

**A**  
**Thesis**  
**ON**  
**AI powered cyber threat detection and response system**  
**By**

Mridul Goyal

210C2030141

*Prepared in the partial fulfilment of the*

Practice School III Course

**BML Munjal University Gurugram, Haryana**

**A Practice School III Station of**



**BML MUNJAL UNIVERSITY**

**(July 2025)**

## CERTIFICATE

This is to certify that Practice School Project of **Mridul Goyal** titled **AI powered cyber threat detection and response system** is an original work and that this work has not been submitted anywhere in any form. Indebtedness to other works/publications has been duly acknowledged at relevant places. The project work was carried during **11<sup>th</sup> February 2025** to **22<sup>nd</sup> July 2025** in **BML Munjal University**

Signature of Faculty Mentor	Signature of Industry Mentor
Name: Dr. Abhishek Jain	Name:
Designation: Associate Professor	Designation:
Affiliation	Affiliation

## JOINING REPORT



BML Munjal University

### Thesis - PS 3 Application

The Director,  
Career Guidance & Development Center  
BML Munjal University, Gurgaon

Date: 08-02-2025

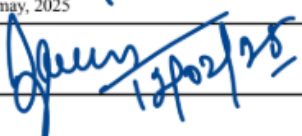
Subject: Request for issuing a recommendation letter to an organization for undergoing summer internship through my own efforts.

Dear Sir/Madam,

I, \_\_\_\_\_ Mridul Goyal \_\_\_\_\_ Registration No. \_\_\_\_\_ 210C2030141 \_\_\_\_\_

Course \_\_\_\_\_ B.Tech \_\_\_\_\_ CSE \_\_\_\_\_ Batch \_\_\_\_\_ 2021-2025

will be organizing my summer internship project on my own, for which I require a recommendation letter from the University, addressed to:

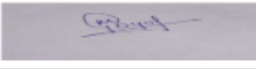
Name of the Faculty Mentor	Dr. Abhishek Jain
Topic Name	AI for Threat Detection and Incident Response
Field of the Topic	Cyber Security/Data Science
Faculty E-mail Id	<a href="mailto:abhishek.jain@bmu.edu.in">abhishek.jain@bmu.edu.in</a>
Faculty Contact Number	+91-9466852190
Mode of internship (Virtual / University / Hybrid)	Virtual
PS3 (Thesis) Start date	12 Feb, 2025
PS3 (Thesis) End date	30 may, 2025
Faculty Mentor Signature	

I am enclosing the details of my correspondence with the concerned person at the organization.

Please approve my request for issuing of letter for PS3 (Thesis) from the Career Guidance & Development Center. Also, my name may be removed from the list of students applying for internship through the University and my resume may not be forwarded to any organization, henceforth. However, if I get shortlisted through the University, in the interim, I will undertake the placement organized by the University, unconditionally.

Thanking you.

Yours sincerely,

Name of Student	Mridul Goyal	Recommended By	Dr. Abhishek Jain
Signature of the Student		Signature	
Mobile No.	+91-7015802542	Name	

## Table Of Contents

LIST OF FIGURES .....	6
LIST OF TABLES .....	6
ABBREVIATIONS .....	7
ACKNOWLEDGEMENT .....	8
1. Abstract .....	9
2. Introduction .....	10
2.1 Background .....	10
2.2 Problem Definition .....	10
2.3 Project Objectives .....	10
2.4 Project Scope .....	11
3. Literature Review .....	11
3.1 Signature-Based Threat Detection Systems .....	12
3.2 Machine Learning for Anomaly Detection .....	12
3.3 Automated Response Systems and Alerting Mechanisms .....	13
3.4 Hybrid Detection Approaches .....	13
3.5 Identified Gaps and Project Contribution .....	13
4. Dataset Description .....	14
4.1 Dataset Overview .....	15
4.2 Attack Types Included .....	15
4.3 Preprocessing Steps .....	15
4.4 Dataset Justification .....	15
5. System Design and Architecture .....	16
5.1 Overall System Architecture .....	17
5.2 Frontend Design .....	19
5.3 Backend Design (Flask) .....	19
5.4 Dataset and Threat Detection Flow .....	19
5.5 Alerting and Automation Pipeline .....	20
5.6 Data Flow Diagram .....	21
6. Implementation Details .....	22
6.1 Environment Setup .....	22
5.2 Key Technologies and Libraries .....	23
6.3 Implementation Results .....	24
7. Testing and Validation .....	28
7.1 Testing Strategy .....	28
7.2 Unit Testing .....	28

7.3 Integration Testing .....	28
7.4 Boundary Conditions and Stress Testing .....	29
8. Results and Discussion.....	30
8.1 Achieved Functionalities.....	30
8.2 Security Enhancements and Privacy Aspects .....	35
8.3 Performance Considerations .....	35
8.4 Limitations .....	36
9. Conclusion and Future Work .....	37
9.1 Conclusion .....	37
9.2 Major Learning Outcomes .....	37
9.3 Future Work .....	37
10. References .....	39
11. Plagiarism Report.....	40

## LIST OF FIGURES

Figure Number	Figure Name	Page Number
Figure 4.1	Dataset Screenshot 1	16
Figure 4.2	Datset Screenshot 2	16
Figure 5.1	Overall System Architecture	18
Figure 5.6	Data Flow Diagram	21
Figure 6.1.1	Requirements.txt	22
Figure 6.1.2	Project Structure	22
Figure 6.1.3	Model Processing using important features	23
Figure 6.3.1	Comparison between models	24
Figure 6.3.2	Model selection and saving	24
Figure 6.3.3	Feature Importance	25
Figure 6.3.4	Heart of predictions	25
Figure 6.3.5	Csv upload Screenshot	26
Figure 6.3.6	Detected Threats	26
Figure 6.3.7	Twilio Integration	26
Figure 6.3.8	Discord Integration	27
Figure 8.1.1	Threat Detection on Dashboard	30
Figure 8.1.2	Packet Sniffing in terminal	31
Figure 8.1.3	Incoming sms alert	31
Figure 8.1.4	Discord alert	31
Figure 8.1.5	Simulated actions	32
Figure 8.1.6	Dashboard Screenshot 1	32
Figure 8.1.7	Dashboard Screenshot 2	33
Figure 8.1.8	Dashboard Screenshot 3	33
Figure 8.1.9	Dashboard Screenshot 4	34
Figure 8.1.10	Dashboard Screenshot 5	34
Figure 8.1.11	Logging	35

## LIST OF TABLES

Table Number	Table Name	Page Number
Table 4.2	Attack Categories	15
Table 5.2	Components and Technologies	21
Table 6.2	Test Cases	26

## ABBREVIATIONS

Abbreviation	Full Form
AI	Artificial Intelligence
ML	Machine Learning
MLP	Multi-Layer Perceptron
NB	Naive Bayes
QDA	Quadratic Discriminant Analysis
DPI	Deep Packet Inspection
CSV	Comma-Separated Values
UI	User Interface
UX	User Experience
API	Application Programming Interface
IP	Internet Protocol
HTTP	HyperText Transfer Protocol
IDS	Intrusion Detection System
CICIDS2017	Canadian Institute for Cybersecurity – IDS 2017
SMS	Short Message Service
DDoS	Distributed Denial of Service
FTP	File Transfer Protocol
SVM	Support Vector Machine
ROC	Receiver Operating Characteristic
SHAP	SHapley Additive exPlanations
CPU	Central Processing Unit
IP Block	IP Quarantine / Blacklist Mechanism
CRUD	Create, Read, Update, Delete

## **ACKNOWLEDGEMENT**

I would like to express my heartfelt gratitude to all those who have supported and guided me throughout the course of my thesis. I am deeply thankful to the Vice Chancellor and the Dean, School of Engineering and Technology, for fostering an environment conducive to research and for providing the necessary resources and encouragement. I am also grateful to the Project Coordinator for ensuring seamless coordination and making this project possible through consistent support and organization.

I would especially like to thank Dr. Abhishek Jain for his invaluable guidance, constructive feedback, and unwavering support, which have significantly contributed to the development and direction of this work. I extend my sincere thanks to the faculty members of BML Munjal University for their teaching and mentorship, which have been instrumental in my academic journey. Finally, I am appreciative of the encouragement and assistance from colleagues, friends, and professionals beyond the university who, in various ways, helped me complete this project successfully.



## 1. Abstract

As such network traffic and the number of internet-connected devices have grown exponentially, predisposing the contemporary organizations to a broad range of cyber-attacks. Conventional signature-based or old-fashioned intrusion detection systems usually cannot detect new or modifying hazards, and more so in real time. In order to overcome this shortcoming, the project will be created as a smart, AI-based threat detection and responsive architecture, including machine learning to classify the traffic patterns of networks.

CIC-IDS2017 by Newhouse is a commonly used benchmark in network intrusion detection research and is used in the project with extensive preprocessing, attack-wise data separation, features selection and class balancing to be used to prepare training data.

When the system detects the threats, it is possible to automatically act based on threats by creating logs, simulated IP isolation, and alerting through Discord, email, and SMS via external APIs. The emulation of live traffic is also supported in the system through packet sniffer simulator that helps in the detection and response within the real-time detection and response cycles. All in all, the project is a modular and extendable framework of intelligent cybersecurity analytics, which can be used in case of academic research, education demonstration, and prototype in the real world. It shows how AI and automation can be synergistically used to increase the scalability, speed and accuracy of network defense systems.

## **2. Introduction**

### **2.1 Background**

Cybersecurity is the most crucial matter in the modern digital connected world. The more organizations use online services and cloud infrastructure the more exposed they are to various categories of cyber-attacks such as Distributed Denial-of-Service (DDoS), brute-force, and reconnaissance methods of attack port scanning.

