

# TUM Data Innovation Lab

Munich Data Science Institute (MDSI)

Technical University of Munich

&

iABG

Final report of project:

# Cyberthreat development and attack detection with Artificial Intelligence in 5G campus networks


| | |
|---|---|
| Authors | Aziz Ben Ali, Paul Ortulidis-Pflanz, Petra Juros, Sang Hyeon Lee, Thomas Erhart |
| Mentor(s) | Dr. Víctor Hugo Baños González, Maik Holzhey (iABG) |
| TUM Mentor | Dr. Alessandro Scagliotti |
| Project lead | Dr. Ricardo Acevedo Cabra (MDSI) |
| Supervisor | Prof. Dr. Massimo Fornasier (MDSI) |


July 19, 2024

# Abstract

This project concentrates on cybersecurity in 5G networks by developing a machine learning-based system for detecting cyber attacks. For that we started by configuring a 5G network environment, including the 5G Core and 5G Radio Access Network (RAN). Then generated network traffic data through normal operations and simulated attacks like Hydra and Metasploit. After gathering and preprocessing the data, various machine learning models were implemented and evaluated. The methods included supervised (Random Forest, XGBoost, and Deep Neural Networks), unsupervised (Hierachical CLustering, K-Means, Variational Autoencoders, and Self Organizing Maps), and reinforcement learning techniques. After evaluating the results obtained, we highlight the XGBoost achieved the highest precision and recall in detecting anomalies. Additionally, we explored active learning to potentially enhance the model performance. Finally, we create and integrate the model in a Security Information and Event Management (SIEM) system, with an alert of potential threats feature. This report details the setup of the 5G environment, the execution of attacks, data preprocessing, and assessment of different machine learning models in analysing 5G network traffic.


# Contents

# 1 Introduction

Technology transforms our world daily, impacting every aspect of our lives, including how companies conduct business. To remain competitive and resilient, companies must adapt and develop strategies to meet the challenges of the digital era. The launch of 5G technology marks a significant leap in mobile communications, promising unprecedented speed, low latency, and the ability to connect a vast number of devices simultaneously. This next-generation network is expected to revolutionize various sectors, including healthcare, transportation, and smart cities, by enabling innovative applications such as autonomous vehicles, remote surgeries, and the Internet of Things (IoT).

However, cybersecurity challenges must be addressed to ensure the safety and integrity of data and communications. Already today and in the past the importance of cybersecurity can be seen looking at the numbers: in 2023, there were over 2,000 cyberattacks affecting nearly 350 million victims. Additionally, 94 percent of companies reported experiencing email security incidents, resulting in a 2.7 billion dollar loss in 2022. [3]

Given that 5G technology is newly developed, it introduces novel cybersecurity threats. Traditional approaches to cybersecurity may not be as effective in this new landscape. Therefore, our project aims to develop a machine-learning model capable of detecting attacks in network traffic, focusing primarily on reinforcement learning while also exploring supervised and unsupervised learning methods. Leveraging them, we aim to identify attacks of the MITRE ATT&CK framework from normal traffic data.

# 2 State of the Art Approaches and Algorithms

The starting point for this project was two theses. The first one was "AI-based Open Source SIEM for cyber threat detection and response", written by K. Blachut. The second one was authored by Nicolas Zapf with the title "Detection and Response in an AI based SIEM for 5G campus network". In both, they presented great result when dealing with cybersecurity threats. One leveraging supervised and the other unsupervised machine learning methods.

## 2.1 Supervised Learning Approaches

In the first thesis [2] Blachut used artificial neural networks (ANN) to detect anomalies in the network traffic. Firstly, she implemented a binary classification and then a multiclass classification model. She divided the dataset into train and test sets. Training set consisted of 80 percent of the data and testing set of the rest. In the ANN for binary classification, Rectified Linear Unit(ReLu) was used as an activation function for all hidden layers and sigmoid functions for the output layer. Adaptive Moment Estimation(Adam) was the chosen optimizer function. The results of performing binary classification were really good: the model predicted 99.64 percent of all samples correctly. For the multiclass classification task, the ANN was adapted. The activation function of the last layer was changed to a softmax and the loss function to the categorical crossentropy loss. The model should predict the correct on of the seven classes (six different types of attacks and normal traffic). Accurate labels were given to 98.69 percent of the data. Attacks were labeled as regular traffic in 0.48 percent of the cases. Regular traffic was classified as an attack in 0.00 percent of the cases.

## 2.2 Unsupervised Learning Approaches

The second thesis focused primarily on unsupervised machine learning method. Initially, anomaly detection was performed using compression techniques such as PCA and autoencoders. For PCA, they preprocessed the data and divided it into training and test sets. The training set consisted exclusively of normal traffic data, while the test set included 90% normal traffic data and 10% attack samples. It was experimented with various numbers of principal components, achieving the best result with more than 18 principal components, where the F1-score was nearly 1.

It was also employed autoencoders, progressively increasing the latent dimension with each iteration. The F1-score averaged around 0.9 across all iterations, with precision consistently higher than the F1-score. Clustering was another approach used for outlier detection, setting the number of clusters to two (normal traffic and attack samples) for all methods. Various clustering techniques were applied: K-means in the first iteration, MiniBatchKMeans in the second, BisectingKMeans in the third, Birch Clustering in the fourth, and Gaussian Mixture Clustering in the fifth. The results were promising, with precision generally at 1, recall at 0.7, and F1-score at 0.82. The best result was achieved using Gaussian Mixture Clustering, where the recall was nearly 1, precision was 0.9, and F1-score was almost 0.95.

# 3 Data Generation

Generating 5G network traffic data is crucial for developing and testing cybersecurity measures. The data helps simulate various network scenarios, including normal operations and potential cyber-attacks, to train and evaluate machine learning models for anomaly detection and threat mitigation. This section describes the setup and configuration of the 5G network components used in our data generation process, specifically the 5G Core and 5G RAN.

## 3.1 5G Setup

### 3.1.1 5G Core Network Configuration

The 5G Core Network manages key functions such as session management, mobility management, and security of the 5G network. It ensures seamless data flow and control across the network.

#### Setup

To configure the 5G Core Network, we built docker images for various network functions, including the Access and Mobility Management Function (AMF), Session Management Function (SMF), Network Repository Function (NRF), User Plane Function (UPF), Unified Data Repository (UDR), Unified Data Management (UDM), Authentication Server Function (AUSF), Network Slice Selection Function (NSSF), Network Exposure Function (NEF), and Policy Control Function (PCF). We used the open-source implementation guide from OpenAirInterface to achieve this setup.

#### Access and Mobility Management Function (AMF) Configuration

The AMF is responsible for handling user equipment (UE) access and mobility procedures, including registration, connection management, and handovers. We configured the AMF to ensure efficient management of UE sessions and mobility events.

#### Session Management Function (SMF) Configuration

The SMF manages session contexts and handles the establishment, modification, and release of sessions for the user plane. This includes managing IP address allocation, quality of service (QoS) enforcement, and traffic routing policies.

### 3.1.2 5G RAN Configuration

The Radio Access Network (RAN) is responsible for the radio interface, ensuring proper communication between the User Equipment (UE) and the 5G Core Network.

#### Setup

We configured the USRP B210 to act as the next-generation Node B (gNB), which processes signals and manages radio transmissions. This setup was also achieved using the OpenAirInterface framework.

#### USRP Hardware Driver (UHD) Configuration

The UHD provides the necessary drivers and software to interface with the USRP B210 hardware. Proper configuration of the UHD is crucial for the correct operation of the gNB.

#### gNB (5G Base Station) Software Configuration

The gNB software is responsible for managing the physical and MAC layers of the radio interface, including scheduling, retransmissions, and radio resource management. We configured the gNB to ensure efficient communication with the UEs and the 5G Core Network.

