Masters in Cybersecurity

Machine Learning Applied to Security

Final Project

João Luís - 107403

Diogo Almeida - 108902

January 14, 2025

Contents

1 Introduction **2**

2 Dataset Analysis **3**

2.1 Dataset Structure 3

2.2 Feature Analysis 5

2.3 Dataset Split 11

3 Architecture Design **15**

4 Models Comparison **19**

5 Grafana and Results **22**

1 Introduction

Machine learning has emerged as a powerful tool in the field of cybersecurity, offering new capabilities for detecting and preventing network-based threats. This project explores the application of machine learning techniques to network security, specifically focusing on the real-time classification of network packets as either benign or malicious.

The core of our approach lies in its dynamic learning capability. Rather than relying solely on static, pre-trained models, we've developed a system that continuously learns and adapts to new network traffic patterns. Our architecture combines real-time packet analysis with automated model retraining, leveraging the Stochastic Gradient Descent (SGD) algorithm's partial fit functionality to enable incremental learning without complete model reconstruction.

The project encompasses several key components: a comprehensive analysis of network traffic data, the development of a modular containerized architecture for real-time processing, and the implementation of a visualization system using Grafana for monitoring model performance.

Final results demonstrate the effectiveness of this approach, with the model's accuracy improving from 77% to approximately 96% through continuous learning. This significant improvement highlights the value of adaptive machine learning systems in modern cybersecurity applications, where threat patterns constantly evolve and static detection methods may quickly become outdated.

2 Dataset Analysis

The dataset selected for model training and internet packet flow simulation initially consisted of 23,153 instances, each containing 23 attributes, thereby offering a substantial amount of data for model training.

2.1 Dataset Structure

Each database entry is composed of the following attributes:

- **ts:** Package timestamp
- **uid:** Unique identifier for the package
- **id.orig_h:** Source IP address
- **id.orig_p:** Source port
- **id.resp_h:** Receiver IP address
- **id.resp_p:** Receiver port
- **proto:** Transport protocol
- **service:** Type of application service
- **duration:** Connection duration
- **orig_bytes:** Total bytes sent by the source
- **resp_bytes:** Total bytes answered by the receiver
- **conn_state:** End status of the communication session
- **local_orig:** Indicates whether the source host of a connection belongs to the local network
- **local_resp:** Indicates whether the receiver of a connection belongs to the local network
- **missed_bytes:** Indicates how many bytes were lost during packet capture
- **history:** Sequence of state transitions during a connection
- **orig_pkts:** Number of original packets sent or received on a connection
- **orig_ip_bytes:** Total amount of data (in bytes) sent by the origin of the connection
- **resp_pkts:** Number of packets sent or received by the receiver on a connection

- **resp_ip_bytes:** Total amount of data (in bytes) sent by the receiver of the connection
- **tunnel_parents:** Which tunnels or encapsulation connections are involved in the communication
- **label:** Indicates whether the packet is malicious or benign
- **detailed-label:** Indicates the type of malicious packet

An example of the dataset can be seen below:

	ts	uid	id.orig_h	id.orig_p	id.resp_h	id.resp_p	proto	...	orig_pkts	orig_ip_bytes	resp_pkts	resp_ip_bytes	tunnel_parents	label	detailed-label
0	1.545404e+09	CrDn63WjJEmrWGjqf	192.168.1.195	41040	185.244.25.235	80	tcp	...	3	180	0	0	-	Benign	-
1	1.545404e+09	CY9lJW3gh1Eje4usP6	192.168.1.195	41040	185.244.25.235	80	tcp	...	1	60	0	0	-	Benign	-
2	1.545404e+09	CcFXLynukEDnUlvgl	192.168.1.195	41040	185.244.25.235	80	tcp	...	1	60	0	0	-	Benign	-
3	1.545404e+09	CDrkrSobGYxHhYfth	192.168.1.195	41040	185.244.25.235	80	tcp	...	94	5525	96	139044	-	Benign	-
4	1.545404e+09	CTWZQf2oJSvq6zmPAc	192.168.1.195	41042	185.244.25.235	80	tcp	...	3	180	0	0	-	Benign	-

Figure 1: Dataset

2.2 Feature Analysis

Initially, the content of each feature was examined to identify those with missing or constant values across all entries, which were subsequently removed. The following features were removed from the dataset: ts, as it was not relevant; uid, as it served solely as a unique identifier; local_orig and local_resp, as they contained no values; and tunnel_parents, as all entries had identical values.

For the duration feature, which exhibited significant dispersion in its values, we first handled the missing entries by imputing them with the median of the available data. The choice of the median was made to prevent the influence of extreme values, which could distort the imputation. After addressing the missing values, we applied the **StandardScaler** to the duration feature. This transformation normalized the feature, ensuring that the data followed a standard normal distribution with a mean of 0 and a standard deviation of 1. The standardization process helped to reduce the impact of the varying scales and made the feature more suitable for machine learning algorithms, which typically perform better when the data is normalized.

Subsequently, features without numerical values were identified and encoded using **OneHotEncoder**. This method converts each unique category within a feature into a separate binary column (0 or 1), with a maximum of seven possible categories per feature.

Next, to assess the importance of each feature, box plots were employed. These plots provide a visual representation of the distribution of each feature, allowing for an examination of how the data is spread. This technique is useful for identifying the central tendency and variability of the data, and it also facilitates the detection of potential outliers. Additionally, box plots offer insights into the underlying distribution of the data, helping to determine whether the features follow a normal distribution.