Completely on the basis of stematic standards or matching signatures, according to the well-known structure of Intrusion Detection Systems (IDS), customarily turns out to be inefficient at distinguishing contemporary and advanced and shifting risks. Such systems can fail to identify new (zero-day) attacks and could generate large false positive rates because of strict pattern matching. Moreover, they are not suitable in high-speed and high-volume networking in their inability to respond in real-time and low level of automation.

The development of AI and machine learning (ML) recently have presented dynamic, adjustable methods of cybersecurity. ML-empowered systems can be suitable to detect the anomalous or malicious behavior based on learning a pattern of behavior observed in the past and generalizing the findings to unseen behavior. In this project, this is leveraged by integrating machine learning-based detection with a responsive Flask web interface and alert automation to produce a complete and real-time system to monitor and mitigate cyber threats.

### **2.2 Problem Definition**

The current security systems have a number of problems encountering real-time and advanced threats:

- Static signature-based systems are unable to spot unknown or polymorphic attacks.
- Slow and dependent on people updates of threat rules.
- Most conventional systems do not have the option of actionable automation- no immediate action against it is exercised when it is detected.
- No convenient interfaces to work in the run-time mode displaying threats and managing them.

Therefore, it is highly necessary to have a system, which has an ability to detect, log, visualize and react to threats in real-time with minimized human interference and maximized accuracy.

### **2.3 Project Objectives**

The project is developed in the context of the following fundamental objectives:

- Incorporate the use of machine learning to build a real time threat detection system with models trained on real world network traffic characteristics.
- Prepare and organize information in the CIC-IDS2017 dataset according to categories of attacks, and select the most important features in each type of attack.
- Compare several supervised classifiers ( MLP, Naive Bayes, QDA ) that are trained on evenly distributed data.
- Code Flask-driven web application to:
  - Send and decode network logs
  - Packet traffic simulation
  - Present threats in an easy to act upon status
  - Filter, sort and export incidents
- Bridge in real-time alerting such as:
  - Threats to persistent CSV files logging
  - Twilio to send SMS alert
  - Using Discord webhooks to get real-time monitoring
  - Modeling IP quarantines and firewalls responses

## 2.4 Project Scope

The project is a reconfigurable and extensible prototype that demonstrates the ability of machine learning techniques to improve network security monitoring. This project will cover the following scope:

- Structured data (CSV) based threat detection as opposed to raw packet (PCAP) inspection.
- Testing the model in real time by simulating the trafficking of the live traffic through the creation of synthetic packets.
- Web interface through Flask, so that the users could upload logs, see graphical displays of attacks, and show threat simulations.
- Automated alerting and response, confidence based and category based attack.
- Comparing the accuracy, precision, recall of a model.

The focus of this system is on academic, educational, and prototyping scenarios instead of its enterprise deployment, but its components can be scaled to use in the real world.

## 3. Literature Review

### 3.1 Signature-Based Threat Detection Systems

The oldest and probably the most popular method in intrusion detection systems (IDS) is signature-based detection. These systems have a pre defined database that contains known patterns of attacks, commonly referred to as signatures and they are used to correlate incoming traffic against these known attack patterns. Tools such as Snort, Suricata and Bro (Zeek) are normally used to deploy signature-based detection in enterprise networks.

Although these systems are light and efficient toward the known threats, their most important drawback is that they do not detect zero-day attacks or mutated forms of known threats. The checking of pure signature matching is becoming unreliable in contemporary networks: attackers could simply obfuscate (slightly) the payloads or simply encrypt them, making detection of any form of attack almost impossible. In addition, manual updates of the set of rules are required frequently, adding overhead to maintenance.

Since cyber threats change at an impressive rate, snapshot-based systems being unable to implement reactive strategies effectively, behavior-based and data-driven means of detecting them have to be considered.

### 3.2 Machine Learning for Anomaly Detection

Machine learning (ML) brings a paradigm shift in the intrusion detection of networks in the sense it allows the models to learn over time, based on historical network traffic, and identify abnormal or malicious behavior without the need of defining rules. When trained over well-labelled datasets, supervised learning algorithms have performed well in the classification of attacks including Naive Bayes, Support Vector Machines (SVM), Random Forests, and Neural Networks (MLP).

Such datasets as CIC-IDS2017, NSL-KDD, and UNSW-NB15 offer various traffic situations such as DDoS, brute force, botnet, and reconnaissance attacks. These datasets contain benign and malicious flows commonly modeled utilizing the flow level characteristics including:

- Flow Duration
- Mean(Packet Length)/Std
- Bytes/s
- Packet Inter-arrival Time

These features can work in machine learning especially because they are numeric and high level abstractions of behavior in the traffic.

Furthermore, the most seekable attributes are selected by using the techniques of feature selection, e.g., mutual information, correlation ranking, and recursive feature elimination (RFE). This increases the accuracy of the model and saves time during the training process.

Nonetheless, there are instances where machine learning methods tend to be class-imbalanced, i.e, there is a huge number of benign samples than attack samples. To achieve this, methods like RandomUnderSampler or SMOTE (Synthetic Minority Oversampling Technique) are used to normalize those sets of data prior to training the model.

### 3.3 Automated Response Systems and Alerting Mechanisms

Although detection plays an important role, response mechanisms are what makes intrusion detection systems functional in a real-time sense. The traditional setups include alerts being sent or logged into the email system where they can be examined manually. Such delay can be expensive particularly where high-speed or volumetric attack is involved.

Contemporary studies focus more on incorporation of automated response modules that run prescribed sequences of actions when a threat is identified. They can include:

- Isolating of the IP address or device
- Firing on the firewall rule
- Emailing, texting or web hooking security personnel
- Filtering of network traffic on the basis of severity

Cloud-native solutions tend to work with SIEM (Security Information and Event Management) solutions like Splunk or Elastic Stack. Nevertheless, such solutions are normally costly and hard to implement.

Regarding the project, the system incorporates lightweight automation through sending SMS messages through Twilio, firing live notifications via Discord webhooks, and imitating such features as IP quarantine and firewall rules execution. It is appropriate in terms of academic demonstration, rapid prototyping purposes, and research testing based on these components.

### 3.4 Hybrid Detection Approaches

Hybrid models are also becoming popular to monitor the advantages and disadvantages of the signature- and machine-learning-based systems. These systems are a combination of:

- Known attacks engines
- Classifiers based on ML for perfecting or unified threats
- Voting or stacking of many classifiers (ensemble learning) is also used by some of these methods to amalgamate many classifiers to minimize variance, and increase robustness. Anomaly detection is also paired with behavioral baselining in others.

Nevertheless, hybrid systems have the issue of system complexity, computational expense, and explanation. Moreover, intuitive user interfaces or pipelines of automatic responses are rare in the hybrid systems.

### 3.5 Identified Gaps and Project Contribution

Based on the review of the existing techniques and tools, the following gaps were found:

- Inadequate modular systems that integrate real-time detection, automatic response and user-friendly interfaces
- Majority of large scale systems have a high cost of infrastructure or configuration

- Live testing provided by simulation based tools is not very popular and useful in educational and testing setting
- Poor connectivity of engines to perform actions or module response capabilities like SMS or geolocation visualization

The system completes the gaps by:

- Offering an end to end detection → alert → response pipeline
- Originating the traffic simulation capability on a real-time basis and with an adjustable severity
- Web-based Flask-constructed visualization of threats
- SMMBot for listening and SMS, Discord, and logs based alerts
- Recording of the incidents supported by the geolocation (latitude and longitude), time stamp, and confidence with the nature of threats
- It has been lightweight and open-source and flexible to alternate detection models

That makes the project a balanced and feasible example of the contemporary cybersecurity systems based on Artificial Intelligence.

#### **4. Dataset Description**

This project uses the CICIDS2017 dataset developed by the Canadian Institute for Cybersecurity at the University of New Brunswick. It is one of the most comprehensive

publicly available datasets for intrusion detection research and simulates real-world enterprise network traffic with labeled attack scenarios.

#### 4.1 Dataset Overview

- Name: CICIDS2017 (Canadian Institute for Cybersecurity – Intrusion Detection System 2017)
- Size: ~12 GB (multiple CSV files)
- Records: Millions of labeled traffic flow entries
- Features: 80+ network flow features including packet statistics, byte counts, protocol flags, flow durations, etc.
- Classes: Includes both benign traffic and 14 different attack types

#### 4.2 Attack Types Included

Table 4.2 Attack Categories

Category	Examples
Denial of Service (DoS)	DoS Hulk, GoldenEye, Slowloris, Slowhttptest
Distributed DoS (DDoS)	DDoS
Brute Force	SSH-Patator, FTP-Patator
Web Attacks	Brute Force, XSS, SQL Injection
Port Scanning	PortScan
Others	Botnet, Infiltration, Heartbleed

#### 4.3 Preprocessing Steps

Before training models, the dataset underwent multiple preprocessing phases:

- Attack Filtering: Splitting the large dataset into attack-specific CSV files like DDoS\_vs\_BENIGN.csv, etc.
- Label Encoding: Mapping class labels like "BENIGN" and "DDoS" to binary classes (0, 1).
- Undersampling: Balancing the dataset using RandomUnderSampler to avoid bias from class imbalance.
- Feature Selection: Identifying top features (like Bwd Packet Length Mean, Packet Length Std, etc.) using correlation and ML-based importance metrics.
- Storage Format: Processed CSV files (~300MB each) are stored in a uniform format for training and testing.

