

**Princess Sumaya University for Technology**

**King Hussein School for Computing Sciences**

**Network Traffic Analysis and Anomaly Detection Using Machine Learning**

**Prepared By:**

**Fatima Aljabali  
Rasha Shabaro**

**Sereen Alrahahleh**

**Ayat Nahar**

**Supervised By:**

**Dr. Mu’away Al-Dala’ien**

**Project Submitted in Partial Fulfillment of the Requirements for the Degree of Science in Cybersecurity**



**Submitted: May 2024**

**Declaration of Originality**

This document has been written entirely by the undersigned team members of the project. The source of every quoted text is clearly cited and there is no ambiguity in where the quoted text begins and ends. The source of any illustration, image or table that is not the work of the team members is also clearly cited. We are aware that using non-original text or material or paraphrasing or modifying it without proper citation is a violation of the university’s regulations and is subject to legal actions.

Names and Signatures of team members:

Fatima Aljabali

Rasha Shabaro

Sereen Alrahahleh

Ayat Nahar

**Acknowledgements**

We express our deep gratitude for the establishment of this team, as the fellowship and support we experienced during our time together will always be cherished. We extend our sincere appreciation for the hard work we dedicated to this project.

We extend our deepest gratitude to Dr. Mu'awya Al-Dala'ien for his unwavering support and invaluable guidance throughout the course of this research. His exceptional expertise, insightful feedback, and constant encouragement have been instrumental in the success of the Twin Guard project. Dr. Al-Dala'ien profound understanding of the subject matter, combined with his visionary approach, has enriched our research journey and fostered an environment of intellectual curiosity.

We would also like to express our heartfelt appreciation to Princess Sumaya University for Technology (PSUT) for providing the necessary resources and platform that enabled us to undertake this ground breaking study. The university's commitment to fostering innovation and research excellence has been instrumental in our pursuit of scientific knowledge and technological advancements. We are truly grateful for PSUT's continued support and dedication to empowering its students to explore new frontiers.

Lastly, we would like to thank our families and loved ones for their unwavering support and understanding throughout this demanding research endeavour. Their patience, encouragement, and belief in our abilities have been a constant source of strength and motivation.

**F.Aljabali**

**R.Shabaro**

**S.Alrahahleh**

**A.Nahar**

**Table of Contents**

[Declaration of Originality](#_gjdgxs)

[Acknowledgements](#_30j0zll)

[Table of Contents](#_1fob9te)

[List of Figures](#_3znysh7)

[List of Tables](#_2et92p0)

[Abbreviations](#_tyjcwt)

[Abstract](#_3dy6vkm)

[Chapter 1 Introduction](#_1t3h5sf)

[1.1 Overview](#_4d34og8)

[1.2 Problem Statement](#_2s8eyo1)

[1.3 Significance of the study](#_17dp8vu)

[1.4 Study objectives](#_3rdcrjn) 2

[1.5 Study Contribution](#_26in1rg) 2

[1.6 Outline of the report](#_lnxbz9) 3

[Chapter 2 Project Plan](#_35nkun2) 4

[2.1 Project Deliverables](#_1ksv4uv) 4

[2.2 Project Tasks](#_44sinio) 4

[2.3 Roles and Responsibilities](#_2jxsxqh) 6

[2.4 Risk Assessment](#_z337ya) 7

[2.5 Cost Estimation](#_3j2qqm3) 8

[2.6 Project Management Tools](#_1y810tw) 8

[Chapter 3 Literature Review and Related Work](#_4i7ojhp) 9

[3.1 Related Work](#_2xcytpi) 9

[3.2 Knowledge Gap](#_1ci93xb) 12

[Chapter 4 Requirements Specification](#_3whwml4) 13

[4.1 Stakeholders](#_2bn6wsx) 13

[4.2 Platform Requirements](#_qsh70q) 13

[4.3 Functional Requirements](#_3as4poj) 14

[4.4 Non-Functional Requirements](#_1pxezwc) 14

[4.5 Other Requirements](#_49x2ik5) 15

[Chapter 5 System Design](#_2p2csry) 16

[5.1 Architectural Design](#_147n2zr) 16

[5.2 Logical Model Design](#_3o7alnk) 17

[Chapter 6 Implementation](#_23ckvvd) 23

[6.1 General Implementation Description](#_ihv636) 23

[**6.2** Pipeline Implementation Description](#_32hioqz) 39

[6.3 Model Implementation](#_1hmsyys) 63

[6.4 Additional Implementation Details](#_41mghml) 63

[Chapter 7 Testing](#_2grqrue) 67

[7.1 Testing Approach](#_vx1227) 67

[7.2 Testing Results](#_3fwokq0) 69

[**7.3** Discussion](#_1v1yuxt) 70

[Chapter 8 Conclusions and Future Work](#_4f1mdlm) 71

[8.1 Conclusions](#_2u6wntf) 71

[8.2 Future work](#_3tbugp1) 71

[References](#_28h4qwu) 72

[Appendices 4](#_nmf14n)