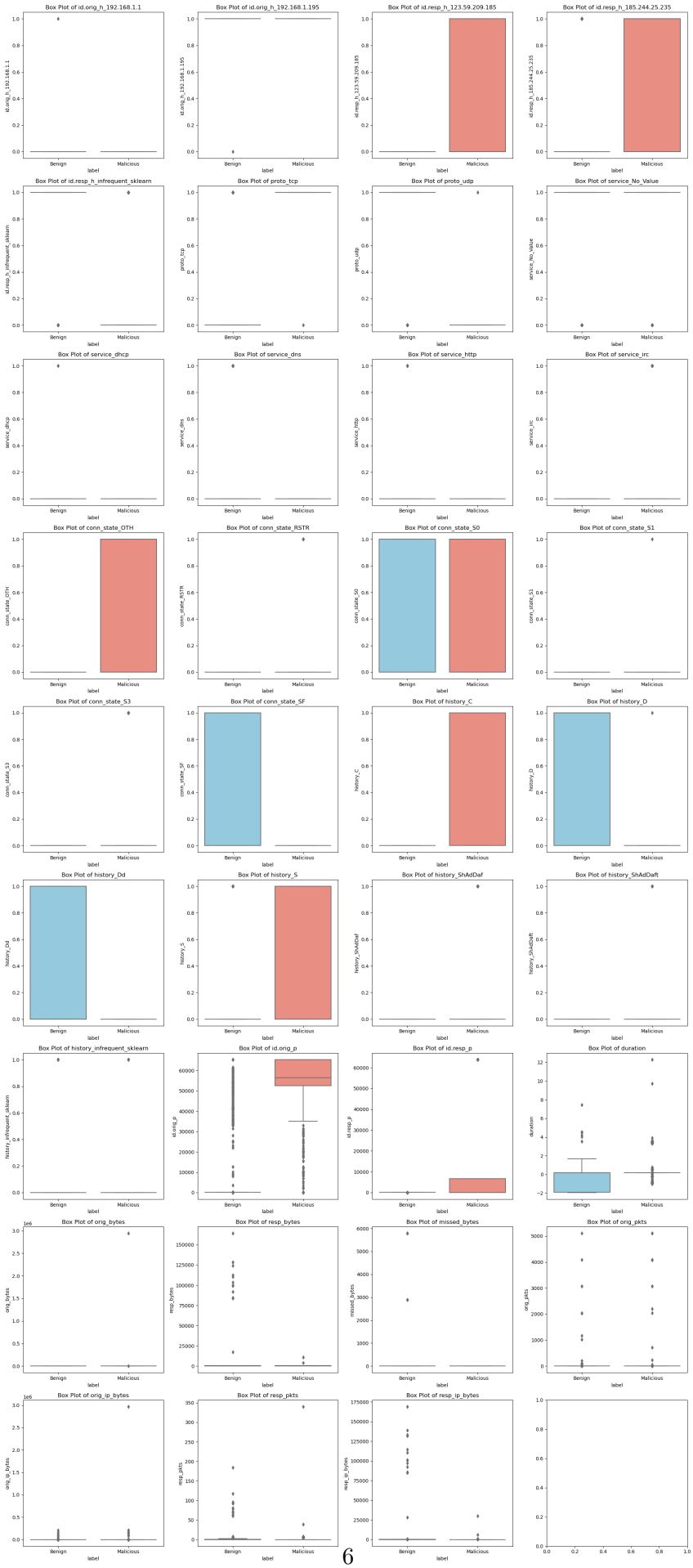

Figure 2: Box Plotes

Certain features exhibit distinct differences between the entries. For example, the id.orig_p and id.resp_p features display greater dispersion or concentration within specific ranges for malicious traffic. In the case of the duration feature, malicious entries are more dispersed, whereas benign entries tend to cluster more closely. These variables, therefore, may hold significant potential for differentiating between benign and malicious traffic. On the other hand, variables such as proto, service, and certain connection states (conn_state) show minimal or no meaningful variation between the two classes, suggesting that they may have limited value for class separation. Additionally, some features, such as history_C and history_S, are binary or exhibit clearly separated categories, making them directly applicable for classification models.

To better understand the distribution of values for each variable, histograms were generated for each feature of the dataset. These visualizations help to reveal whether the values are concentrated around a central point or if they follow a uniform distribution. If the histogram displays isolated bars at the extremes, far from the central mass of the data, it suggests the presence of outliers or anomalous data points that may require special handling.

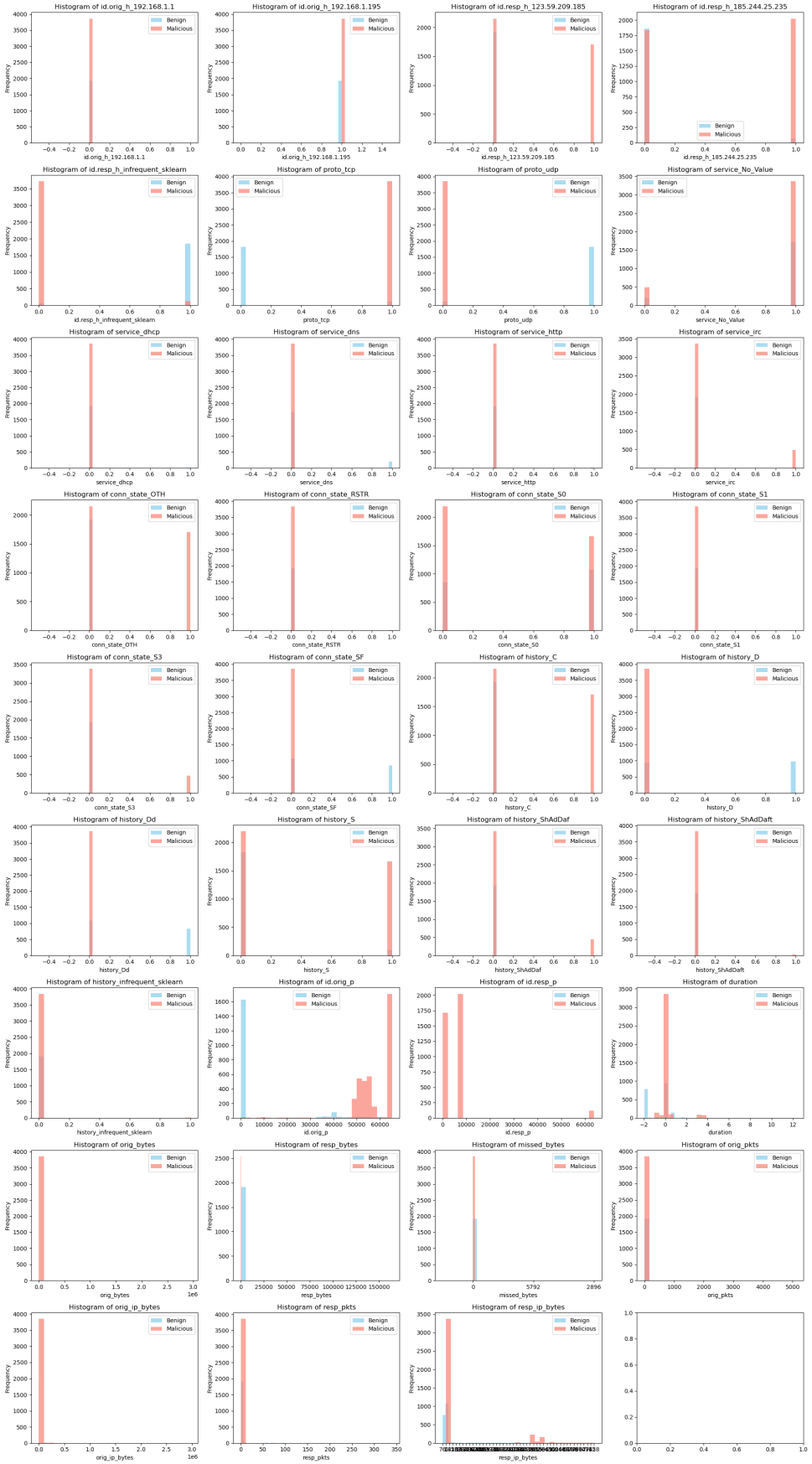

Figure 3: Histograms

Variables such as id.orig_p, id.resp_p, and duration exhibit distinct differences in the distribution of values between the labels. Specifically, id.orig_p shows a concentration of values within a certain range for Malicious traffic, while Benign traffic displays more dispersion across different values. Similarly, the duration feature reveals that Benign traffic tends to have values concentrated around lower ranges, whereas Malicious traffic demonstrates a more dispersed distribution. These characteristics suggest that these variables may serve as valuable indicators for differentiating between benign and malicious traffic.

Variables such as orig_bytes, resp_bytes, and missed_bytes exhibit extreme values, indicative of outliers. These outliers are potentially significant, as they may highlight anomalous patterns that are characteristic of malicious traffic. Identifying and analyzing these extreme values can be crucial for detecting unusual behavior and differentiating malicious activities from benign traffic.

After individually analyzing each feature, a correlation analysis was conducted between the features using a correlation matrix. Each value in the matrix represents the degree of correlation between two features, typically measured using **Pearson's correlation coefficient**, which ranges from -1 to 1. A value of -1 indicates a perfect negative correlation (as one feature increases, the other decreases proportionally), while a value of 1 indicates a perfect positive correlation (as one feature increases, the other increases proportionally). This analysis provides insights into the relationships between the features and helps identify potential redundancies or dependencies within the dataset.