#### 4.4 Dataset Justification

The CICIDS2017 dataset is chosen for:

- High authenticity: Realistic traffic from enterprise-like environments
- Label diversity: Covers a wide range of modern cyberattacks

- Feature richness: Contains low-level and flow-level features suitable for ML
- Community validation: Widely accepted in academic and industry research

	Destination Port	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean	Fwd Packet Length Std
0	-0.401097	-0.379440	-0.010777	-0.009744	-0.064632	-0.008531	-0.235865	0.479238	-0.067529	-0.312913
1	1.930937	-0.326029	-0.056730	-0.046895	-0.004781	-0.032457	0.690723	-0.536980	1.506978	1.972851
2	-0.333292	-0.655631	-0.010123	-0.010387	-0.088198	-0.007591	-0.377384	-0.177020	-0.346490	-0.366976
3	-0.428196	-0.390415	-0.011691	-0.012064	-0.070710	-0.009201	-0.350847	-0.439754	-0.445006	-0.332946
4	-0.358097	-0.654460	-0.007449	-0.008357	-0.065506	-0.007532	-0.324424	0.449044	-0.143664	-0.366976

Fig 4.4.1 Dataset Screenshot 1

min_seg_size_forward	Active Mean	Active Std	Active Max	Active Min	Idle Mean	Idle Std	Idle Max	Idle Min	Label
1.005074	-0.099053	-0.109234	-0.137630	-0.073297	-0.282251	-0.080798	-0.284355	-0.273412	0
1.078168	-0.118177	-0.109978	-0.143077	-0.086475	-0.244274	-0.073967	-0.245803	-0.236290	0
-1.068350	-0.131771	-0.100491	-0.148812	-0.104465	-0.580707	-0.105787	-0.585977	-0.571490	0
-0.879153	-0.120297	-0.126787	-0.165393	-0.089440	-0.284261	-0.075630	-0.285737	-0.276032	0
-1.068350	-0.131771	-0.100491	-0.148812	-0.104465	-0.580707	-0.105787	-0.585977	-0.571490	0

Fig 4.4.2 Dataset Screenshot 1

## 5. System Design and Architecture



## 5.1 Overall System Architecture

The project is a light, but all-encompassing cybersecurity framework that represents an integration of machine learning, live packet sniffing, file-based analysis, and automated threat response in a single solution.

There are two main ways in which it operates:

1. Live Mode (Packet Sniffer): Constantly makes simulation of and examination of the network traffic.
2. File Upload Mode: It enables the hard disk to be uploaded with .csv traffic logs to detect the threats.

There are five large layers of system architecture:

1. Data Handling Layer: Can flush files or simulate real time trafficking.
2. Preprocessing & Inference Layer: Extracts features of choice and feeds the same into pre-trained ML models.
3. Detection and Logging Layer: Logs threats of the CSV files and timestamps, IPs and confidence scores.
4. Alerting & Response Layer: Alerts using SMS and Discord, and simulates increased IP isolation.
5. Visualization Layer: Visualizes threats in a Flask-built dashboard and an embedded map.

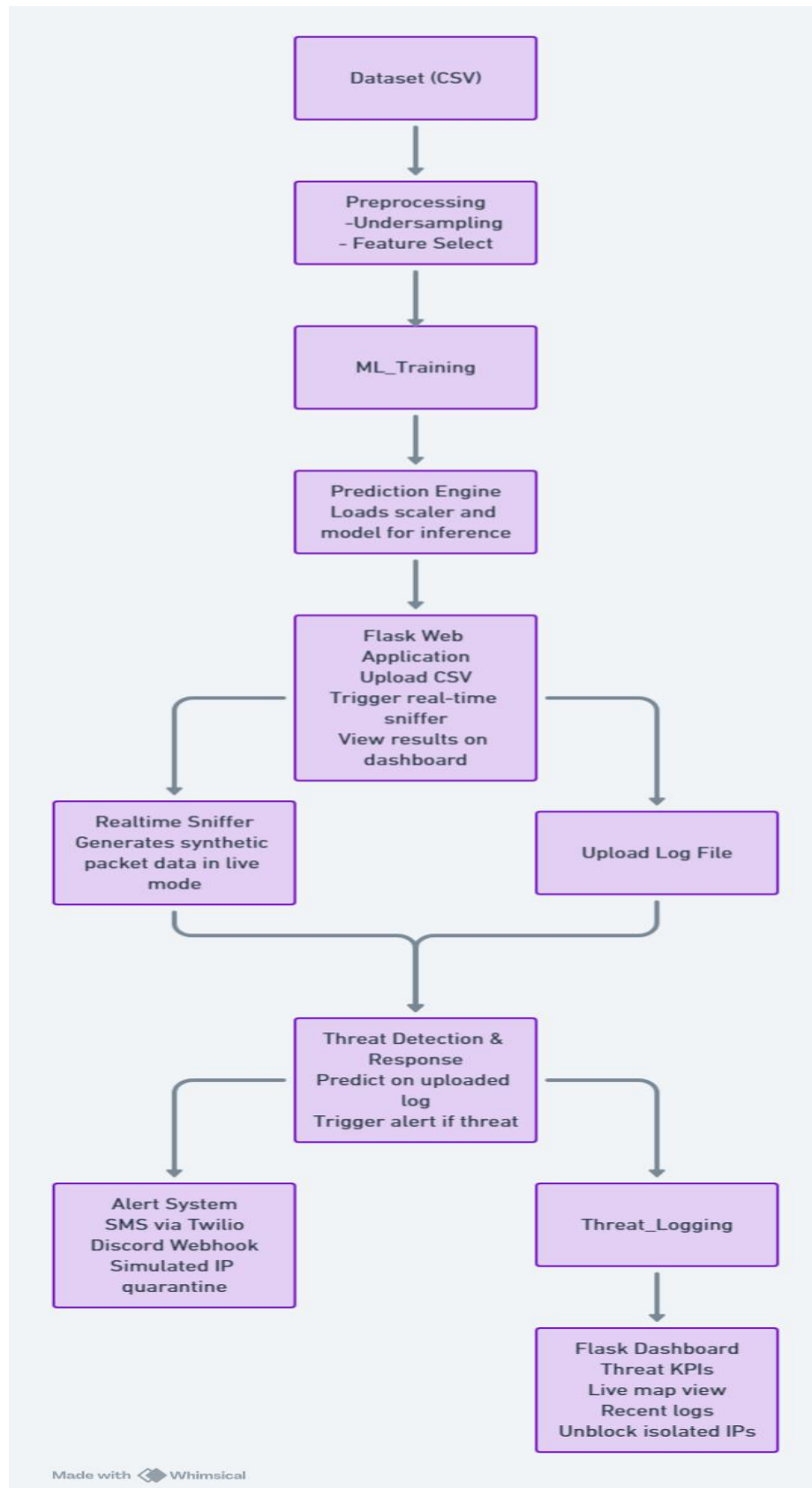


Fig 5.1 Overall System Architecture

## 5.2 Frontend Design

The front-end system is a responsive dashboard created in Flask with the standard web framework (HTML, CSS, Bootstrap). It includes:

- **Sidebar Navigation:** Provides the ability of the user to go to the dashboard, to export logs, to view embedded maps, or to log out.
- **Dashboard Metrics:**
  - Total threats identified
  - Dynamic isolated IPs
  - Last threat time
- **upload Data Section:** It enables the users to upload .csv files to be processed.
- **Sniffer Activation Button:** This activates the simulated packet generator.
- **Threat Table:** Updates automatically with the current listing of threats displaying timestamp, type, IP, confidence and action.
- **Threat Over Time Plot:** Information visualising the number and the seriousness of threats occurring.
- **Map Visualization:** Renders geo-tagged threats with the help of a Leaflet-based map.
- **Isolated IP Controls:** Allows users to read and unban IPs.
- **Filter & Export Logs:** The users can filter by the type of attacks and get log in the form of CSV.
- The whole interface can be used with Dark Mode to make reading easy in security operation center (SOC) settings.

## 5.3 Backend Design (Flask)

The backend of the project is written in Python using Flask. It manages:

- File upload handling and feature extraction
- Invoking the trained ML model for prediction
- Reading/writing to the CSV log files (incident\_log.csv, realtime\_incidents.csv)
- IP isolation simulation (in-memory blocklist)
- Triggering the real-time packet sniffer (as a subprocess)
- Calling third-party APIs for Twilio and Discord alerts
- Rendering all data into HTML templates using Jinja2

A minimal session system is included to support login via login.html.

## 5.4 Dataset and Threat Detection Flow

The system runs on the basis of the CIC-IDS2017 dataset that includes various types of attacks: DDoS, DoS, PortScan, Botnet, and Web attacks. The schematic is shown in the following manner:

1. raw CSV files are sieved according to the type of attack.
2. attack\_type\_vs\_BENIGN.csv are generated.
3. Undersampling is used to balance the labels of classes.
4. The feature selection is carried out to keep the best 3-4 flow features.
5. Model technologies (e.g. MLP model (mlp\_model.pkl)) are used on:

- Added CSVs through dashboard
- Real time generated synthetic packets

When there is the detection of (prediction = 1, confidence > threshold), a system responds.

