealat/DSTI_Deeplearning: Holding repository for all sub projects