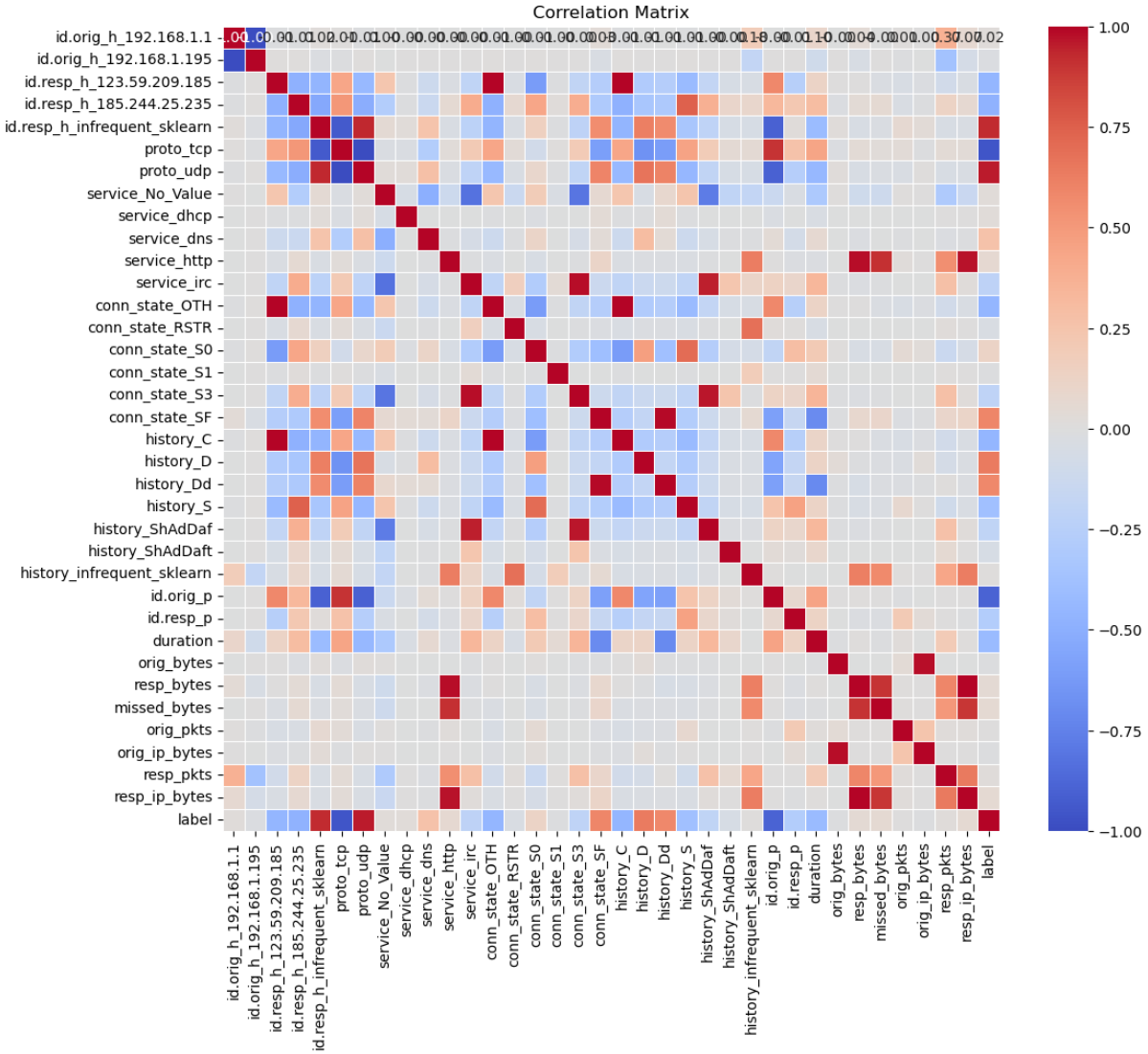

Figure 4: Correlation Matrix

Variables that exhibit a strong correlation (either positive or negative) with the label are considered potential candidates for further analysis or as inputs for predictive modeling. For instance, id.resp_p, duration, and resp_bytes appear to show some degree of correlation with the label, suggesting their relevance in distinguishing between classes. Conversely, high correlations among independent variables may indicate multicollinearity, which can lead to redundancy in predictive models. For example, orig_bytes and resp_bytes seem to be strongly correlated, highlighting the need to address potential multicollinearity in the feature set.

2.3 Dataset Split

Prior to splitting the dataset for training and testing the model, an analysis of the distribution of data for each label was conducted. This step was crucial to ensure that both the benign and malicious classes were adequately represented, preventing any potential bias that could arise from an imbalanced dataset.

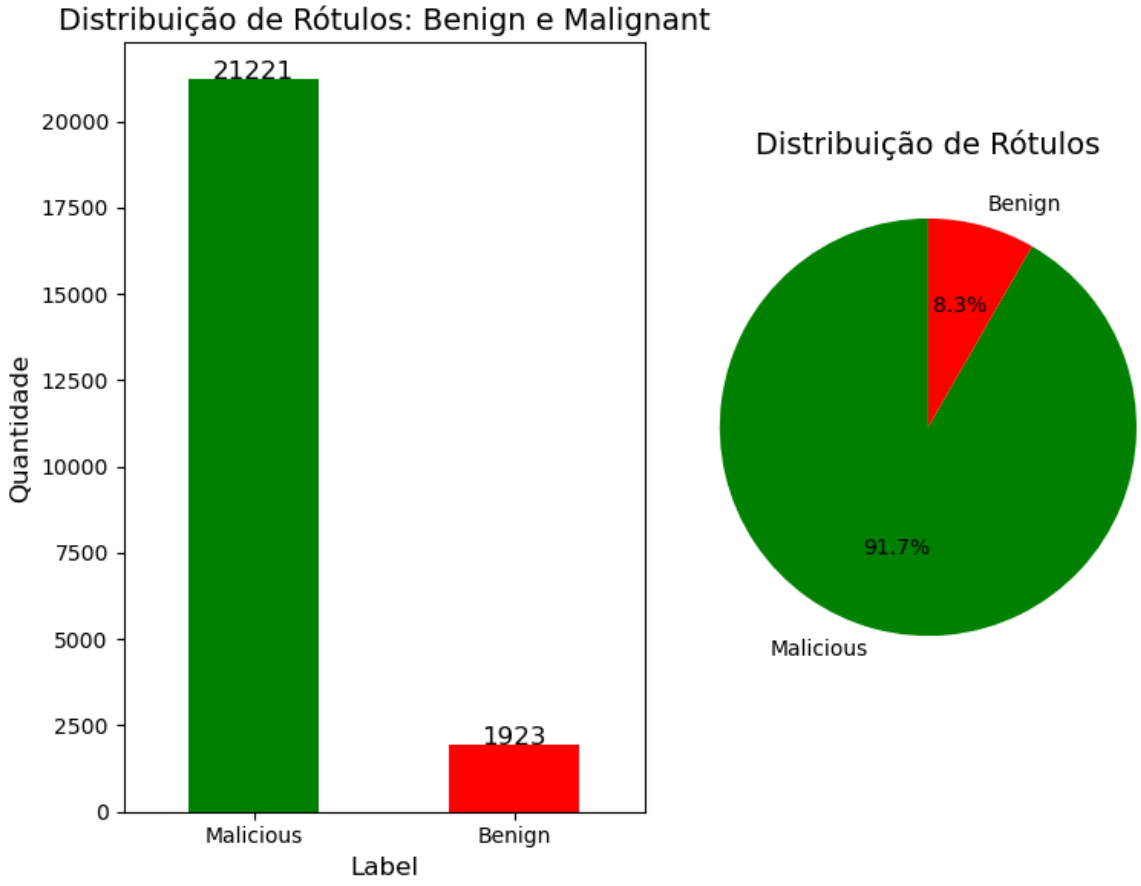

Figure 5: Inicial Distribution

As observed, the initial distribution of the dataset is highly imbalanced, with 91.7% of the data labeled as malignant and only 8.3% labeled as benign.

Additionally, a detailed analysis was conducted on the types of malignant data to understand the distribution of different categories within the malicious traffic. This analysis provides insights into the specific characteristics of the malicious data, which can inform further steps in model training and evaluation.