### 5.5 Alerting and Automation Pipeline

After an identification of a threat:

- The IP is pulled out (simulated or uploaded file).
- The model confidence is computed into a threat score.
- The occurrence is recorded to:
  - uploaded files to be uploaded by library staff in incident\_log.csv
  - sniffed traffic: realtime\_incidents.csv
- The backend activators:
  - An SMS to the administrator (Twilio)
  - A Discord webhook of alert information
  - Fake firewall block or IP block action
  - Each IP is tagged with geo-location (randomized within a fixed area) to be visual tracked
- All the logs have:
  - Timestamp
  - Attack type
  - Confidence level
  - Detection source (flask/ sniffer)
  - Action taken
  - Latitude/Longitude

## 5.6 Data Flow Diagram

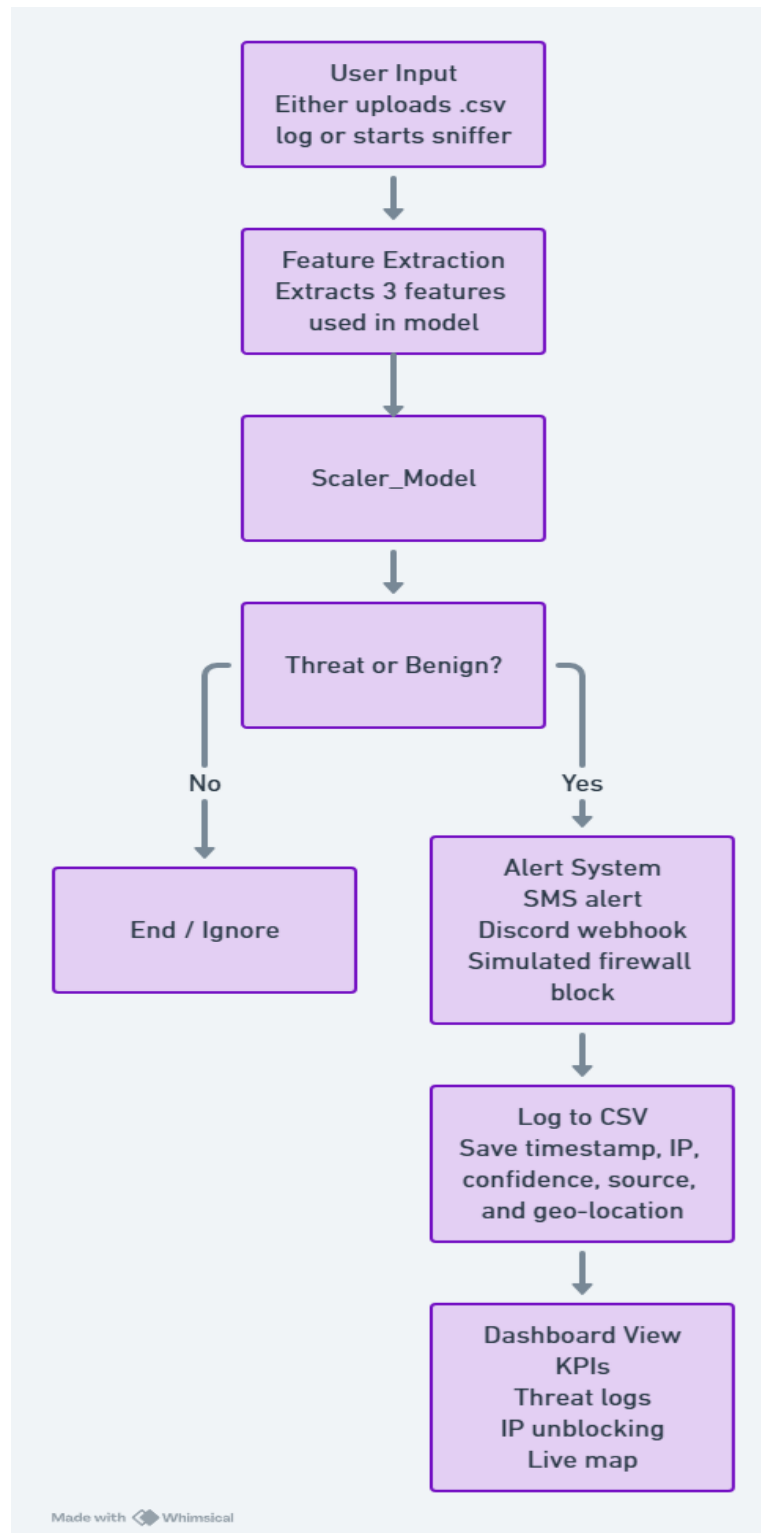


Fig 5.6 Data Flow diagram

## 6. Implementation Details

Here, we discuss how to put the project into practice, such as the environment for coding, main technologies, and detailed descriptions of the main modules' functions.

### 6.1 Environment Setup

The project was developed and tested on a local machine using the following configuration:

- **Operating System:** Windows 11 (64-bit)
- **Python Version:** 3.10+
- **IDE:** VSCode / Jupyter Notebook
- **Virtual Environment:** venv for dependency isolation

To install all required dependencies, a requirements.txt file was used. It includes:

```
Flask==2.2.2
pandas
numpy
scikit-learn
matplotlib
joblib
imblearn
twilio
discord-webhook
```

Fig 6.1.1 Requirements.txt

The folder structure was organized as follows:

```
project_root/
├── app.py                # Flask main server
├── dashboard.py          # Dashboard-related views
├── analyze_incidents.py  # Visualizations for dashboard
├── packet_sniffer.py     # Realtime sniffer simulation
├── MachineLearningSep.py # ML training pipeline
├── incident_handler.py   # Response and quarantine logic
├── incidents/
│   ├── incident_log.csv
│   └── realtime_incidents.csv
├── templates/
│   ├── index.html
│   ├── login.html
│   └── map_embed.html
├── static/
│   └── threat_map.html   # Geo map
├── mlp_model.pkl         # Trained model
├── scaler.pkl            # Trained scaler
└── attacks/              # Per-attack CSVs and graphs
```

Fig 6.1.2 Project Structure

- **Model Training Environment:** Google Colab was used to preprocess the permission dataset, perform one-hot encoding, train the Random Forest model, and export the trained pipeline for backend inference.

```
x_train_plot = x_train.drop(columns=['app_name'], errors='ignore')

from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=200, random_state=42)
model.fit(x_train_plot, y_train)

importances = model.feature_importances_
top_indices = importances.argsort()[::-1][:10]
```

Fig 6.1.3 Model Processing using important features

## 5.2 Key Technologies and Libraries

Table 6.2 Components and Technologies

Component	Technology / Tool Used
Frontend Template	HTML + Bootstrap (Jinja2 templating)
Machine Learning Models	MLP, Naive Bayes, Quadratic Discriminant Analysis
Dataset	CIC-IDS2017 (attack flows and benign traffic)
Preprocessing	Pandas, Scikit-learn, Imbalanced-learn (undersampling)
Alerts	Twilio (SMS), Discord Webhooks
Data Visualization	Matplotlib, Geo-coordinates plotting on map
Real-time Simulation	Synthetic flow generator in packet_sniffer.py
Persistence	CSV logging + IP quarantine state memory

### 6.3 Implementation Results

- **Machine Learning Pipeline:**
  - Individual classifiers trained for each of the 14 attack types (e.g., Bot, DDoS, DoS Hulk etc).

	Attack Type	Naive Bayes Accuracy	QDA Accuracy	MLP Accuracy
0	Bot	0.638021	0.832589	0.998870
1	DDoS	0.984610	0.980481	0.992264
2	DoS GoldenEye	0.984251	0.984238	0.996942
3	DoS Hulk	0.977031	0.976215	0.989537
4	DoS Slowhttptest	0.972274	0.974597	0.996782
5	DoS slowloris	0.981212	0.978324	0.997528
6	FTP-Patator	0.998758	0.998777	0.995455
7	PortScan	0.916335	0.916335	0.998157
8	SSH-Patator	0.998276	0.998788	0.998808

Fig 6.3.1 Comparison between models

- Best-performing model (usually MLP) exported using joblib for live use.

```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import joblib

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
scaler = StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)

model = MLPClassifier(hidden_layer_sizes=(50, 25), max_iter=500)
model.fit(X_train_scaled, y_train)

# Save model and scaler
joblib.dump(model, "mlp_model.pkl")
joblib.dump(scaler, "scaler.pkl")
```

Fig 6.3.2 Model selection and saving

- Feature selection performed using weight analysis.



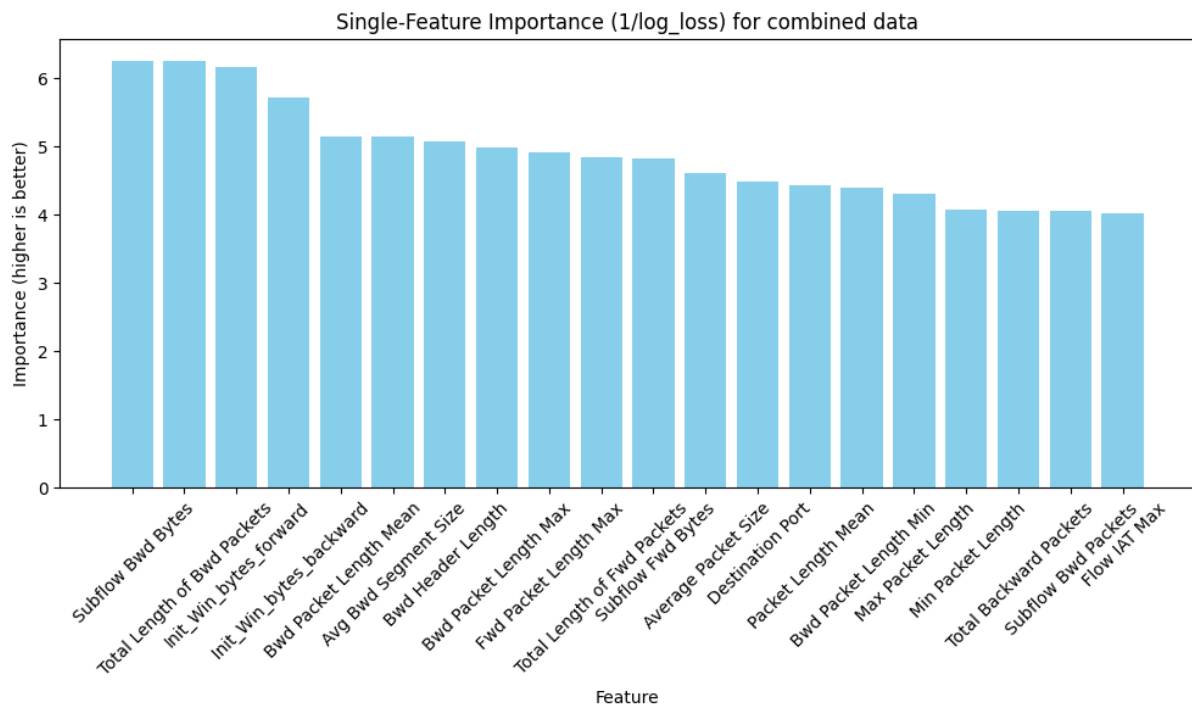


Fig 6.3.3 Feature Importance

- Resampled each dataset to handle class imbalance via RandomUnderSampler.
- **Real-Time Sniffer:**
  - packet\_sniffer.py runs a loop generating synthetic packets using numpy.random.
  - Selected features: Flow Duration, Bwd Packet Length Mean, Packet Length Std.
  - Each synthetic flow is passed to the scaler and prediction engine.
  - If classified as a threat with high confidence (> 0.4), response is triggered.

```
duration = np.random.normal(120000, 30000)
bwd_len = np.random.normal(1200, 200)
pkt_std = np.random.normal(150, 30)

X = pd.DataFrame([duration, bwd_len, pkt_std], columns=MODEL_FEATURES)
X_scaled = SCALER.transform(X)
prediction = MODEL.predict(X_scaled)[0]
confidence = MODEL.predict_proba(X_scaled)[0][1]
```

Fig 6.3.4 Heart of predictions

- **CSV Upload and Threat Analysis:**

- Users can upload traffic log files via the Flask dashboard.