### 3.1.3 Modem and User Equipment

In our setup, user equipment (UE) was connected to 5G modems, specifically the Quectel RM500Q, to generate 5G traffic. The UE simulates various network activities, including web browsing, video streaming, and file transfers, to generate diverse traffic patterns.

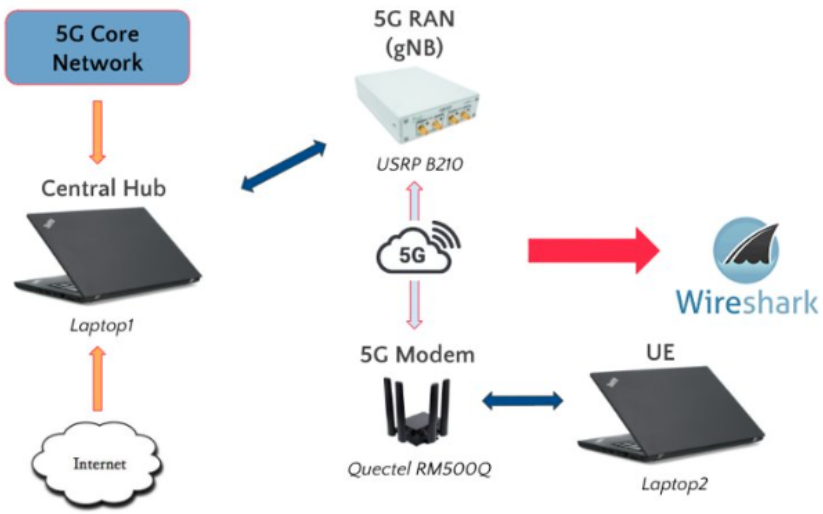


Figure 1: Diagram of the 5G Network Setup, including the 5G Core, 5G RAN, and UE with the Quectel modem.

### 3.1.4 Network Protocol Analyzer

We used Wireshark, a network protocol analyzer, to monitor and capture 5G traffic in real-time. Wireshark provided detailed insights into the network traffic, allowing us to analyze the data and identify potential anomalies.

### 3.1.5 Conclusion

The 5G network setup, including the core network, RAN, and user equipment, along with network protocol analysis tools, provided a robust framework for generating and capturing 5G network traffic

data. This data serves as foundation for training and evaluating machine learning models aimed at classifying 5G network traffic.

## 3.2 Performed Attacks

After we had set up the 5G network, we were able to perform the attacks ourselves and generate more data for our model training later. We performed Hydra and Metasploit attack.

### 3.2.1 Hydra Attack

Kali Linux is a specialized operating system designed for penetration testing and ethical hacking. [4] To perform the Hydra attack we need the Hydra tool, which is a preinstalled tool in Kali Linux. Hydra performs a brute-force attack, trying to guess and crack valid pairs of username and password. When using Hydra, there is a useful option to monitor the count of failed attempts. This is important because companies like Facebook or Google have limited the maximum number of attempts when trying to login. While we were doing the Hydra attack we also used EtherApe. EtherApe is a graphical network monitoring tool. It displays network activity graphically. We used it to monitor attacker's machine traffic. The attack was performed as follows. First as mentioned, we had a 5G core, 5G RAN and Kali Linux installed. Then, we searched on the internet for a dictionary of most commonly used combinations of usernames and passwords, which was handed to Hydra to start brute-forcing. We then attacked the RAN with Hydra on our Kali Linux machine until we found the right combination of username and password.

### 3.2.2 Metasploit Attack

#### General Setup

During the Metasploit attack a victim and an attacker system is needed. The setup was as follows: The victim computer ran a 5G campus network and an application server called Apache Tomcat. Apache Tomcat is a web server primarily designed for Java-based applications. This server was the target of the attack, aimed to access the 5G Campus Network. For this the attacker leverages the Metasploit Framework, progressing through the three stages of reconnaissance, exploitation, and payload deployment:

Reconnaissance: Reconnaissance is the phase where information is gathered about the target system to identify potential vulnerabilities and plan the attack.

Exploitation: Exploitation is the process of taking advantage of a vulnerability in the target system to gain unauthorized access or control.

Payload: A payload is the part of the exploit that performs the intended malicious action, such as opening a reverse shell to provide remote access to the attacker.

#### Reconnaissance

During our attack, in the first stage, the attacker gathered as much information as possible about the victim. To achieve this goal, he used tools like nmap to identify any open ports and the services running on those ports. While scanning the network, the attacker PC discovered that the Apache Tomcat Manager was open and potentially vulnerable to exploitation.

#### Exploitation

Having gathered the necessary information, we move to the exploitation phase. In this stage, we tried gain unauthorized access to the Apache Tomcat Manager.

The file contained a reverse shell as payload. A reverse shell is a type of shell where the target machine initiates a connection back to the attacker's machine, allowing the attacker to execute commands on the target system remotely without getting noticed.

#### Payload

With everything in place, the exploit was executed. The Metasploit Framework uploaded the malicious file to the Apache Tomcat Manager. This process established a reverse shell, granting unauthorized remote access to the server.

From this reverse shell, arbitrary commands could be executed on the compromised server, effectively gaining control over the Apache Tomcat instance and potentially the broader 5G Campus Network. This foothold allowed to further explore, exfiltrate data, and potentially launch additional attacks within the network.

# 4 Data Analysis and Preprocessing

The data gathered from these attacks together with recorded normal traffic and all the data available were used as the basis for training our models. The traffic data of all types is first saved as "packet capture" (=pcap) files. To get this files, the traffic was analyzed with the protocol analyzer tool Wireshark, which makes it possible to capture the current traffic running on a network. The traffic gets seperated in 30 second intervalls and then the specific features of the intervalls get extrated. A list with all 52 features and a short description can be found in the following:

| **Feature Name** | **Explanation** |
|---|---|
| interval start | Start time of the interval in a formatted date-time string. |
| source ports count | Number of unique source ports in the captured packets. |
| destination ports count | Number of unique destination ports in the captured packets. |
| internal tcp count | Count of TCP packets with both source and destination IPs being internal. |
| ports ratio | Ratio of destination ports count to source ports count. |
| packet count per internal source port avg | Average packet count per internal source port. |
| packet count per internal destination port avg | Average packet count per internal destination port. |
| duplicates sum | Total number of duplicate packets. |
| retransmission sum | Total number of retransmitted TCP packets. |
| window size avg | Average TCP window size. |
| tcp packet length avg | Average length of TCP packets. |
| initial rtt avg | Average initial round-trip time for IP packets. |
| flag X count | Count of unique TCP flags. |
| packets sum | Total number of packets. |
| internal packets sum | Total number of internal packets. |
| packets count | Total count of packets. |
| small packets count | Count of packets with a length less than 100 bytes. |
| internal packets count | Count of packets with both source and destination IPs being internal. |
| packet count ratio | Ratio of internal packets count to total packets count. |