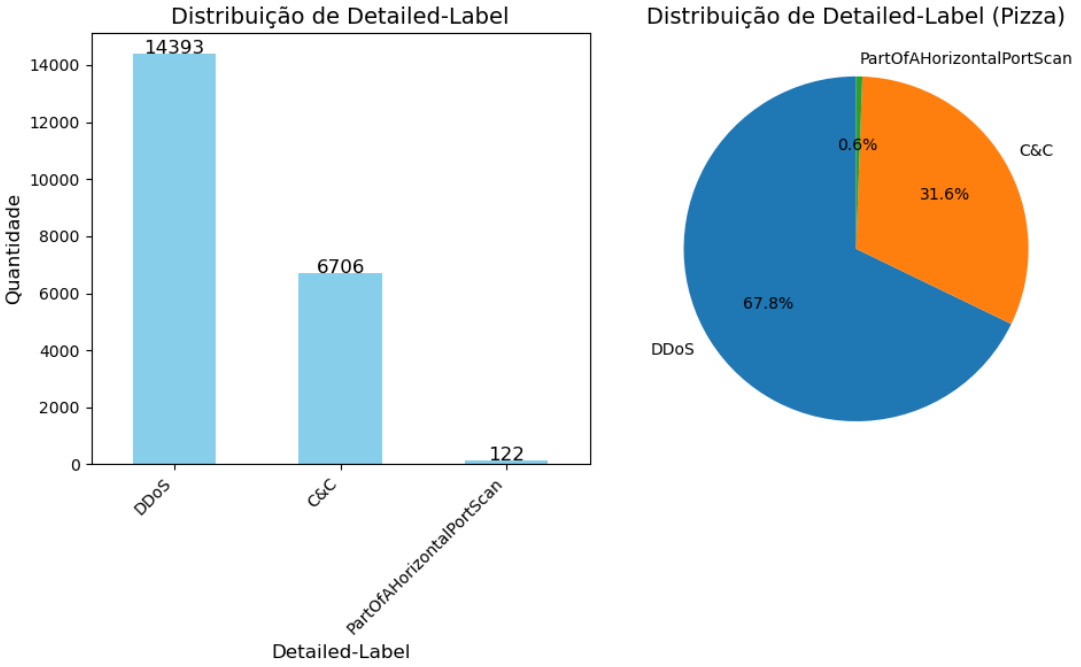

Figure 6: Inicial Malicious Distribution

As shown, 67.8% of the malicious data consists of DDoS packets. Given the high similarity of DDoS packets, the first step was to reduce their representation by 60%. This adjustment resulted in the following distribution of the data:

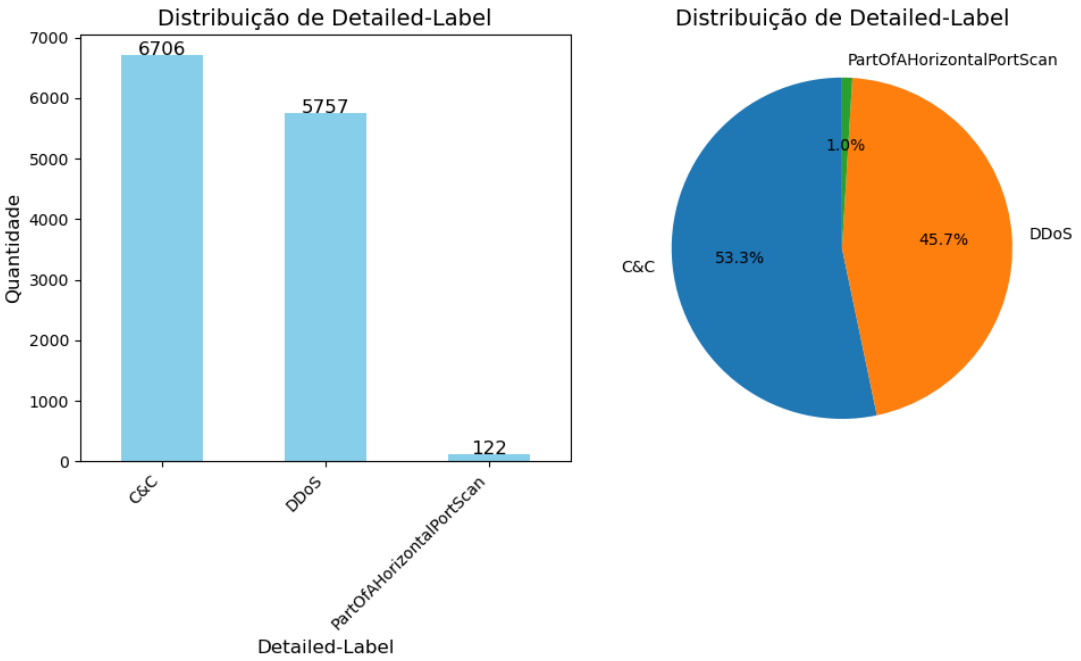

Figure 7: Balanced DDoS

To further balance the dataset, 70% of the malicious data was removed. However, the malicious data labeled as "PartOfAHorizontalPortScan" was retained, as it represented a small portion of the data. The resulting distribution of the labels was as follows:

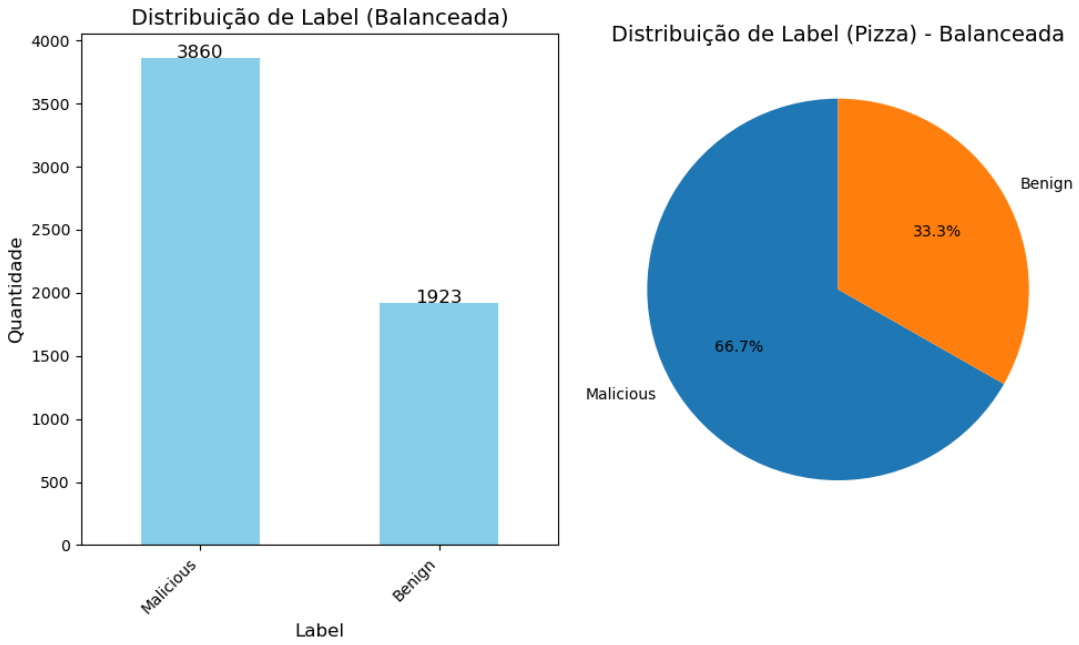

Figure 8: Balanced Distribution

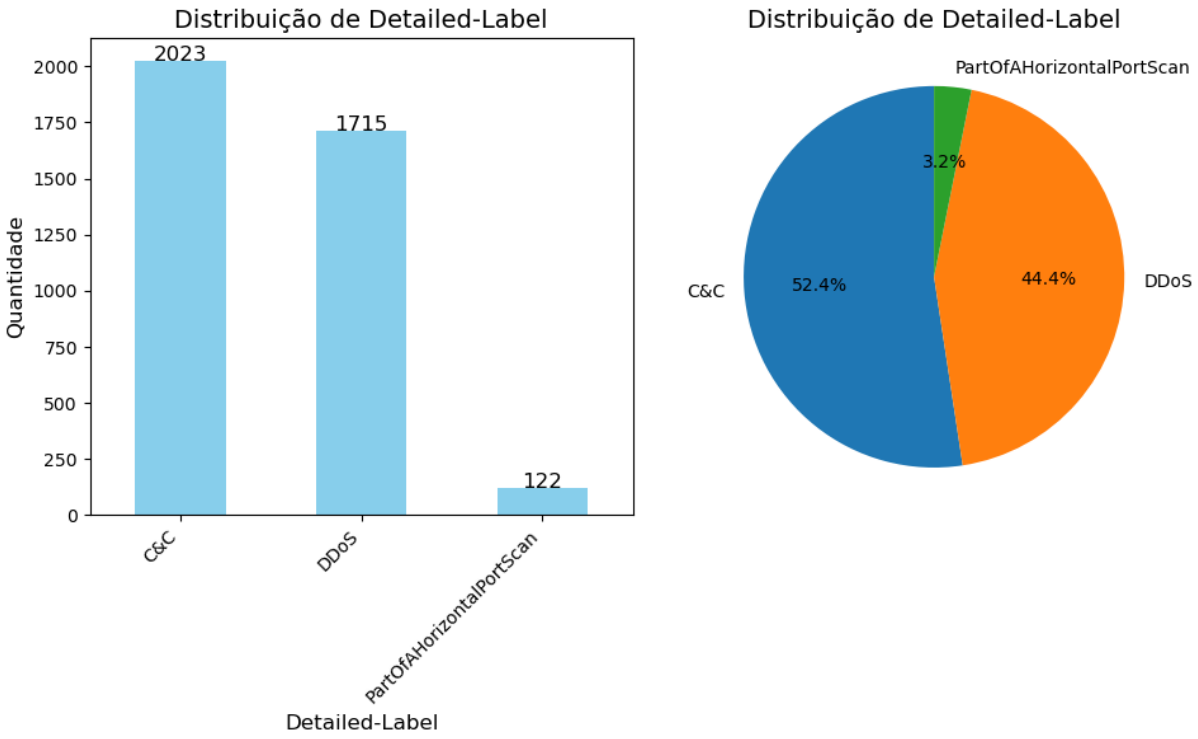

Figure 9: Balanced Malicious Distribution

The final distribution reveals that 66.7% (3860) of the data is labeled as Malicious, while 33.3% (1923) is labeled as Benign. With this improved balance, the dataset is now sufficiently balanced to be split into training and testing subsets.

The dataset was then split for training. A total of 10% of the data was allocated for training the model. This lower percentage, compared to the standard practice of using 70-80% for training, was chosen because the **primary goal** of the project is to **allow the model to be retrained** using the test data from the real-time packet flow.

After selecting 10% of the dataset for training, exactly 700 malicious samples and 500 benign samples were removed from the remaining 90%. These samples were saved in a separate CSV file, which will be used for consistent evaluation of the model's retraining performance over time, ensuring that metrics are always calculated using the same data points. The remaining 90% of the dataset was then reserved for **simulating real-time packet flow** and facilitating future retraining of the model.

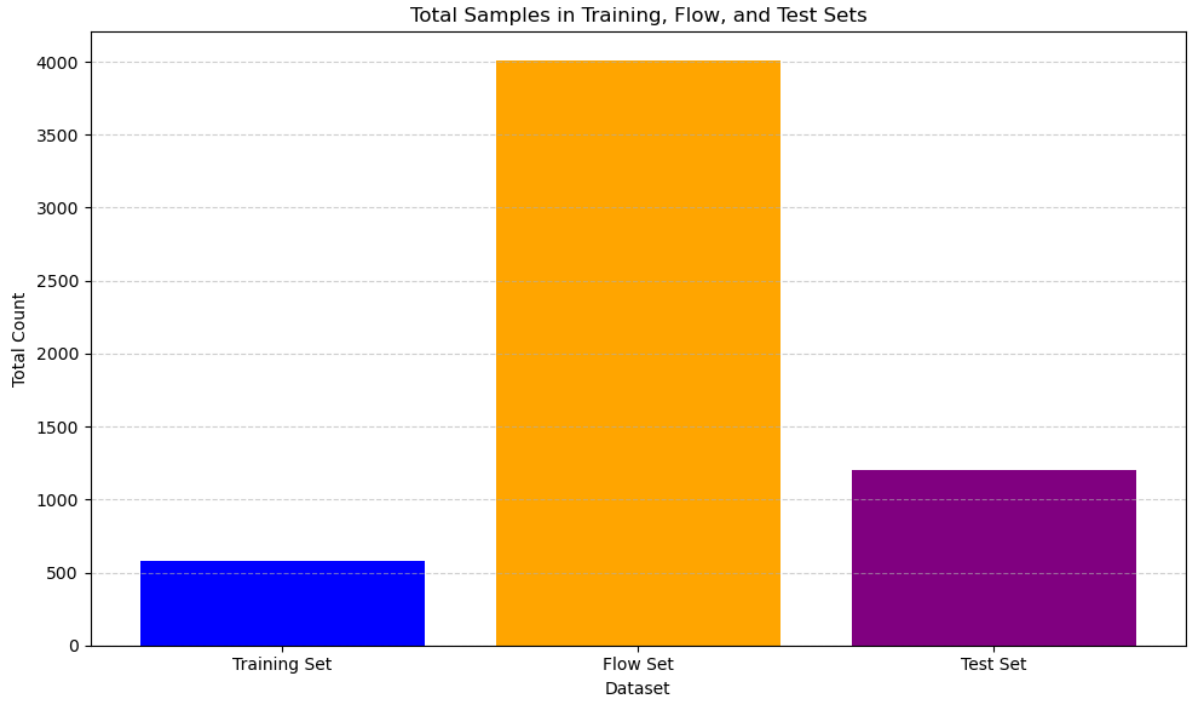

Figure 10: Dataset Split

3 Architecture Design

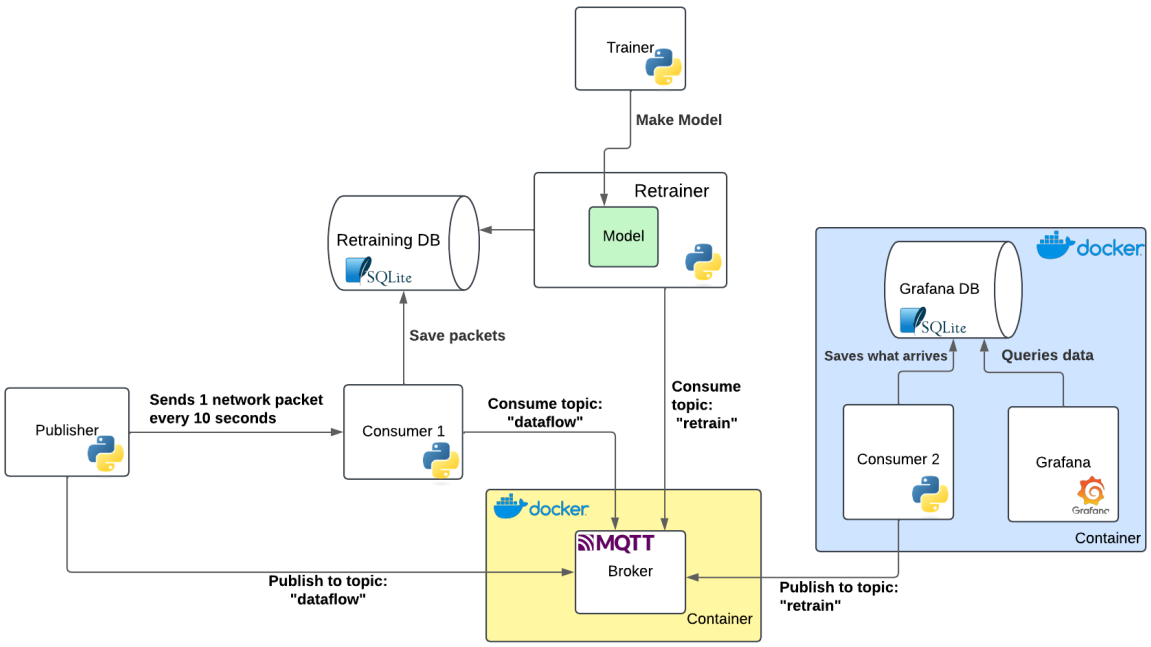

Figure 11: System architecture and respective technologies

The system architecture is designed to process, analyze, and visualize data in a streamlined pipeline using modular components and containerized technologies.