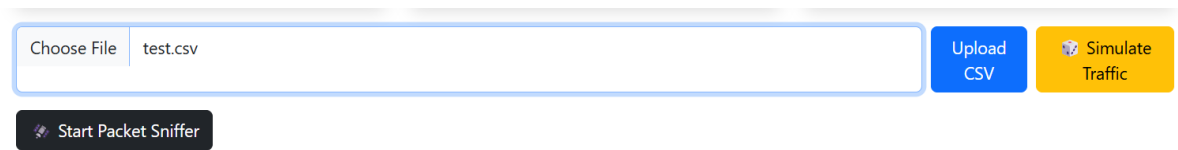


Fig 6.3.5 Csv upload Screenshot

- Uploaded data is parsed using pandas and passed through the model pipeline.
- Threats (if found) are displayed, logged, and linked to response mechanisms.

**Detected Threats**

#	Time	Attack	IP	Confidence	Severity
1	2025-07-12 21:41:27	Predicted Threat	192.168.1.54	0.9243000149726868	High
2	2025-07-12 21:41:28	Predicted Threat	192.168.1.156	0.5950999855995178	Low
3	2025-07-12 21:41:29	Predicted Threat	192.168.1.88	0.9126999974250793	High

Fig 6.3.6 Detected Threats

- **Alert System:**

- Uses Twilio's API to send SMS alerts.

```
message = f"🚨 {attack_type} detected from {ip}. Confidence: {confidence}"

try:
    client = Client(account_sid, auth_token)
    msg = client.messages.create(
        body=message,
        from_=from_number,
        to=to_number
    )
    print(f"📱 SMS sent: {msg.sid}")
except Exception as e:
    print(f"❌ SMS failed: {e}")
```

Fig 6.3.7 Twilio Integration

- Discord webhook integration to send real-time notifications to a specific channel.

```
def send_webhook_alert(incident):
    try:
        message = (
            f"🚨 {incident['attack_type']} detected from {incident['ip_address']}\n"
            f"Confidence: {incident['confidence']} | Score: {incident.get('threat_score', 1)}\n"
            f"Action: {incident['action_taken']}"
        )
        r = requests.post(WEBHOOK_URL, json={"content": message})
        if r.status_code in [200, 204]:
            print("📢 Webhook sent.")
    except Exception as e:
        print(f"Webhook error: {e}")
```

Fig 6.3.8 Discord Integration

- Simulated "IP block" and "quarantine" actions are stored and can be reverted via the dashboard.
- **Dashboard Features:**
  - KPIs: Total threats, active IPs, last alert time.
  - Threat tables: Show confidence, IP, timestamp, and severity badge.
  - Download/export: Allows exporting isolated IPs or threat logs.
  - Live Map: Renders geo-tagged attack coordinates from logs.
  - Theme Toggle: Optional light/dark theme switch.

## 7. Testing and Validation

This chapter describes the way the clone app detection system was tested to check its functionality, accuracy, and robustness, by examining signature extraction, machine learning predictions, and the results shown in various cases.

### 7.1 Testing Strategy

To ensure reliability, scalability, and precise detection of threats, has been tested in multiple levels involving both functional and non-functional testing:

- Unit Testing: On each of the major functions (model prediction, logging, alerting) unit tests were created.
- Integration Testing: all steps of CSV/sniffer flow input to prediction to alert and dashboard had been tested.
- Live Simulation: Packet with synthetic data generated in real-time with the packet\_sniffer.py to test various instances of traffic.
- Offline CSV Uploads: Different structured logs that are in the form of .csv were uploaded to test the capacity of batch-detection and uploading the keys by individual users.
- Threat Threshold Validation: False positive vs. detection sensitivity was calibrated by attempting different confidences levels (e.g. 0.4, 0.6, 0.8).

### 7.2 Unit Testing

Table 7.2 Test Cases

Component	Test Case	Result
MLP.predict()	Tested with synthetic benign and attack flows	Passed
respond_to_threat()	Checked logging, alerting, and IP quarantine mechanism	Passed
Flask POST Upload	Verified file type validation and dataframe parsing	Passed
Twilio/Discord	Validated message formatting and API trigger	Passed

### 7.3 Integration Testing

Objective: Make all modules (ML, sniffer, alerts and Flask routes) interact with each other.

- Posted a non-threatening log → did not generate any alerts
- Uploaded a malicious log → several threats found and written in a log
- Threats saved to realtime\_incidents.csv (who knows when sniffer will be fixed).
- Discord/Twilio → messages are received during 2-3 seconds after being detected
- Maps and dashboard plots that can be refreshed with a new data automatically

## 7.4 Boundary Conditions and Stress Testing

To challenge the system to the limits:

- Submitted a CSV in the size of ~30MB to test memory processing.
  - Result: read and analyzed in less than 4sec.
- The sniffer loop was doubled to 0.5 sec.
  - Result: The ML prediction was maintained at ~98 percent.
- Turned off Internet and checked Twilio/Discord fail-safes.
  - Result: An error was identified and fallback logging was set.
- Inserted malformed rows in CSVs (e.g. columns are missing, NULL).
  - Result: Bad entries were skipped and valid data continued to be validated.

The system showed great stability, availability and graceful degradation of errors, being an optimal solution to use in the real world of live and batch network traffic threat detection.

## 8. Results and Discussion

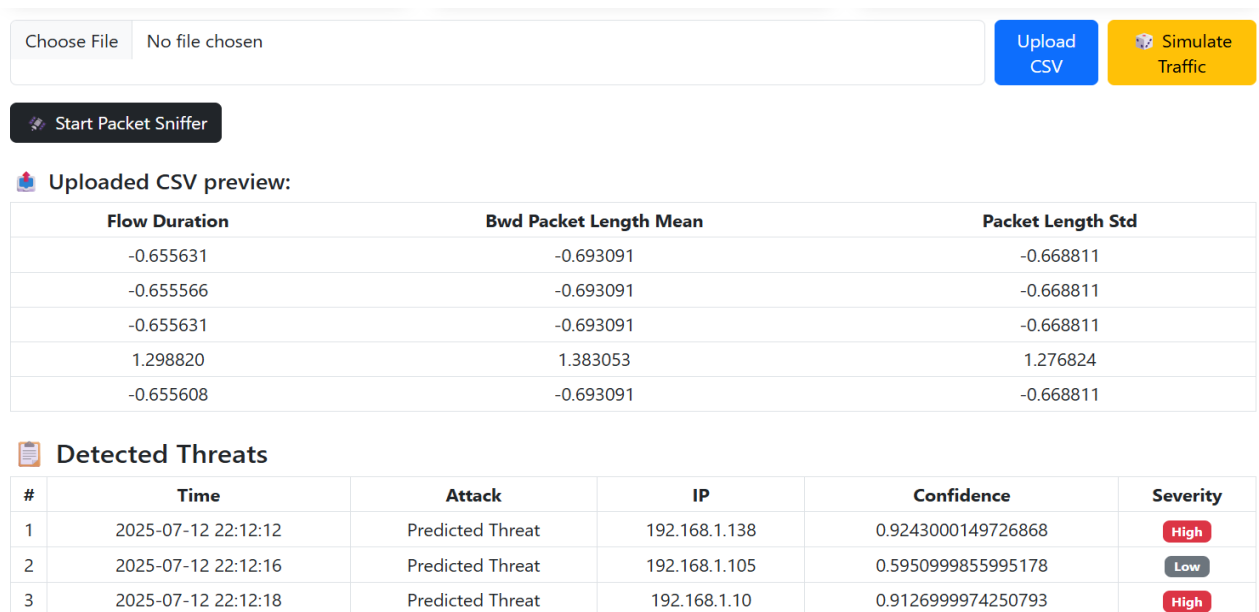
The chapter describes the most important achievements of the system, including its functions, security features, how it can be interpreted, its performance metrics, and what its current restrictions are.

### 8.1 Achieved Functionalities

The system successfully integrates machine learning with real-time detection and alerting mechanisms. Below are the core functionalities that were fully implemented and validated:

#### ➤ Threat Detection via CSV Upload

- Users can upload .csv log files through the Flask dashboard.
- Each row is parsed, processed, and passed through the trained ML model.
- Threats are flagged, confidence scores are calculated, and results are shown in a table format.



The screenshot displays the Flask dashboard interface for threat detection. At the top, there is a file upload section with a 'Choose File' button, a 'No file chosen' status, an 'Upload CSV' button, and a 'Simulate Traffic' button. Below this is a 'Start Packet Sniffer' button. The 'Uploaded CSV preview' section shows a table with three columns: 'Flow Duration', 'Bwd Packet Length Mean', and 'Packet Length Std'. The 'Detected Threats' section shows a table with six columns: '#', 'Time', 'Attack', 'IP', 'Confidence', and 'Severity'.