| | |
|---|---|
| packet sum ratio | Ratio of internal packets sum to total packets sum. |
| packet length avg | Average packet length. |
| internal packets avg | Average length of internal packets. |
| ip pairs count | Number of unique source-destination IP pairs. |
| internal ip pairs count | Number of unique internal source-destination IP pairs. |
| packet length when ssh avg | Average length of packets when using SSH. |
| packet length when ssh to normal avg | Ratio of average packet length when using SSH to overall average packet length. |
| ssh packets count | Count of packets using SSH. |
| ssh packets ratio | Ratio of SSH packets count to total packets count. |
| ssh packet length avg | Average length of SSH packets. |
| ssh padding length avg | Average padding length in SSH packets. |
| ssh unique pairs count | Number of unique SSH source-destination IP pairs. |
| ssh unique pairs count ratio | Ratio of unique SSH IP pairs count to overall IP pairs count. |
| ssh unique pairs count internal ratio | Ratio of unique SSH IP pairs count to internal IP pairs count. |
| ssh median time between frames | Median time between consecutive SSH packets. |
| arp count | Count of ARP packets. |
| arp packet ratio | Ratio of ARP packets count to total packets count. |
| http get count | Count of HTTP GET requests. |
| http post count | Count of HTTP POST requests. |
| udp packets avg | Average length of UDP packets. |
| severity group X count | Placeholder for severity group count, needs to be updated. |
| source addresses count | Number of unique source IP addresses. |
| destination addresses count | Number of unique destination IP addresses. |
| ip pairs ratio | Ratio of internal IP pairs count to overall IP pairs count. |
| address ratio | Ratio of destination addresses count to source addresses count. |
| pkt cnt per internal src adr avg | Average packet count per internal source IP address. |
| pkt cnt per internal dst adr avg | Average packet count per internal destination IP address. |
| pkt cnt per internal pair avg | Average packet count per internal source-destination IP pair. |
| unique ipdest to totaltraffic ratio | Ratio of unique destination IP addresses to total packets count. |
| ttl avg | Average Time-To-Live (TTL) value of IP packets. |
| ip pkt length avg | Average length of IP packets. |
| entropy source ip | Entropy of source IP addresses, indicating their distribution and randomness. |
| entropy destination ip | Entropy of destination IP addresses, indicating their distribution and randomness. |

Table 1: Network traffic features for anomaly detection [2], [9]

All these features of the corresponding piece of traffic get saved in a csv file. Moreover, we performed the following preprocessing steps. Firstly, we checked for highly correlated features and removed the ones with a correlation over 90 percent. Further we droped Features with low class correlation coefficients below 1%. Secondly, we transformed the data to make sure that all are in the same range. Lastly we checked if the dataset was imbalanced and used the RandomOversampling method to make it balanced.

In the end we have the following numbers of 30 second traffic samples of the respecting type (specific attack or normal traffic):

| Class | Number of samples |
|---|---|
| Normal Traffic | 776 |
| Hydra Attack | 191 |
| Docker Command | 26 |
| ARP Poisoning | 46 |
| DDOS | 23 |
| SQL Injection | 17 |
| Injections Exploit | 11 |
| nmap | 6 |

Table 2: Summary of the amount of gathered network traffic samples

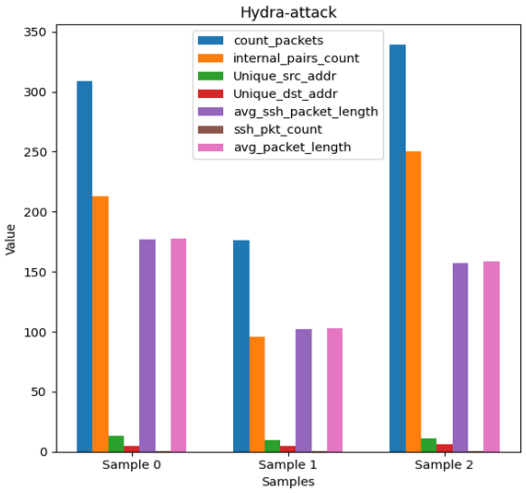


Figure 2: Selected features of Hydra Attack samples

So in total we have 1096 samples with 52 extracted features for every single one. A visualization of some common features of three Hydra Attack samples can be seen in figure 4.

# 5 Selected Algorithms and Methodology

To efficiently predict and mitigate attacks in 5G infrastructure, we employ different advanced machine learning techniques, encompassing supervised, unsupervised, and reinforcement learning approaches. Each model and methodology is chosen for its unique strengths in addressing specific challenges associated with 5G network security. In the following sections, we will explore each employed model, detailing its role and the reasons behind its selection.

## 5.1 Unsupervised Learning

Unsupervised learning is a powerful paradigm in machine learning that derives patterns and structures from data without the need for given labels. Unlike supervised learning, where the goal is to learn a general mapping from given inputs to given outputs based on annotated samples, unsupervised learning focuses on discovering the inherent structure of the data by itself. Since it does not rely on predefined labels, unsupervised learning is more adaptable and can uncover novel insights that might not be evident through traditional supervised methods and in this way help during classification of unknown attack data as well. Because in the given use case of 5G attack detection the focus lies on separating samples belonging to specific attacks from others, in this section unsupervised clustering algorithms are explained and analyzed. Here the chosen clustering algorithms are the K-means clustering, Hierarchical clustering and Self-Organizing maps. Moreover, the variational autoencoder is leveraged for the use case of anomaly detection to identify outliers which differ to much from normal 5G data traffic.

### 5.1.1 Hierarchical Clustering

#### Theory

Hierarchical clustering is a method which always seeks to find clusters with similar data points to merge or separate them over arbitrary stages. This method can be either agglomerative ("bottom-up"), meaning that the start are as many clusters as data points. From this initial state the closest clusters are searched and merged until the end state is reached at certain number of clusters or when everything got merged to a single one. The second variant besides the agglomerative hierarchical clustering is divisive ("top-down")

hierarchical clustering. Here, the start is one cluster containing all data points, which gets separated in clusters with higher similarity. Now, we will focus on the agglomerative approach, which is more commonly used. In the first step the pairwise distance between all the single data points has to be assessed. Here different affinity metrics are used to define the similarity like euclidean distance or cosine similarity. To decide which clusters will be merged in the next hierarchy level, different linkage criteria are used to assess inter-cluster distance in the second step. Common are for example single (shortest distance between any pair of points), average (average distance between all pairs of points) or complete ( longest distance between any pair of points) linkages. Then the closest clusters get merged. This last two steps of assessing the inter-cluster distance and merging is repeated until the desired end state is reached. [6]

#### Results

In our case the final hierarchical clustering model had the following attributes. Like already said it was an agglomerative approach and the goal was to end up with 8 clusters. Moreover the algorithm used the cosine affinity for measuring the distance between data points and the complete linkage criteria to assess inter/cluster distances. In Figure 3 the results can be seen.

Moreover, we quantify the result with two metrics, the Adjusted Rand Index (ARI) and the Normalized Mutual Information (NMI). Both are metrics used to evaluate the similarity between two data clusterings, and are typically used for evaluating algorithm's output clustering compared to a ground truth clustering. With Hierachical CLustering we achived an ARI of 0.96 indicating a very high level of agreement between the clustering result and the ground truth, suggesting that the clustering algorithm performed very well. The NMI with a score of 0.81 indicates a decent high degree of shared information between the clusterings, suggesting that the clustering algorithm captured a significant amount of the structure present in the ground truth.

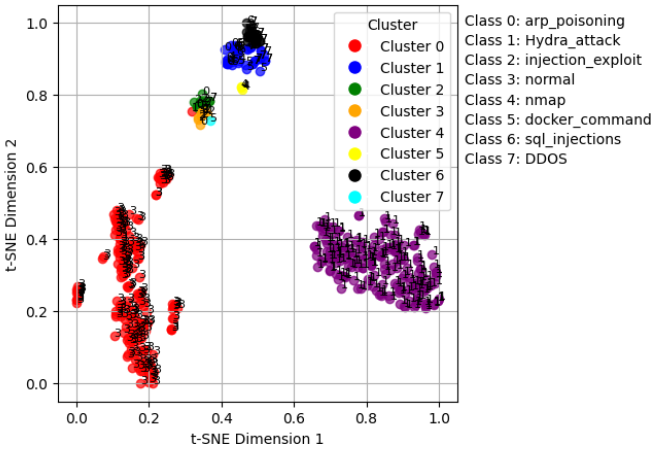


Figure 3: True Class Labels and Cluster Borders in Latent Space (t-SNE)

### 5.1.2 K-Means Clustering

#### Theory

K-Means clustering is a widely-used unsupervised learning algorithm that aims to partition a dataset into K distinct clusters, where each data point belongs to the cluster with the nearest mean (also called centroid). In the context of 5G campus network data samples, K-Means clustering can effectively segregate different types of network activities or traffic patterns, helping to identify distinct groups such as regular network usage and specific attack patterns of Hydra attack for example. The only hyperparameter to choose here is the number of clusters K. At the start of the algorithm, the K centroids of the clusters are initialized randomly. Then the distance between the data and all centroids is assessed and all data points at hand are assigned to the closest centroid. When all are assigned the centroids are updated as the mean of all data points of the corresponding cluster. Assessing the distance, assigning all data points and updating the centroids is repeated until the centroids do not change anymore. The advantages of the K-means algorithm are the efficiency and scalability. On the other hand the random choice of K and initialization make the algorithm non-deterministic and reduce the robustness. [1]

**Results**

In the end the tested K-means algorithm is initialized always with the same random centroids to secure reproducibility. The number of seperate clusters in our model is set to 8 again. The t-SNE plot visualizes the clustering results of the K-Means algorithm applied to the network traffic data of a 5G network. Each point on the plot represents a 30 second network traffic sample, with colors indicating the clusters assigned and numbers referring to the true class labels. The definition of the numbers is defined in the list on the right.