Firstly, it was developed in a python script which is responsible for training the initial model. After creating the model, it is tested and a text file is created in which all the model's accuracies are stored throughout the flow.

```
Model saved to '../Model/logreg_model.pkl'.
Model Performance on Test Data:
Accuracy: 0.816729088639201

Classification Report:
              precision    recall  f1-score   support

         0.0       0.79      1.00      0.88      2764
         1.0       0.98      0.41      0.58      1241

    accuracy                           0.82      4005
   macro avg       0.89      0.71      0.73      4005
weighted avg       0.85      0.82      0.79      4005


Confusion Matrix:
[[2756    8]
 [ 726  515]]
Final accuracy saved to ../Model/final_accuracy.txt
```

Figure 12: Model Train

The flows begins with the **Publisher**, which sends a packet from the dataset every 10 seconds. These packets are published to the **"dataflow"** topic within the **MQTT broker**, a central communication hub hosted in a Docker container. The MQTT broker facilitates the exchange of messages between the various components of the system.

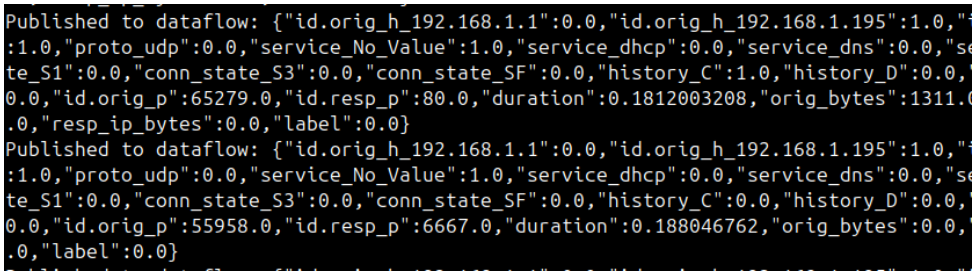

Figure 13: Publisher sending data

A **Consumer** subscribes to the **"dataflow**" topic, processes the incoming packets, stores them in the **Retraining Database**, implemented with SQLite, and keep track how many packets have received. This database serves as a repository for data required for the retraining of the machine learning model.

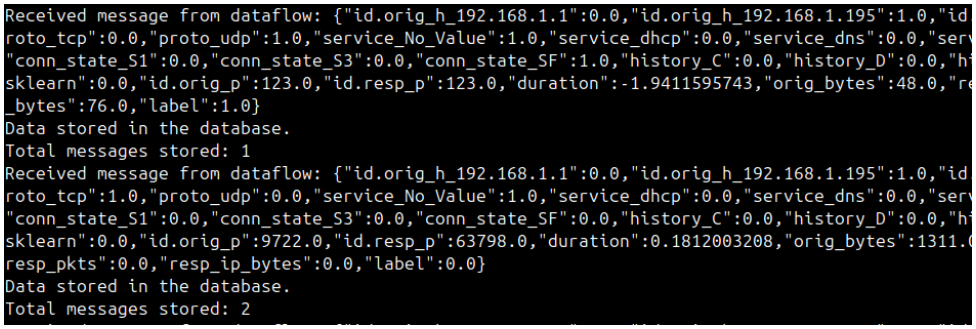

Figure 14: Consumer receiving data

The Consumer in every stablished value of packets calculates the MCC and accuracy using the Test dataset and sends this metrics to the Consumer in the Grafana Container.

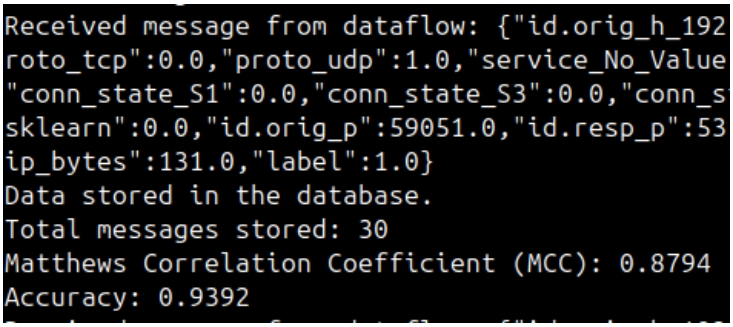

Figure 15: Consumer calculating metrics

Grafana's Consumer receives this values and evaluates them. Firstly stores the values in a database and then if the MCC is above 0.85 it sends a notification to the Retrain script. The Grafana graph, a visualization tool deployed within a Docker container, will be update with this new values show to line, one for the MCC and another for the Accuracy metric. Users can monitor performance metrics of the model evolution in the graph in real time.

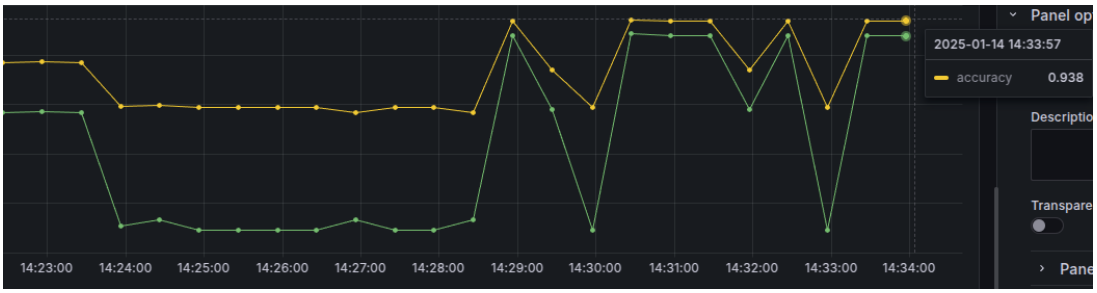

Figure 16: Grafana Graph

The **Retrainer** receives the notification and starts by making a retrain.csv using the values saved in the database. After retraining with the partial fit function it tests the model with the fixed test dataset and saves the values in the file of accuracies. Then proceeds to remove this entries from the database so in the future don't use them to retrain the model.

```
Received message from retrain: {"retrain":true}
Retraining the model...
Exported data to retrain.csv
Model updated and saved.
Model Performance on Test Data (AFTER):
Accuracy: 0.5683333333333334

Classification Report:
              precision    recall  f1-score   support

         0.0       0.65      0.56      0.60       700
         1.0       0.48      0.58      0.53       500

    accuracy                           0.57      1200
   macro avg       0.57      0.57      0.57      1200
weighted avg       0.58      0.57      0.57      1200

Cleaning up the database...
Deleted all data from the database.
Model retrained successfully.
```

Figure 17: Model retraining

```
Final Accuracy: 0.8167 - Inicial
Final Accuracy: 0.77083 - 0
Final Accuracy: 0.77083 - 1
Final Accuracy: 0.56833 - 2
Final Accuracy: 0.77083 - 3
Final Accuracy: 0.77250 - 4
Final Accuracy: 0.77083 - 5
Final Accuracy: 0.59167 - 6
Final Accuracy: 0.59583 - 7
Final Accuracy: 0.58917 - 8
Final Accuracy: 0.58917 - 9
Final Accuracy: 0.58917 - 10
Final Accuracy: 0.58917 - 11
Final Accuracy: 0.56667 - 12
Final Accuracy: 0.58917 - 13
Final Accuracy: 0.58917 - 14
Final Accuracy: 0.56667 - 15
Final Accuracy: 0.93917 - 16
Final Accuracy: 0.74000 - 17
Final Accuracy: 0.58917 - 18
Final Accuracy: 0.94250 - 19
Final Accuracy: 0.93917 - 20
Final Accuracy: 0.93917 - 21
Final Accuracy: 0.74000 - 22
Final Accuracy: 0.93917 - 23
Final Accuracy: 0.58917 - 24
Final Accuracy: 0.93917 - 25
```

Figure 18: Accuracies

This is the full arquitecture of our project made to replicate a real time network packet flow with a real time tracking and real time retraining model each can be always adapting to new data.