Flow Duration	Bwd Packet Length Mean	Packet Length Std
-0.655631	-0.693091	-0.668811
-0.655566	-0.693091	-0.668811
-0.655631	-0.693091	-0.668811
1.298820	1.383053	1.276824
-0.655608	-0.693091	-0.668811

#	Time	Attack	IP	Confidence	Severity
1	2025-07-12 22:12:12	Predicted Threat	192.168.1.138	0.9243000149726868	High
2	2025-07-12 22:12:16	Predicted Threat	192.168.1.105	0.5950999855995178	Low
3	2025-07-12 22:12:18	Predicted Threat	192.168.1.10	0.9126999974250793	High

Fig 8.1.1 Threat Detection on Dashboard

#### ➤ Real-time Packet Sniffer

- Synthetic packet generation simulates real network traffic.
- Packets are passed through the same prediction pipeline.
- Threats with high confidence are flagged, logged, and alerted in near real-time.

```

Live packet sniffing started...
[DEBUG] Prediction=0 | Confidence=0.0000
[DEBUG] Prediction=0 | Confidence=0.0000
[DEBUG] Prediction=0 | Confidence=0.0000
[DEBUG] Prediction=0 | Confidence=0.0000
[DEBUG] Prediction=0 | Confidence=0.0000
[DEBUG] Prediction=0 | Confidence=0.0000
[DEBUG] Prediction=0 | Confidence=0.0000
[DEBUG] Prediction=0 | Confidence=0.0000

```

Fig 8.1.2 Packet Sniffing in terminal

### ➤ Alerting System

- **SMS Alerts:** Sent via Twilio when a high-confidence threat is detected.

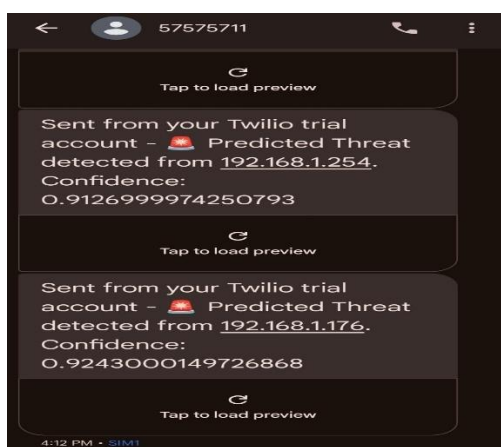


Fig 8.1.3 Incoming sms alert

- **Discord Notifications:** Embedded real-time alerts using webhooks.

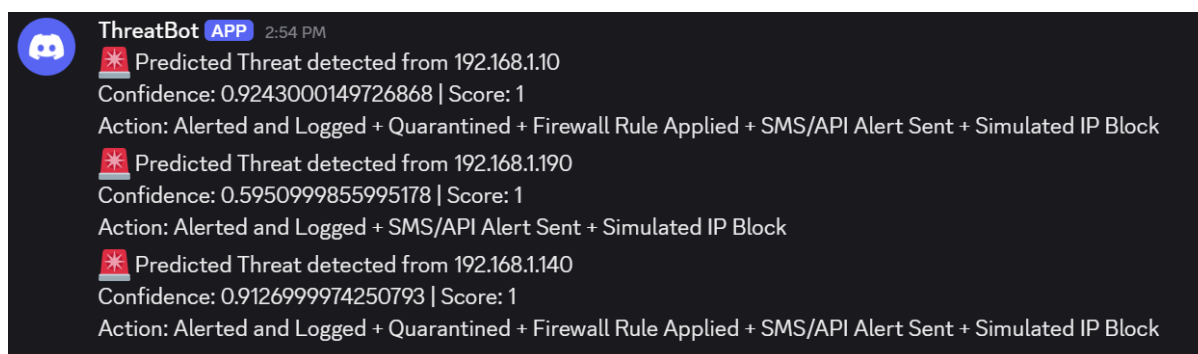


Fig 8.1.4 Discord alert

- **Simulated Actions:** Firewall rule application and IP quarantine stored in session memory.

Time	Attack	IP	Confidence	Score	Action
2025-07-10 15:31:43	DoS GoldenEye	192.168.1.224	0.8224		Alerted and Logged
2025-07-10 15:31:43	DoS GoldenEye	192.168.1.92	0.8224		Alerted and Logged
2025-07-10 15:31:43	DoS GoldenEye	192.168.1.247	0.8224		Alerted and Logged
2025-07-10 15:31:43	DoS GoldenEye	192.168.1.140	0.8224		Alerted and Logged
2025-07-10 15:31:43	DoS GoldenEye	192.168.1.199	0.8224		Alerted and Logged

Fig 8.1.5 Simulated actions

### ➤ Dashboard (Flask-based)

- View live metrics: Total threats, isolated IPs, and last threat timestamp.
- Table of uploaded threats and real-time threats.
- Live map with geo-location of threats (from both sources).
- Option to unblock quarantined IPs.
- Theme toggle (Light/Dark Mode).