The plot shows a good alignment between some true class labels and the assigned clusters. For instance, samples of class 1 which corresponds to the Hydra Attack are predominantly grouped within the cluster 1, suggesting that the algorithm can effectively isolate this type of attack. Other clusters contain multiple class labels, indicating overlap between different types of traffic or attacks. For instance, cluster 0 has mixed class labels of Docker Command (5), SQL Injection (6) and DDOS (7). It shows that these traffic types are more challenging to distinguish with the algorithm based on the given features.

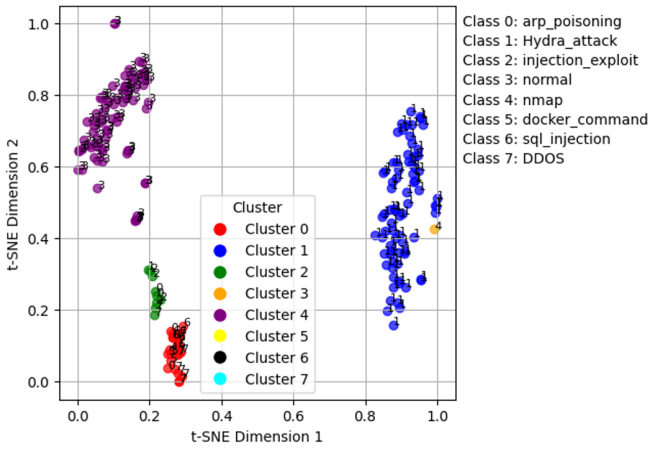


Figure 4: True Class Labels and Cluster Borders in Latent Space (t-SNE)

### 5.1.3 Self Organizing Maps

**Theory**

Self Organizing Maps (SOM) is another unsupervised learning method we used for attack detection in our project. SOM was first implemented by Kohonen in 1982, which is why it is often called the Kohonen map. [8]

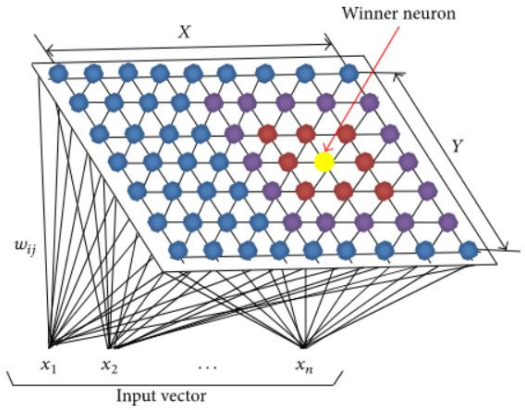


Figure 5: Self Organizing Map Representation

The SOM algorithm works as follows. We have a dataset with $n$ data points, and each data point has $m$ features. At the same time, we define a two-dimensional grid. Each element of the grid is called a neuron. A neuron has a weight vector of dimension $m$. The goal of the algorithm is to find a suitable weight vector for each neuron so that the grid models the distribution of the data in input space by clustering similar feature vectors. For each data point, the algorithm calculates which neuron has the weight vector with the biggest similarity to the data point. The neuron with the corresponding biggest

similarity or shortest distance is called a winning neuron. The weight vector of the winning neuron gets updated based on the input data point. The formula for updating the winning neuron is :

$$w_{winner} = w_{winner} + \beta(x - w_{winner})$$

Where $x$ is the vector of the input data point and $\beta$ is the learning rate. The neighbors of the winning neuron also get updated, weighted by the distance to the winning neuron. [8]

**Results**

Before we implemented the SOM algorithm, we wanted to reduce the number of features using PCA. To find out how many principal components to use, we wanted the principal components to explain more than 95 percent of the cumulative variance. That is achieved by using PCA with 13 principal components. The results of SOM are shown with the confusion matrix of the model. The precision of the model is 0.91, indicating that the majority of the positive predictions are indeed true positives. The recall is 0.78, which suggests that the model misses a significant number of actual positive samples, resulting in a high number of false negatives. The F1-score, which considers precision and recall, is 0.79. It is important to note that accuracy is not used to evaluate the model's performance due to the imbalanced nature of our dataset.

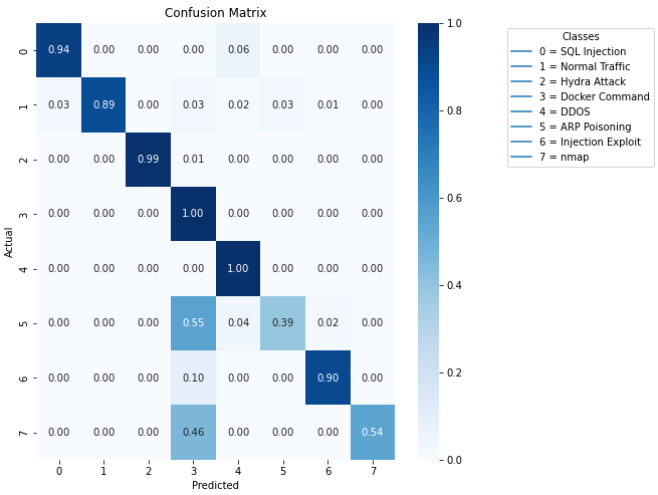


Figure 6: Confusion Matrix for SOM

### 5.1.4 Variational Autoencoder

**Theory**

A Variational Autoencoder (VAE) is a generative model that learns to encode input data into a latent space and then decodes it back to reconstruct the original input. This process involves an encoder that maps the input data to a latent space characterized by a mean and variance, and a decoder that reconstructs the input data from the latent representation.

In the context of anomaly detection, the VAE is first trained on a dataset consisting of normal traffic data. The training process aims to minimize the reconstruction error, which is the difference between the input data and the reconstructed data. Once trained, the VAE can accurately reconstruct normal traffic data, resulting in low reconstruction errors.

The diagram illustrates the anomaly detection process using a VAE:

1. **Training the VAE:**
- The VAE model is trained and tested on an initial normal traffic dataset.
- The encoder maps the input data to a latent space, and the decoder reconstructs the input from the latent representation.
- The reconstruction error is computed, and a threshold is established based on the reconstruction errors of the normal traffic data.

2. **Using the VAE for Anomaly Detection:**
- The trained VAE model is used to evaluate new traffic data.
- The input data is encoded and then decoded to compute the reconstruction error.
- If the reconstruction error exceeds the established threshold, the data is classified as an anomaly (potential attack).
- If the reconstruction error is below the threshold, the data is classified as normal.

This approach leverages the VAE's ability to learn the patterns of normal traffic data, making it sensitive to deviations that indicate anomalies or potential attacks.