[Appendix A](#_37m2jsg) 4

[A.1 Users’ Manual 4](#_1mrcu09)

A.2 Users’ Manual ……………….…………………………………... 77

[Appendix B](#_46r0co2) 8

[B.1 Document Changes](#_2lwamvv) 8

[Appendix C](#_111kx3o) 9

[C.1 Code Documentation](#_3l18frh) 9

**List of Figures**

[Figure 3.1: Isolation forest algorithm 10](#_206ipza)

[Figure 5.1: Architecture of system. 1](#_206ipza)6

[Figure ‎5.2: Use Case Diagram. 1](#_206ipza)7

[Figure ‎5.3: Package and Class Diagrams. 1](#_206ipza)8

[Figure ‎5.4: Deployment Diagram. 1](#_206ipza)8

[Figure ‎5.5: Activity Diagram. 1](#_206ipza)9

[Figure ‎5.6: Sequence Diagram. 1](#_206ipza)9

[Figure ‎5.7: State Transition Diagram. 1](#_206ipza)9

[Figure 5.8: Amomalyzer. 20](#_206ipza)

[Figure 5.9: Upload the dataset. 20](#_206ipza)

[Figure 5.10: Zoom in for the uploading screen 2](#_206ipza)1

[Figure ‎5.11: Anomalyzer Alert. 2](#_206ipza)1

[Figure ‎5.12: Anomalyzer Alert Meaning 2](#_206ipza)2

[Figure 7.2.1: Testing Set Report. 6](#_206ipza)9

[Figure7.2.2: Capturing Output from Real Traffic. 70](#_206ipza)

[Figure Appendix A.1: Start Page. 7](#_206ipza)4

[Figure Appendix A.2: Uploading Page. 7](#_206ipza)4

[Figure Appendix A.3: Upload the Dataset. 7](#_206ipza)5

[Figure Appendix A.4: Capture Real Traffic. 7](#_206ipza)5

[Figure Appendix A.5: Result Page (Real Traffic Capturing) 7](#_206ipza)6

[Figure Appendix A.6: Alert Pop-Up Window. 7](#_206ipza)6

Figure Appendix A.7: Alert Meaning ………………………………………..……………...……..77

**List of Tables**

Table 1: The Audience Our Project Serves………………………………………………2

Table 2: Outline of the Report……………………………………………………………3

Table 3: Project Deliverables………………………………………………………….…4

Table 4: Analysis Phase in Project Tasks ………………………………………...………5

Table 5: Design Phase in Project Tasks…………………………………………….……5

Table 6: Implementation Phase in Project Tasks…………………………………...……6

Table 7: Roles and Responsibilities……………...………………………………………7

Table 8: Risk Assessment……………………………………………..…….……………7

Table 9: Cost Estimation……………………………………………..…..………………8

Table 10: Project Management Tools……………………………………………………8

Table 11: Stakeholders…………………………………………………….……………13

Table 12: Functional Requirements…………………………….………………………14

Table 13: Non-Functional Requirements ………………………………………………15

Table 14: Feature Implementation Table………………………………………………66

**Abbreviations**

**IDS** Intrusion Detection System

**ML** Machine Learning

**PCAP** A Common Format for Storing Packet Captures

**CSV** Comma-separated Values file

**UI/UX** User Interface & User Experience

**SVM** Support Vector Machine

**OCSVM** One Class Support Vector Machine

**DBSCAN** Density-Based Clustering Non-Parametric Algorithm

**ML** Machine Learning

**IoT** Internet of Things

**IPS**                  Intrusion prevention system

**PCA**  Principal Component Analysis

**GUI** Graphical User Interface

**RFE**  Recursive Feature Elimination

**CNN**  Convolutional Neural Networks

**Network Traffic Analysis and Anomaly Detection Tool Using Machine Learning**

**By**

Fatima Aljabali & Rasha Shabaro &

Sereen Alrahahleh & Ayat Nahar

Supervisor

Dr. Mu'awya Al-Dala'ien

**Abstract**

This study explores the integration of machine learning techniques into intrusion detection systems (IDS) to improve their effectiveness and accuracy in identifying security threats. Traditional IDS rely on predefined rules and signatures to detect malicious activities, which can be insufficient in the face of evolving cyber threats.