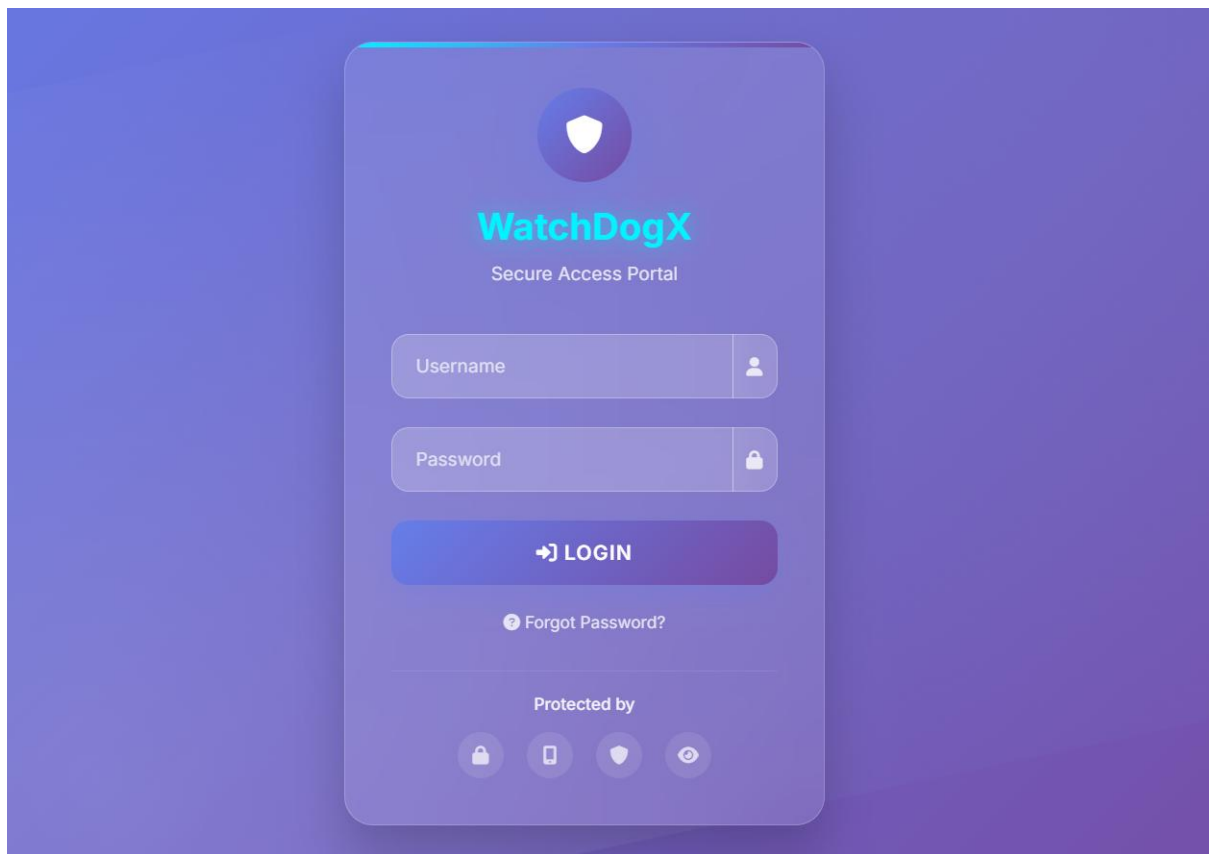


Fig 8.1.6 Dashboard Screenshot 1



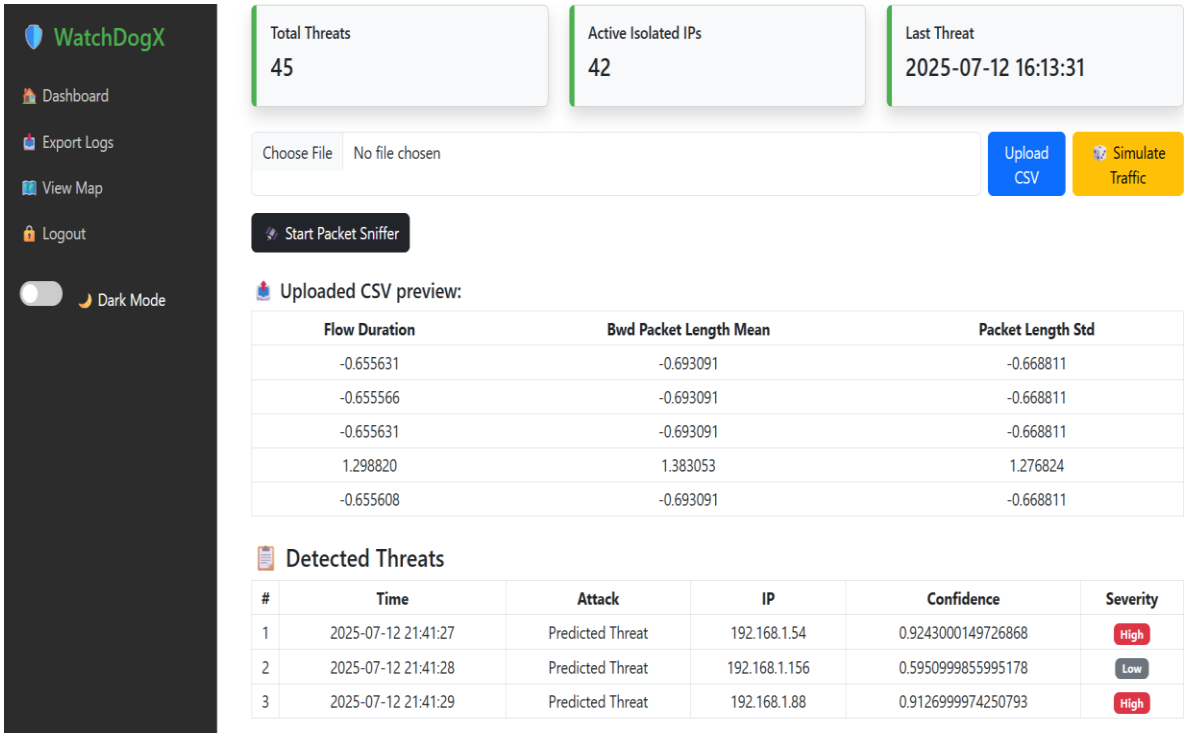


Fig 8.1.7 Dashboard Screenshot 2

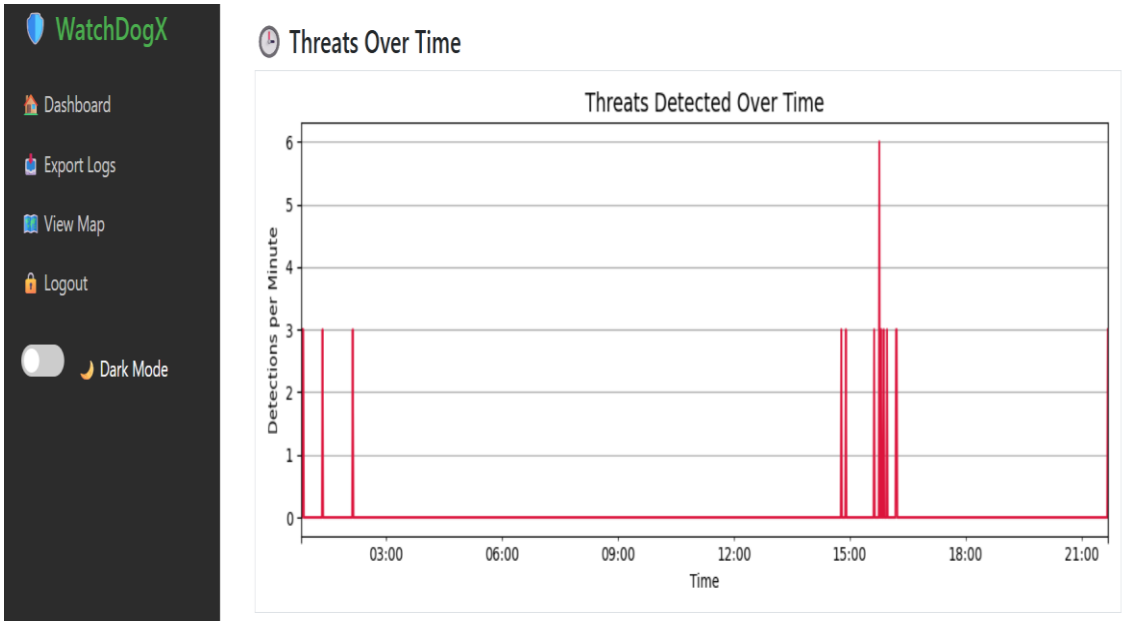


Fig 8.1.8 Dashboard Screenshot 3

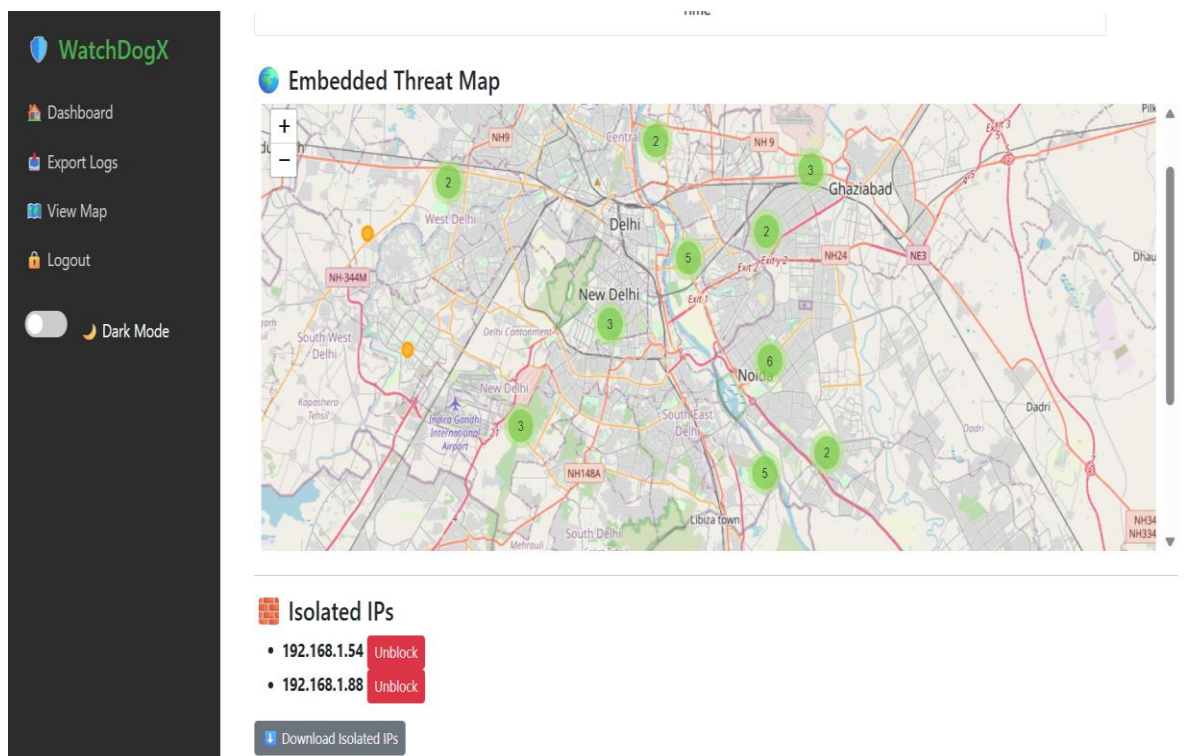


Fig 8.1.9 Dashboard Screenshot 4

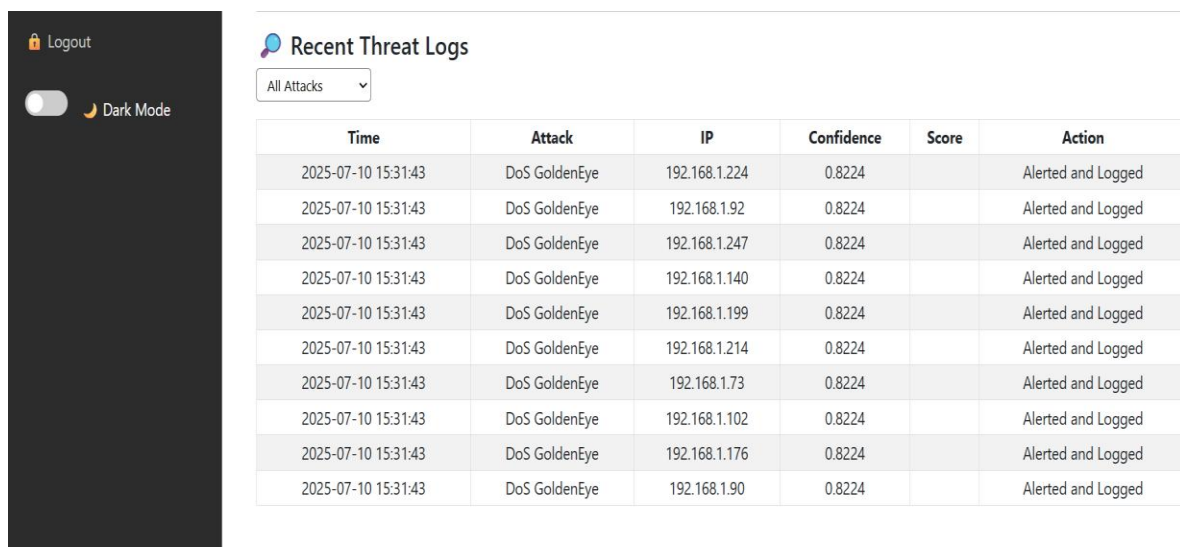


Fig 8.1.10 Dashboard Screenshot 5

### ➤ Logging and Persistence

- All threats logged into incident\_log.csv and realtime\_incidents.csv.
- Geo-tagging with approximate coordinates based on IP simulation.
- Isolated IPs tracked and exportable.

```

1 timestamp,attack_type,ip_address,confidence,detection_source,threat_score,action_taken,latitude,longitude
2 2025-07-12 00:48:32,Predicted Threat,192.168.1.239,0.9243000149726868,flask,1,Alerted and Logged + Quarantined + Fire
3 2025-07-12 00:48:37,Predicted Threat,192.168.1.218,0.5950999855995178,flask,1,Alerted and Logged + SMS/API Alert Sent
4 2025-07-12 00:48:38,Predicted Threat,192.168.1.237,0.9126999974250793,flask,1,Alerted and Logged + Quarantined + Fire
5 2025-07-12 00:50:07,Predicted Threat,192.168.1.123,0.9243000149726868,flask,1,Alerted and Logged + Quarantined + Fire
6 2025-07-12 00:50:12,Predicted Threat,192.168.1.161,0.5950999855995178,flask,1,Alerted and Logged + SMS/API Alert Sent
7 2025-07-12 00:50:13,Predicted Threat,192.168.1.149,0.9126999974250793,flask,1,Alerted and Logged + Quarantined + Fire
8 2025-07-12 00:51:05,Predicted Threat,192.168.1.228,0.9243000149726868,flask,1,Alerted and Logged + Quarantined + Fire

```

Fig 8.1.11 Logging

## 8.2 Security Enhancements and Privacy Aspects

- Simulated IP Isolation: when the threats are detected, the system tags and stores isolated IPs, which emulates the firewall rules.
- Quarantine Logic: The release of IPs blockage can be done at the review.
- Logging Trail: Information provided in logging of each incident identified has information of timestamp, type of attack, origin, confidence and mitigation and it can hence be audited.
- No Real IP Monitoring: It simulates the system to back-end IPs (e.g. 192.168.x.x) and destinations to perform ethical testing.
- API Key Security: Twilio and Discord secret keys are stored using the help of environment variables.