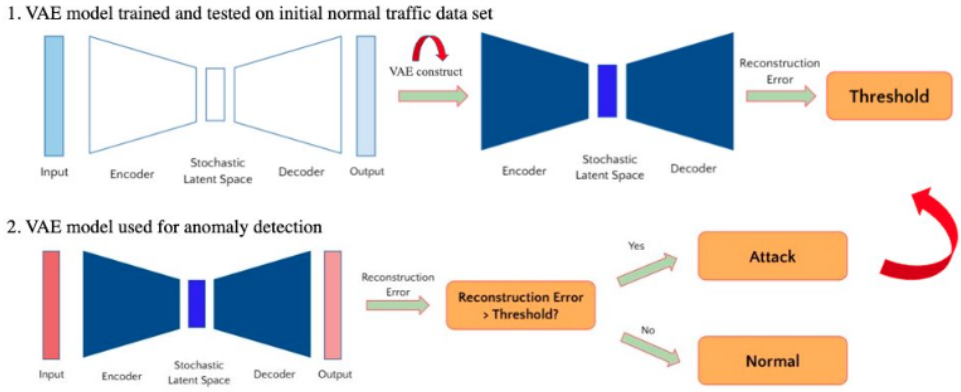


Figure 7: VAE model training and anomaly detection process.

**Results**

The performance of the Variational Autoencoder (VAE) model on the anomaly detection task is summarized by the evaluation metrics and confusion matrix. The precision of the model is 1.0000, however, the recall is only 0.6656, resulting in a high number of false negatives. The F1-score, is 0.7992, reflecting the imbalance between precision and recall.

The Area Under the Receiver Operating Characteristic Curve (AUC-ROC) is 0.8328, which, while decent, further indicates that the model is not performing optimally. The confusion matrix provides additional insight into the model's performance:

- **True Positives (TP):** 209
- **True Negatives (TN):** 188
- **False Positives (FP):** 0
- **False Negatives (FN):** 105

From the confusion matrix, it is clear that the model is highly conservative in its predictions, prioritizing precision over recall. This behavior is evident from the 105 false negatives, which are instances where actual positives are not detected by the model. In an anomaly detection scenario, such a high number of false negatives can be problematic, as it means many anomalies are going undetected.

In conclusion, the VAE model, despite being a straightforward approach to binary classification, does not perform adequately in the context of anomaly detection. The high precision is overshadowed by the low recall, resulting in numerous false negatives. This indicates a need for further tuning or potentially exploring more advanced models or techniques to achieve better performance in detecting anomalies.

## 5.2 Reinforcement Learning

### 5.2.1 Q-Learning

#### Theory

We will follow the ideas outlined by [5] and [7] in the subsequent paragraphs on Q-Learning and Deep Q-Learning. Q-Learning is a model-free reinforcement learning algorithm used to determine the best course of action for an agent to take in a given environment to maximize its cumulative reward over time. The term "model-free" means that the algorithm does not require a model of the environment; it learns directly from interactions with the environment.

In Q-Learning, the agent learns a function called the Q-function, which is denoted by $Q(s, a)$. This function represents the Q-values, which are estimates of the expected utility or total reward of taking a specific action $a$ in a particular state $s$ and then following the optimal policy (the best possible strategy) thereafter. Essentially, the Q-value tells the agent how good it is to take a certain action from a given state.

The core idea of Q-Learning is to update these Q-values iteratively based on the agent's experiences. Here is how the process works:

1. **Initialization**: The algorithm starts by initializing the Q-values for all state-action pairs to arbitrary values, typically zero. Here, in our use case the states are the features and the actions are the classification of the 5G traffic. This means initially, the agent has no knowledge about the environment.
2. **Interaction with the Environment**: The agent interacts with the environment in a series of time steps. At each time step, the agent observes the current state $s$ of the environment.
3. **Action Selection**: The agent selects an action $a$ to take from the current state. The selection is usually based on a policy, such as the $\epsilon$-greedy policy, where the agent mostly chooses the action with the highest Q-value (exploitation) but occasionally chooses a random action (exploration) to ensure it explores the environment adequately.
4. **Receiving Reward and Transition**: After taking the action $a$, the agent receives a reward $r$ from the environment and observes the new state $s'$ that it transitioned to as a result of the action.
5. **Q-Value Update**: The agent updates the Q-value for the state-action pair $(s, a)$ using the Q-Learning update rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left( r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right)$$

   Here:

   - $\alpha$ (the learning rate) determines how much the update is weighted, so how new information overrides old information. It typically ranges between 0 and 1.
   - $r$ is the immediate reward the agent receives for taking action $a$ in state $s$.
   - $\gamma$ (the discount factor) represents the importance of future rewards. It also ranges between 0 and 1, where a value close to 1 makes future rewards more significant.
   - $\max_{a'} Q(s', a')$ is the maximum estimated Q-value for the next state $s'$, considering all possible actions $a'$ from that state.
6. **Repeat**: The agent repeats steps 2-5 for each time step, continuously updating its Q-values based on new experiences. Over time, as the agent explores the state-action space and updates its Q-values, these values converge to the optimal Q-values $Q^*(s, a)$.

The goal of Q-Learning is to find the optimal policy $\pi^*$, which dictates the best action to take in each state to maximize the expected cumulative reward. Once the Q-values have converged, the agent can derive the optimal policy by choosing the action with the highest Q-value ($Q^*$) for each state.

**Results**

From the presented results shown in Figure 8, it is evident that the agent has failed to capture meaningful patterns from the data. Despite employing various reward functions, it always assigns all the traffic to a single class. This issue could stem from several factors. One significant reason is that the model is too simple compared to the complexity of the data. For this reason, we develop a more complex approach, named deep Q-Learning in the next section.

### 5.2.2 Deep Q-Learning

**Theory**

Deep Reinforcement Learning, specifically Deep Q-Learning in our case, extends the Q-Learning algorithm by using deep neural networks to approximate the Q-value function, enabling it to handle high-dimensional state-action spaces. Traditional Q-Learning struggles with large and continuous spaces due to the curse of dimensionality, where the number of possible states/actions increases exponentially with dimensionality, making it computationally expensive to maintain and update Q-values accurately. Deep Q-Learning addresses this limitation by leveraging deep neural networks to approximate the Q-values.

By representing the Q-function with a deep neural network $Q(s, a; \theta)$, Deep Q-Learning can learn from raw sensory inputs and directly output Q-values for each action. This approach not only improves the scalability of Q-Learning to high-dimensional problems but also enhances its ability to learn intricate representations of the environment's dynamics and rewards.

The training process in Deep Q-Learning involves minimizing the following loss function:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right]$$

where $\mathcal{D}$ is the experience replay buffer, $\theta$ are the parameters of the Q-network, $\theta^-$ are the parameters of a target network, $r$ is the immediate reward received after taking action $a$ in state $s$, $s'$ is the next state after taking action $a$, and $\gamma$ is the discount factor.

**Results**

This model also did not capture any patterns from the data, as indicated in Figure 9. Next, we introduce an alternative method designed to enable the model to learn and identify patterns within the training dataset.

### 5.2.3 Human in the training loop

The human intervention occurs during the second phase of training, specifically between episodes 600 and 800. We introduced human feedback at this stage for several reasons. Initially, during the early stages of training, the model showed no signs of learning from the data. Towards the end of training, it becomes challenging to steer the model towards correct classifications, as it has already solidified its learning based on the predefined reward function. Therefore, during training, when the model attempts to classify traffic and if its confidence in the prediction falls below a specified threshold, it prompts the human for input on the correctness of the prediction. Depending on whether the prediction is correct or not, the human provides updated rewards. For correct predictions, the model receives a positive reward, whereas different incorrect predictions receive a negative reward. The specific reward values provided by the human are based on the kind of the class and the predicted value. This step is crucial for several reasons, especially since some attacks, such as nmap, are incremental steps in larger attacks like the Hydra attack.