4 Models Comparison

To evaluate the performance of various models for our problem, we analyzed the confusion matrices, computed their effectiveness metrics, and determined the Matthews Correlation Coefficient (MCC) for the following models:

- Logistic Regression;
- Decision Tree;
- SVM (Support Vector Machine);
- KNN (K-Nearest Neighbors);
- Random Forest;
- Gradient Boosting;
- AdaBoost;
- SGD (Stochastic Gradient Descent).

Below are some graphs for the models indicated:

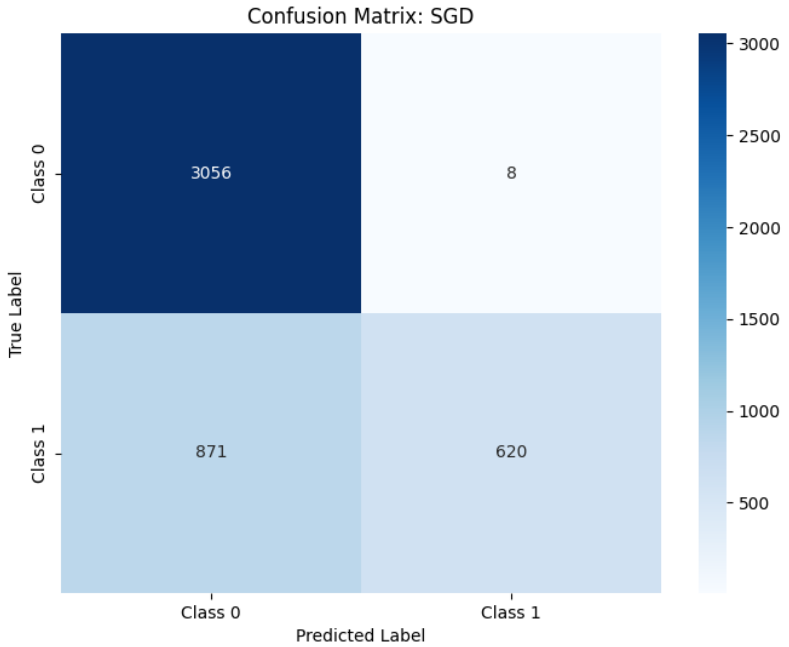

Figure 19: SGD model confusion matrix

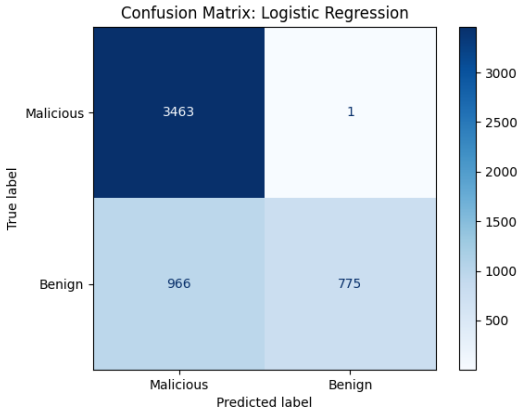

Figure 20: Logistic Regression model confusion matrix

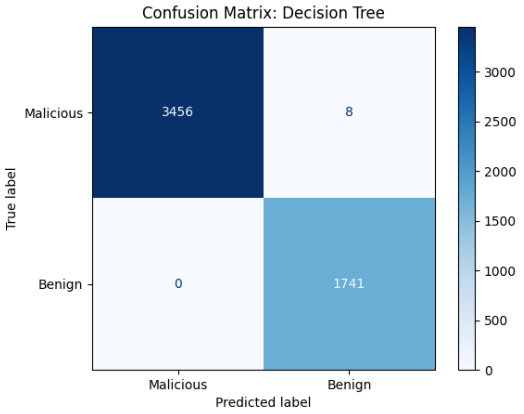

Figure 21: Decision Tree model confusion matrix

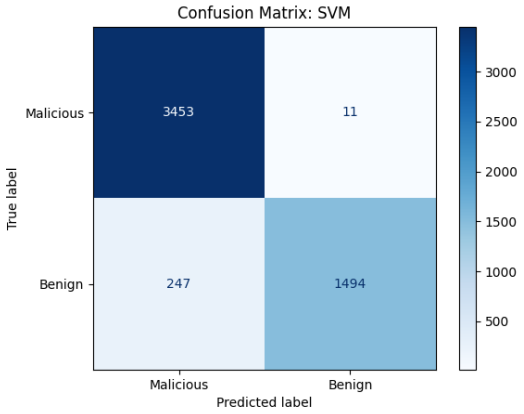

Figure 22: SVM model confusion matrix

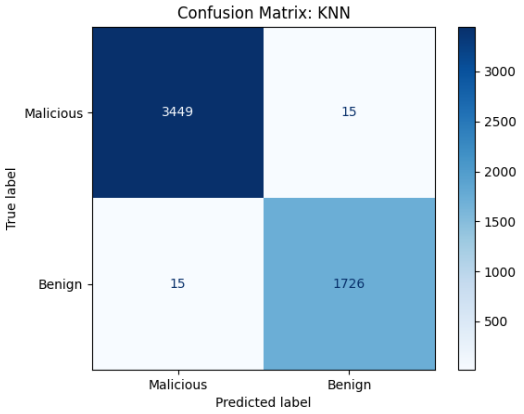

Figure 23: KNN model confusion matrix

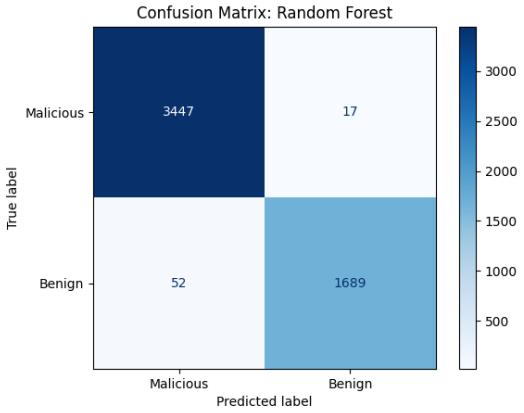

Figure 24: Random Forest model confusion matrix

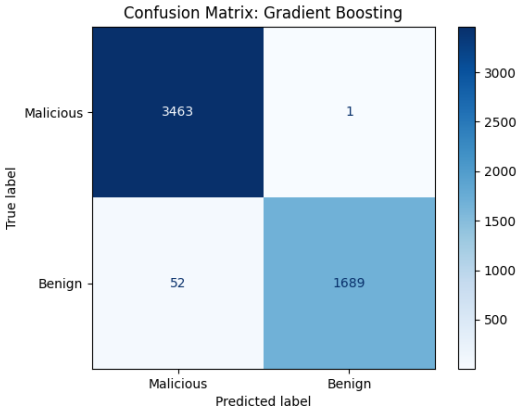

Figure 25: Gradient Boosting model confusion matrix

Although the **SGD model** isn't the one with the best metrics, we ended up **choosing it** because it has a **partial fit function**.

The partial fit function in the SGD model allows for incremental learning, where the model can be updated with new data without requiring a complete retraining from scratch. This feature is particularly advantageous in scenarios involving streaming data or large datasets that cannot be processed in a single batch. By enabling iterative updates, the partial fit function reduces computational overhead and allows the model to adapt dynamically to new patterns in the data, making it highly effective for applications as ours, where retraining with fresh information is essential.

5 Grafana and Results

To visualize the evolution of the model's accuracy and MCC, we used Grafana.