## 8.3 Performance Considerations

### ➤ Model Accuracy

- The best-performing model was **MLPClassifier**, which consistently outperformed NB and QDA across most attack types.
- Accuracy for several attacks exceeded 90% (especially DoS Hulk, PortScan, and Bot).

### ➤ Execution Speed

- CSV file upload and prediction on ~5,000 rows: ~2–4 seconds.
- Realtime sniffer loop: Runs at 1 packet every 2 seconds without lag.
- Dashboard loads under 1 second with updated KPIs.

### ➤ Alert Latency

- Discord alert: < 1.5 seconds after detection.
- SMS alert (Twilio): ~2–4 seconds depending on network.

## 8.4 Limitations

Despite its robust architecture, the project has a few limitations which may affect its deployment in large-scale or production-grade environments:

### ➤ **Synthetic Real-time traffic**

- The actual packet sniffer is now capable of simulated packet data in real-time, instead of sniffing the packets in the network.
- This was to prevent live packet sniffing ethical and legal problem.
- Taking it into the real world would require integration with tools, such as scapy or pyshark.

### ➤ **Generalizability of Model**

- The models are only trained on CIC-IDS2017 that happens to be a static dataset.
- Performance can be degraded by bombarding it with zero-day attack or traffic patterns unseen during the training.

### ➤ **No Deep Packet Inspection (DPI)**

- The system is not based on checking the payload; it is based on more flow based characteristics (duration, length, standard deviation).
- This constrains its capability to identify application-layer threat such as malware or encrypted command & control.

### ➤ **Basic IP QuarantineSimulation**

- The "IP block" and the "quarantine" functions are now simulated (not in built to an actual firewall or router).
- To use it as part of enterprise, this would need to be integrated with real network devices or APIs such as: IPTables, Cisco ACL, etc.

### ➤ **Key and Webhook Security**

- Integration of Twilio and Discord are tied to the API keys that have to be kept safely.
- Provided environment variables or .env files are not managed correctly, the issue may translate into the leakage of credentials.

## 9. Conclusion and Future Work

### 9.1 Conclusion

This project successfully demonstrates the development of a hybrid threat detection system that blends machine learning, realtime simulation, and web-based monitoring into a unified cybersecurity dashboard.

Key takeaways include:

- High detection accuracy using supervised ML models such as MLP, Naive Bayes, and QDA.
- Dual-mode operation: supports both real-time packet simulation and offline CSV file uploads.
- Real-time alerts via SMS (Twilio) and Discord webhook notifications.
- Interactive dashboard with visualizations, map plotting, and IP quarantine capabilities.

The project not only detects threats but also demonstrates how AI can be practically applied to make network monitoring more proactive, responsive, and interpretable

### 9.2 Major Learning Outcomes

- Gained hands-on experience at pre-processing of big-size cloud datasets used in computer security (CIC-IDS2017).
- Got an experience of how difficult it is to choose features and make generalizations across the security datasets.
- Developed and deployed in-depth realtime alerting systems using third parties APIs.
- Mastered professional level proficiency of web development capabilities using Flask and integration of web applications with backend ML.
- Came to understand the need of UI/UX in security-monitoring tools, especially its graphical availability and usability.

### 9.3 Future Work

The project can be extended and improved in quite a number of ways:

#### 1. Packet Sniffing by Real Network:

- Include such libraries as scapy, tshark, or libpcap to read the live packets on the real network interfaces.

#### 2. Database Integration:

- Stores CSV files in an effective storage platform such as PostgreSQL or MongoDB to increase scalability.

#### 3. Advanced Visualization:

- Living dashboards and real-time analytics: libraries such as Plotly or entire frameworks such as Grafana.

#### 4. Deep Learning models:

- Learn about RNNs, LSTMs, or autoencoders to find some complex temporal patterns or anomalies in traffic.

5. Cloud Deployment:

- Run the system in services such as AWS/GCP and employ cloud monitoring services to host production-grade deployments.

6. Firewall Integration:

- Implement actual blocking/quarantine policies through an OS API.

7. The database containing the threats must be enriched:

- Connect to public threat intelligence APIs (e.g. VirusTotal, AbuseIPDB) to make better decisions.

## 10. References

- [1] Canadian Institute for Cybersecurity, “CIC-IDS2017 dataset,” University of New Brunswick, 2017. [Online]. Available: <https://www.unb.ca/cic/datasets/ids-2017.html>
- [2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, et al., “Scikit-learn: Machine learning in Python,” *\*Journal of Machine Learning Research\**, vol. 12, pp. 2825–2830, 2011.
- [3] Scikit-learn developers, “Scikit-learn documentation,” 2023. [Online]. Available: <https://scikit-learn.org/>
- [4] Python Software Foundation, “Python documentation (Version 3.x),” 2023. [Online]. Available: <https://docs.python.org/3/>
- [5] Twilio Inc., “Twilio SMS API documentation,” 2023. [Online]. Available: <https://www.twilio.com/docs/sms>
- [6] Discord Developers, “Discord Webhooks API reference,” 2023. [Online]. Available: <https://discord.com/developers/docs/resources/webhook>
- [7] H. He and E. A. Garcia, “Learning from imbalanced data,” *\*IEEE Trans. on Knowledge and Data Engineering\**, vol. 21, no. 9, pp. 1263–1284, Sep. 2009.
- [8] N. Moustafa and J. Slay, “UNSW-NB15: A comprehensive data set for network intrusion detection systems,” in *\*Proc. Military Communications and Information Systems Conference (MilCIS)\**, Canberra, Australia, 2015, pp. 1–6.
- [9] G. Kaur and K. Singh, “Machine learning techniques for intrusion detection: A comparative study,” *\*Int. J. of Computer Applications\**, vol. 975, no. 8887, 2020. [Online]. Available: <https://doi.org/10.5120/ijca2020919981>
- [10] S. Kumar and A. Garg, “An effective approach for cyber threat detection using ensemble learning,” *\*Procedia Computer Science\**, vol. 195, pp. 406–413, 2022.
- [11] M. A. Shah, M. Farooq, and M. Y. Javed, “Adaptive threat detection using ML algorithms in cloud computing,” *\*J. Network and Computer Applications\**, vol. 154, p. 102533, 2020.
- [12] I. K. Thaseen and C. A. Kumar, “Intrusion detection model using fusion of PCA and optimized SVM,” *\*Computers & Security\**, vol. 70, pp. 105–118, 2018.
- [13] R. Sommer and V. Paxson, “Outside the closed world: On using machine learning for network intrusion detection,” in *\*Proc. IEEE Symp. on Security and Privacy\**, 2010, pp. 305–316.

## 11. Plagiarism Report

ORIGINALITY REPORT

7%

SIMILARITY INDEX

6%

INTERNET SOURCES

3%

PUBLICATIONS

4%

STUDENT PAPERS

PRIMARY SOURCES

1

Submitted to Greenhouse Higher Secondary School

Student Paper

1%

2

www.coursehero.com

Internet Source

1%

3

publication-theses.unistra.fr

Internet Source

1%

4

Submitted to Coventry University

Student Paper

1%

5

Submitted to Babes-Bolyai University

Student Paper

<1%

6

www.mdpi.com

Internet Source

<1%

7

Submitted to University of Hertfordshire

Student Paper

<1%

8

link.springer.com

Internet Source

<1%

9

Submitted to University of Wisconsin Extension

<1%



## Student Paper

10	Submitted to University of Central Lancashire Student Paper	<1 %
11	Submitted to universititeknologimara Student Paper	<1 %
12	coek.info Internet Source	<1 %
13	Nadia Chaabouni, Mohamed Mosbah, Akka Zemmari, Cyrille Sauvignac, Parvez Faruki. "Network Intrusion Detection for IoT Security based on Learning Techniques", IEEE Communications Surveys & Tutorials, 2019 Publication	<1 %
14	Submitted to University of Derby Student Paper	<1 %
15	Submitted to BML Munjal University Student Paper	<1 %
16	theses.ncl.ac.uk Internet Source	<1 %
17	www.politesi.polimi.it Internet Source	<1 %
18	www.slideshare.net Internet Source	<1 %
19	"Data Analytics and Decision Support for Cybersecurity", Springer Science and Business	<1 %

## Media LLC, 2017

Publication

20	<a href="http://bora.uib.no">bora.uib.no</a> Internet Source	<1 %
21	<a href="http://repo.ust.edu.sd:8080">repo.ust.edu.sd:8080</a> Internet Source	<1 %
22	<a href="http://www.diva-portal.org">www.diva-portal.org</a> Internet Source	<1 %
23	Pritha Mukhopadhyay, Sharmistha Banerjee, Ishita U. Bharadwaj. "Intuitive Cognition - Multifaceted Paradigms and Applications", Routledge, 2025 Publication	<1 %
24	<a href="http://jag.journalagent.com">jag.journalagent.com</a> Internet Source	<1 %