#### Result

This method did not show any significant difference compared to the previous approach, as indicated in Figure 10, instead, it presents numerous disadvantages. Firstly, it is highly sensitive to the threshold defined for the confidence level. Additionally, training becomes challenging and time-consuming, as the model constantly requires rewards, necessitating immediate correct answers from humans. This makes it difficult to tune hyperparameters effectively over time. Moreover, it is very essential to have human experts in the 5G domain who can provide accurate rewards based on predictions, correct answers, and the stage of training. Therefore, in the subsequent sections, we will explore other models and approaches that have demonstrated good results in similar use cases to ours.

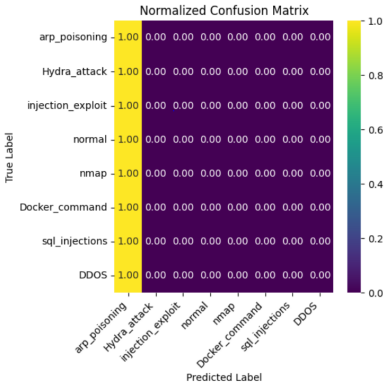


Figure 8: Confusion Matrix: Q-Learning

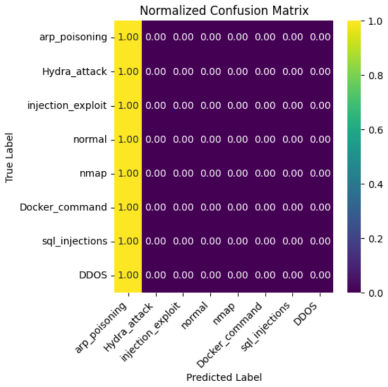


Figure 9: Confusion Matrix: Deep Q-Learning

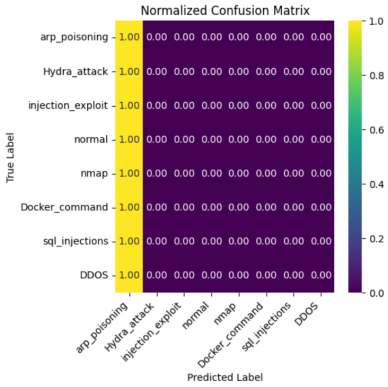


Figure 10: Confusion Matrix: Human-in-the-Loop

## 5.3 Supervised Learning

In the context of anomaly detection within 5G campus networks, models such as Random Forest and XGBoost present supervised learning tools for identifying and classifying anomalous network behavior. While unsupervised learning methods are valuable for detecting previously unseen anomalies without requiring labeled data, supervised learning offers advantages when labeled datasets are available, even if limited. These models learn from the patterns and relationships between features and labels in the training data to predict the output of new, unseen data. By using supervised learning models for the anomaly detection of 5G campus networks, the capability to respond to known types of attacks with high precision can be enhanced. The subsequent sections will dive into the theoretical basis and empirical results of Random Forest and XGBoost for this purpose. Before delving into the Random Forest and XGBoost models, we will briefly examine the performance of a Deep Neural Network (DNN) model, previously implemented by [2] in section 2.1, using our new dataset. This will provide a baseline comparison for the effectiveness of the DNN in our current context.

### 5.3.1 Deep Neural Network (DNN)

#### Theory

The DNN model used for this experiment is similar in architecture to the one described by K. Blachut, with ReLU activation functions for the hidden layers and a softmax activation function for the output layer. The loss function used was categorical cross-entropy, and the optimizer was Adam. However, it is important to note that while the method (DNN) is the same, the actual code implementation used in this experiment differs from the one originally used by K. Blachut.

### Results

The DNN model was tested on our dataset to verify its robustness and generalizability. This model achieved a precision and recall rate of 96% and 96%, respectively, with an F1-score of 95%. The confusion matrix in Figure 11 shows that the model effectively distinguishes between attack and normal traffic. However, similar to the other models, it faces challenges in correctly classifying certain attacks, particularly injection exploits and DDOS attacks. This could be attributed to the imbalance in the dataset or the complexity of the attack patterns.

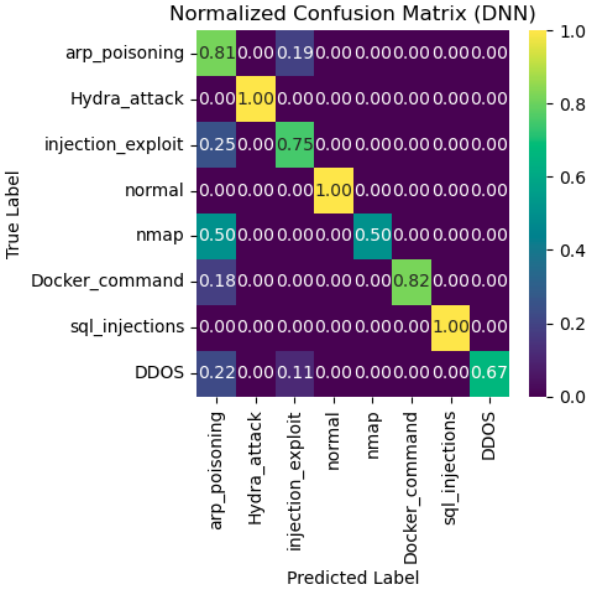


Figure 11: Confusion Matrix: DNN

In summary, while the DNN model achieves high precision and recall, there is always room for improvement, particularly in reducing the misclassification rates further. This underscores the importance of exploring other models like Random Forest and XGBoost, which are detailed in the subsequent sections.

#### 5.3.2 Random Forest

### Theory

Random Forest for classification, shown in Figure 12 is an ensemble learning technique that constructs multiple decision trees to classify data points based on majority voting. Each tree is trained using a different bootstrap sample from the training dataset, introducing randomness by resampling with replacement, known as bagging. Furthermore, at each node in the decision tree, a random subset of features is selected to determine the best split, rather than considering all features. This approach reduces the correlation among trees and enhances the overall model's ability to generalize, preventing overfitting. During prediction, each tree independently classifies a given input, and the class with the most votes across all trees is chosen as the final prediction.

### Results

This model has achieved a very high precision and recall rate of 95%. The confusion matrix in Figure 14 shows that the model effectively distinguishes between attack and normal traffic, with no misclassifications of normal traffic as attacks or vice versa. However, the model still struggles with correctly classifying certain attacks, particularly nmap and ARP poisoning. This could be due to the large number of classes and the small number of samples in some categories. That's why in the next section, we will use another model named XGBoost, known for its performance when dealing with these problems in the datasets .

### 5.3.3 XGBoost

#### Theory

The XGBoost short for Extreme Gradient Boosting, shown in Figure 13 , is a powerful supervised learning algorithm known for its efficiency and performance in structured/tabular data problems. It belongs to the family of gradient boosting algorithms, which iteratively combines weak learners (usually decision trees) to form a strong predictive model. XGBoost improves upon traditional gradient boosting methods by incorporating regularization techniques and a more efficient algorithm design, making it highly scalable and robust. The core principle behind XGBoost is to sequentially fit new models to provide a more accurate estimate of the target variable, focusing on instances where the previous models performed poorly.

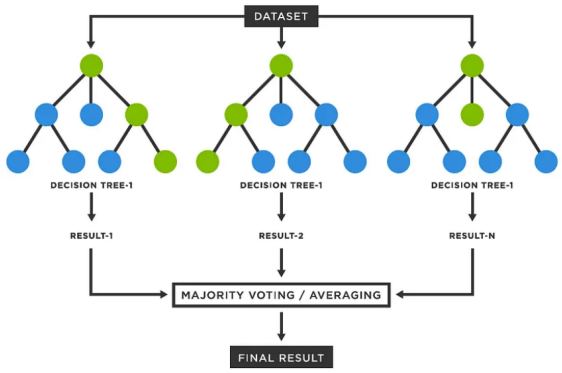


Figure 12: Random Forest Classifier

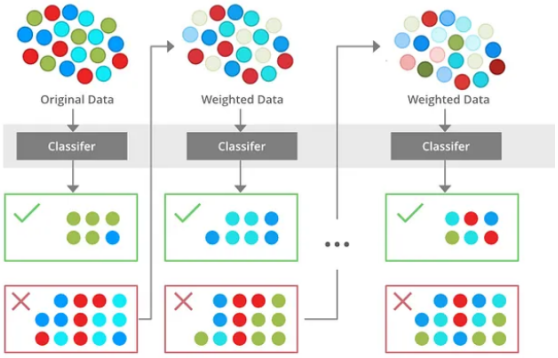


Figure 13: XGBoost Classifier

#### Result

XGBoost Classifier clearly outperforms all the discussed models in previous sections with a precision and recall rate of 98%. The confusion matrix, as shown in Figure 15, demonstrates that the model accurately distinguishes between normal traffic and attacks, with no instances of misclassifying normal traffic as attack or vice versa. Furthermore, the model effectively assigns most of the attack traffic to their correct classes.