Grafana is a widely-used open-source platform for monitoring and visualizing time-series data. It allows users to create dynamic dashboards by connecting to a variety of data sources such as databases, cloud services, and IoT platforms. In our case, we utilized Grafana to visualize the evolution of the model's accuracy and MCC. For that, we started by adding the SQLite DB where these values are stored as the data source. Then, we did the following query to display the values:

```
SELECT
  strftime('%s', timestamp) AS time,
  mcc_value,
  accuracy
FROM
  mcc_data
ORDER BY
  time ASC
```

This way, we monitored both the MCC and accuracy over time, obtaining the following image:

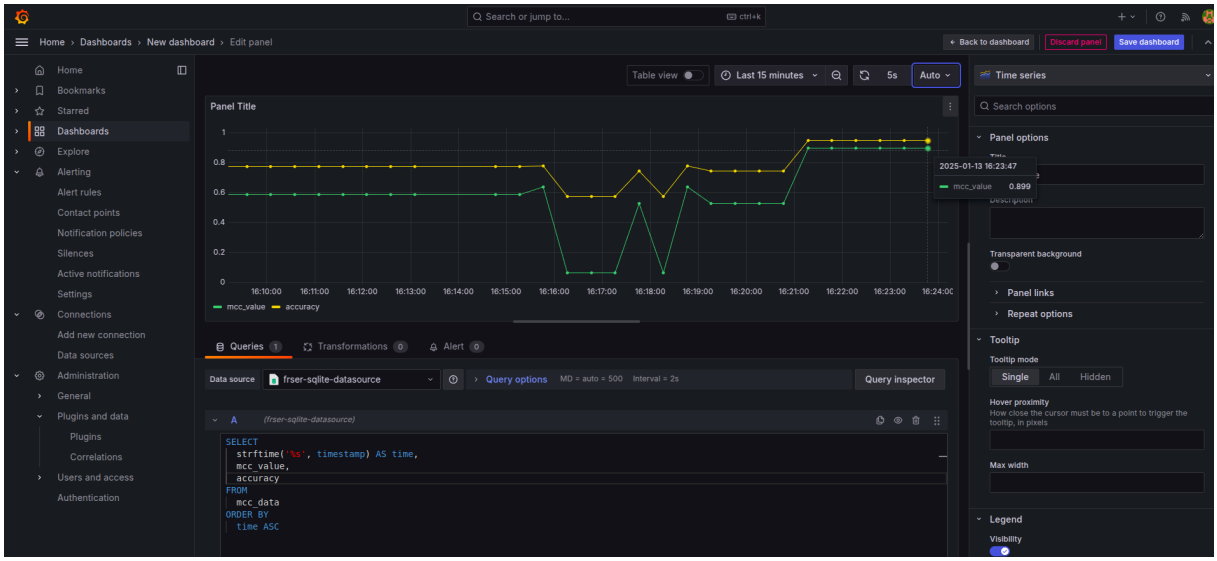

Figure 26: MCC and Accuracy evolution over time with 20 packets each time

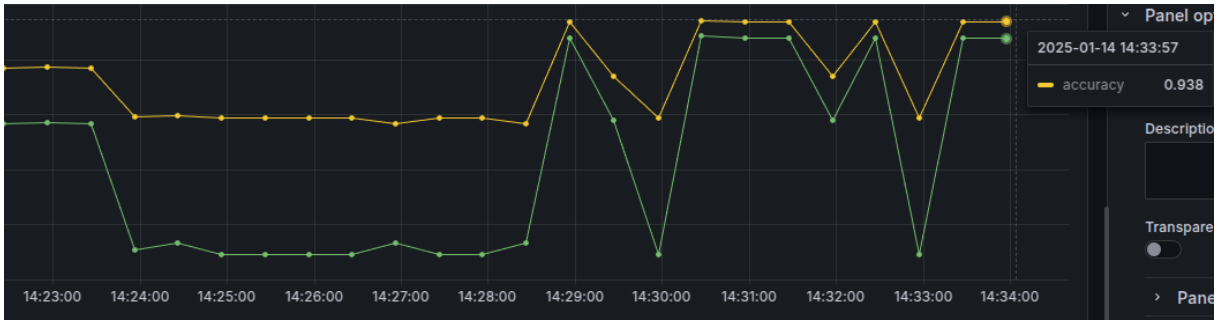

Figure 27: MCC and Accuracy evolution over time with 30 packets each time

We have analyzed the alerts from the Grafana platform, but we prefer to use the alerts directly in the python script for reasons of efficiency.

After analyzing the graph, we observe that the model initially achieves an accuracy of approximately 81%. This accuracy decreases when it receives a batch of 20-30 packets that are predominantly malicious, as the model misclassifies some benign packets as malicious. However, as more packets are processed, the accuracy and MCC (Matthew's Correlation Coefficient) gradually improve. The model exhibits occasional spikes in performance when it receives a large number of packets from a single label, but by the end of the process, it stabilizes and achieves an accuracy of around 94% or higher, representing a 15% improvement from its initial value.

Subsequently, we conducted an experiment using batches of 150 network packets, triggering a retraining notification after each batch. We observed that the graph exhibited greater stability and reached the highest accuracy value more rapidly, with fewer retraining iterations. In this scenario, the model achieved its peak accuracy after thirteen retraining sessions, reaching 96.25%, representing a 19.17% improvement from its initial value.

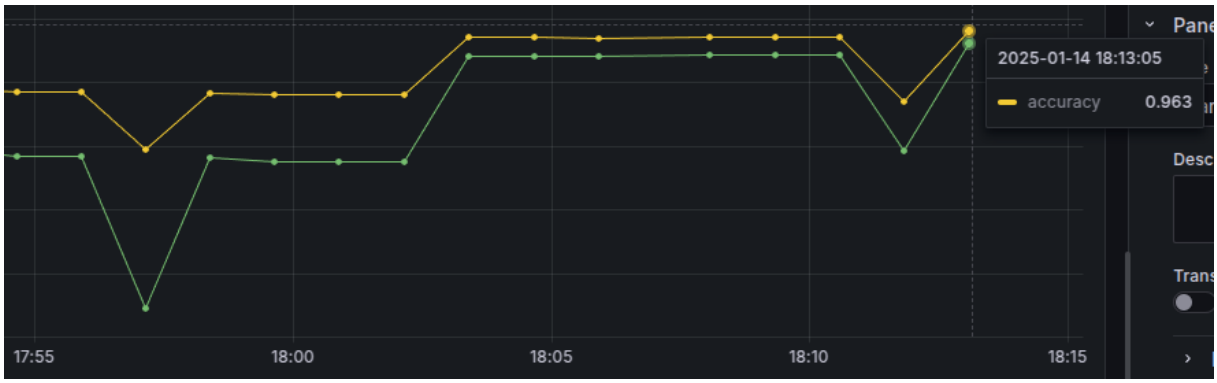

Figure 28: MCC and Accuracy evolution over time with 150 packets each time

This is the comparison from the initial confusion matrix and the final one:

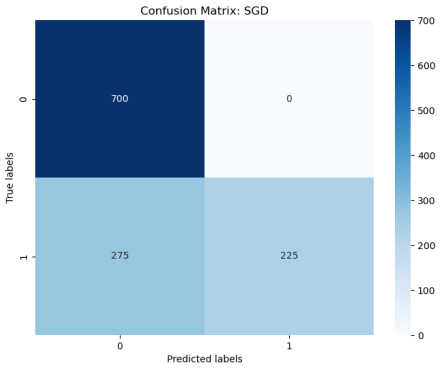

Figure 29: Inicial Confusion Matrix

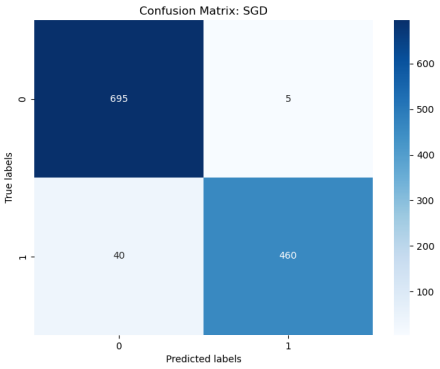

Figure 30: Final Confusion Matrix