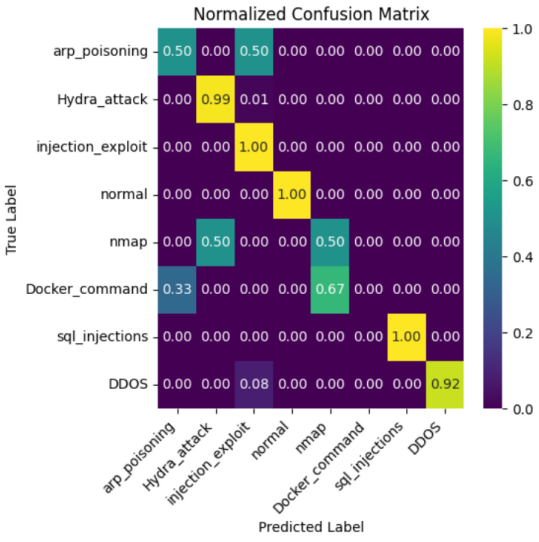


Figure 14: Confusion Matrix: Random Forest Classifier

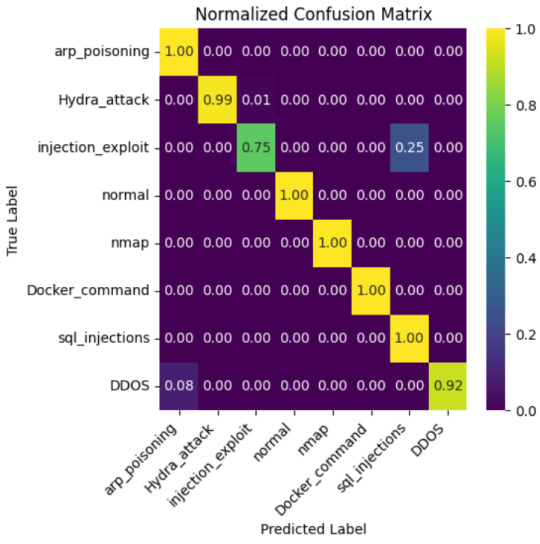


Figure 15: Confusion Matrix: XGBoost Classifier

### 5.3.4 Active Learning in Supervised Learning

#### General Idea and Motivation

In the context of anomaly detection within 5G campus networks, the models such as DNN, Random Forest, and XGBoost have already demonstrated very good performance. However, the question arises whether their performance can be further improved through active learning. Active learning is a machine learning approach where the model is trained iteratively, with the model selecting the most informative samples from an unlabeled dataset to be labeled by a human expert. This process aims to improve the model's performance with fewer labeled samples, thus potentially enhancing its generalizability and robustness.

The general idea of active learning in supervised learning is illustrated in Figure 16. The process starts with an initial labeled dataset used for training. The trained model then goes through a validation phase where it identifies samples with high uncertainty. These samples are then presented for human feedback to get labeled, after which they are included in the training dataset for the next iteration. This cycle continues, with the goal of improving the model's performance with each iteration.

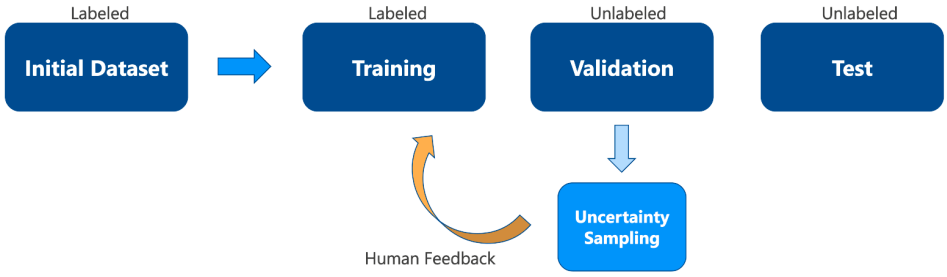


Figure 16: Active Learning in Supervised Learning: General Idea

#### Actual Dataset and Approach

For this experiment, the modAL package was used to implement active learning. The dataset used is detailed in Figure 17. The dataset was divided into 60% for training, 20% for validation, and 20% for testing. One significant challenge faced was the small size of the dataset for several attack classes, which became even smaller when split into training, validation, and test sets. This issue is highlighted in the actual dataset distribution.

| Class | Number |
|---|---|
| Normal | 776 |
| ARP Poisoning | 46 |
| Hydra Attack | 191 |
| Injection Exploit | 11 |
| Docker Command | 26 |
| SQL Injection | 17 |
| DDOS | 23 |
| nmap | 6 |
| **Total** | **1096** |

Initial Dataset

| Class | Number |
|---|---|
| Normal | 466 |
| ARP Poisoning | 28 |
| Hydra Attack | 114 |
| Injection Exploit | 7 |
| Docker Command | 16 |
| SQL Injection | 10 |
| DDOS | 13 |
| nmap | 4 |
| **Total** | **658** |

Training

| Class | Number |
|---|---|
| Normal | 155 |
| ARP Poisoning | 9 |
| Hydra Attack | 38 |
| Injection Exploit | 2 |
| Docker Command | 5 |
| SQL Injection | 4 |
| DDOS | 5 |
| nmap | 1 |
| **Total** | **219** |

Validation

| Class | Number |
|---|---|
| Normal | 155 |
| ARP Poisoning | 9 |
| Hydra Attack | 39 |
| Injection Exploit | 2 |
| Docker Command | 5 |
| SQL Injection | 3 |
| DDOS | 5 |
| nmap | 1 |
| **Total** | **219** |

Test

Figure 17: Active Learning: Actual Dataset

Since we have too few data samples for the attack class nmap, resulting in only one sample each in the validation and test datasets when we do the split, we decided to leave out this class and move forward with the remaining seven classes. This decision helps in focusing on classes with a more substantial number of samples for more reliable validation and testing.

### Results

The results of applying active learning to the three supervised learning models (Random Forest, XGBoost, and DNN) are presented in Figure 18. The performance metrics show that while the models achieved high precision, recall, and F1-scores, there was no significant improvement compared to the models without active learning. In the case of XGBoost, the performance slightly declined. This suggests that active learning did not enhance the model's performance and, in some cases, might have been counterproductive.

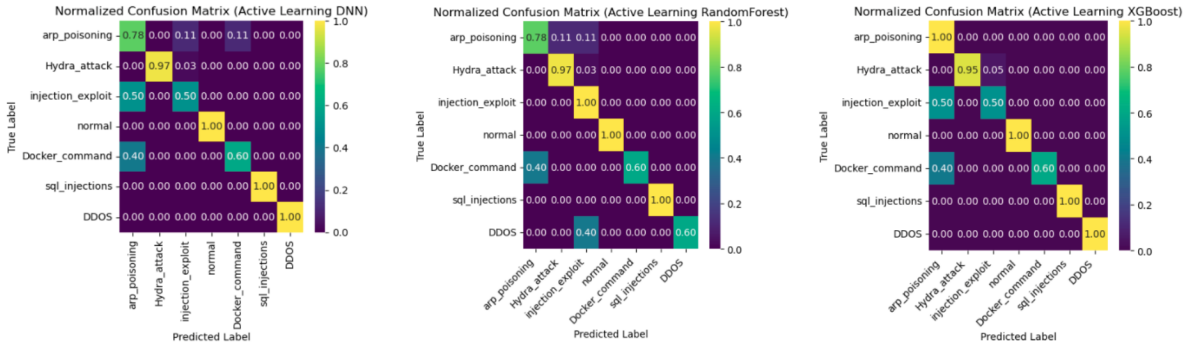


Figure 18: Active Learning: Results

### Interpretation of Results

There was no significant improvement in model performance, which can be attributed to several factors:

- **Small Dataset Size**: Splitting the dataset into three different sets for training, validation, and testing resulted in very few data samples for certain attack classes, leading to insufficient data for effective learning and validation.
- **Active Learning Redundancy**: Active learning may be redundant in supervised learning when the initial labeled dataset is already sufficient for training. The uncertainty sampling method used might not have provided significantly more informative samples than those already given in the training set.

These factors indicate that while active learning is a promising approach, its effectiveness is highly dependent on the dataset size and the nature of the data. For small datasets with high-class imbalances, traditional supervised learning approaches might be more effective without the additional complexity of active learning.

**Potential in Reinforcement Learning**: Active learning may work better in reinforcement learning due to its dynamic and continuous learning nature. In RL, the agent continuously interacts with the environment and learns from these interactions. Active learning can help focus on the most informative experiences, potentially improving exploration efficiency and overall learning performance. However, our current static dataset might not be ideal for demonstrating the full potential of active learning in RL.

# 6 SIEM

Security Information and Event Management (SIEM) is a cybersecurity tool that plays a crucial role in maintaining the security of a network by monitoring traffic and identifying potential threats with given algorithms or patterns. In our application it analyzes network traffic and alerts us when a cyber attack is detected.

Unlike industrial SIEMs that offer real-time analysis, our system processes network traffic data statically by receiving a file containing the data to be analyzed and using a machine learning algorithm to detect a potential cyber-attacks. This approach allows us to leverage advanced analytics to detect complex attack patterns that might not be immediately detected.

In Figure 19, our current approach towards a SIEM is illustrated. On the left side of the interface, users can upload a CSV file containing the entire network data and input their phone number to receive SMS notifications. If a cyber attack is detected, the user will receive a message stating, "A cyber attack was detected. Please look into that." Conversely, if no cyber attack is found, the message will read, "No cyber attacks were detected. It seems like normal traffic." On the right side, after providing the necessary information, users can view a count of normal traffic instances and detected cyber attacks, along with the model's confidence levels for each prediction. In this application, we utilize the XGBoost model for detection.

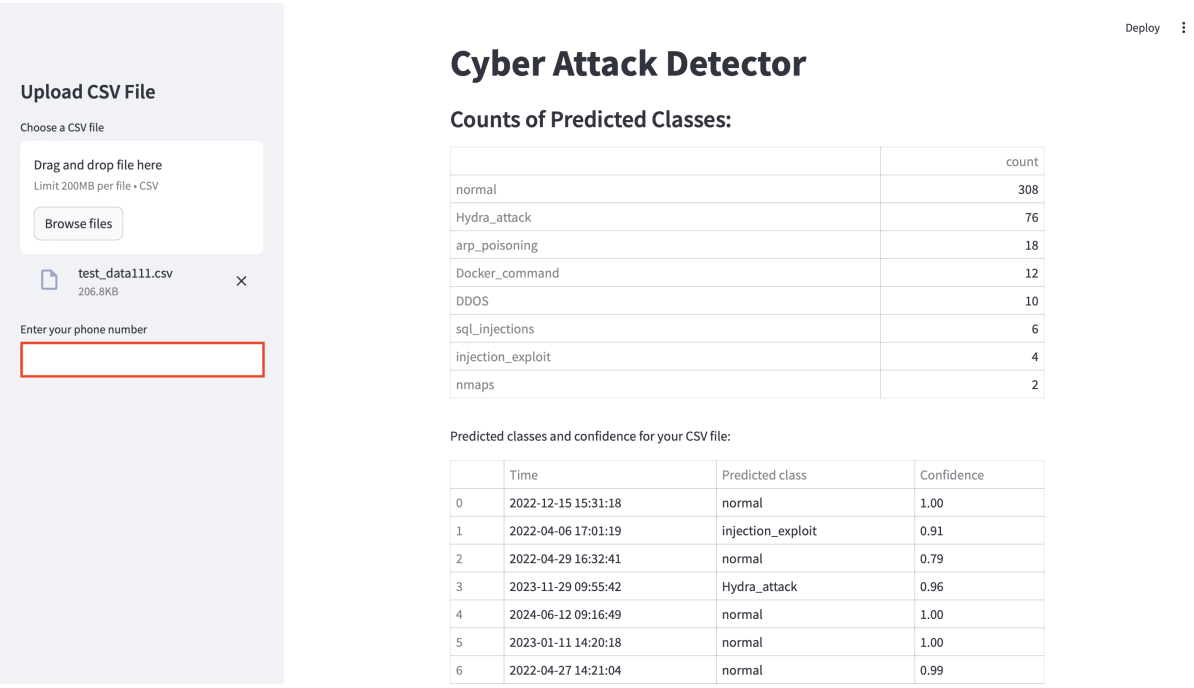


Figure 19: Our current approach towards a SIEM

# 7 Conclusion

In this project we developed and tested machine learning-based approaches for detecting cyber attacks within 5G network traffic. By configuring a 5G network environment and generating diverse traffic data through simulated attacks, we were able to train and evaluate various machine learning models. Our findings indicate that models such as XGBoost and deep neural networks demonstrated high precision and recall in anomaly detection. Additionally, the model was integrated in a Security Information and Event Management (SIEM) system which sends alerts on potential threats based on the given network traffic .

While we tested a lot of various approaches, there are several areas for future work. It could involve evaluating the performance of supervised learning models on completely unseen attack scenarios to better understand their generalization capabilities. Moreover, our unsupervised learning models could be enhanced by integrating active learning techniques, allowing the models to iteratively refine their accuracy by querying the most informative samples. Additional work could furthermore focus on developing even more sophisticated reinforcement learning models. Applying active learning within reinforcement learning could also provide valuable insights and potentially improve model performance.

Concerning the SIEM, an important advancement would be enabling the system to analyze network traffic data in real-time. This real-time capability would allow for immediate detection and response to potential cyber attacks. Currently, our SIEM system only reports potential attacks after they have occurred. Implementing proactive countermeasures to respond to detected threats in real-time would significantly enhance the system's utility and effectiveness in protecting the network.

# References


[1] Mohiuddin Ahmed, Raihan Seraj, and Syed Mohammed Shamsul Islam. The k-means algorithm: A comprehensive survey and performance evaluation. *Electronics*, 9(8), 2020.

[2] Katarzyna Blachut. *AI-based Open-Source SIEM for cyber threat detection and response.* iABG, 2022.

[3] Mariah St. John. Cybersecurity stats: Facts and figures you should know. *Forbes*, 2024.

[4] Kali Linux. *Kali Linux Tools, Tool Documentation, https://www.kali.org/tools/hydra/.* Kali, 2020.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[6] Fionn Murtagh and Pedro Contreras. Algorithms for hierarchical clustering: an overview. *WIREs Data Mining and Knowledge Discovery*, 2(1):86–97, 2012.

[7] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 2nd edition, 2018.

[8] Chang-Hsian Uang, Jiun-Wei Liou, and Cheng-Yuan Liou. Self-organizing reinforcement learning model. In Jeng-Shyang Pan, Shyi-Ming Chen, and Ngoc Thanh Nguyen, editors, *Intelligent Information and Database Systems*, pages 218–227, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[9] Nicolas Zapf. *Detection and Response in an AI-based SIEM for 5G campus network.* iABG, 2023.