Our research aims to enhance IDS by implementing advanced machine learning models that can analyse network traffic data, learn from historical patterns, and adapt to new threats in real-time.

Through this approach, we aim to reduce false positives and improve the detection of genuine security breaches. The results demonstrate that machine learning-enhanced IDS offer significant improvements in identifying sophisticated attacks compared to conventional methods. These findings suggest that adopting machine learning can greatly enhance the resilience and adaptability of network security systems.

# Chapter 1 Introduction

## Overview

The effectiveness and accuracy of intrusion detection tools in identifying security threats can be greatly improved by integrating machine learning techniques into them, since most intrusion detection tools work by analysing network traffic using already existing rules and signatures. Through the use of more sophisticated and adaptable techniques for spotting malicious activity, our initiative aims to improve IDS.

Our goal is to develop a system that can learn from past data, adjust to new and emerging dangers, and make wise decisions instantly by putting machine learning principles into practice. Large volumes of network traffic data can be analysed using machine learning models, which can then be used to spot trends and abnormalities that can point to a security risk. As these models are exposed to different kinds of data and attack vectors over time, they may also get better.

## Problem Statement

Traditional intrusion detection systems primarily rely on analysing network traffic using pre-established rules and signatures to identify potential security breaches. Although these techniques are useful, they frequently fail to keep up with the quickly changing cyber threat scenario. Static rule-based systems lose effectiveness over time since attackers continually create new ways to avoid detection.

Machine learning-enhanced intrusion detection systems (IDS) not only increase detection accuracy but also decrease false positives, which are a significant problem in conventional systems. Security teams may get overloaded with false positives, which could result in alert fatigue and possibly lead to the overlooking of genuine threats. Security analysts may concentrate their efforts on real threats because the IDS can more precisely differentiate between benign and malicious activity by utilising machine learning, and our tool aims to do just that.

## Significance of the project

Our project addresses critical gaps in contemporary cybersecurity solutions. By leveraging machine learning, it offers:

1. Proactive identification of anomalies with enhanced accuracy and reduced false positives.
2. Continuous learning and model updates ensure responsiveness to new and emerging threats.
3. Stronger overall security posture through proactive threat mitigation and efficient monitoring.
4. Minimize false alarms by applying robust anomaly detection techniques that are more sensitive to genuine deviations from normal behaviour.

## Project objectives

1. The primary goal is to detect unusual behaviours in the data that differ from the behaviour expected.
2. To develop an anomaly detection tool with high accuracy and low false positives. Reducing false alarms is very important to prevent unnecessary alerts.
3. To create an IDS by integrating machine learning techniques and strengthening network monitoring capabilities.
4. Processing large amounts of data using proper algorithms.

## Project Contribution

The contributions of our project and our tool:

1. Novelty in the idea

Our tool “Anomalyzer” is a stand-alone IDS application that relies purely on machine learning to detect anomalous behaviours in the network traffic at a certain point in time. Our tool offers a user-friendly interface to either capture the network traffic using T-shark on the host device, or for the user to upload an already captured PCAP file and analyse it using the ML model. This tool also aims to increase accuracy and reduce the false positive rates that tend to be high in a regular IDS system.

1. The audience that it serves and how

| **Audience** | **How it serves them** |
| --- | --- |
| Network Admins | Monitor the network and detect anomalies |
| System Administrators | Reduce threats by detecting anomalies and finding system vulnerabilities. |
| Security Analysts | Help in the investigation of incidents and improving the overall security of the network and identifying attack patterns |
| Data Analysts and Researchers | Identify anomalies and extract abnormal components from the data by analysing massive amounts of data. |

**Table 1: The Audience Our Project Serves**

1. Novelty in the choice of the model

Our project focuses on the implementation of Isolation Forest, an unsupervised ML algorithm, for anomaly detection over the network traffic, since Isolation Forest is one of the best algorithms that detects and separates anomalous points in the traffic, and it seemed to show the best results compared to other unsupervised algorithms like SVM and K-means.

1. Novelty in the structure of the pipeline

Our tool addresses critical challenges in the field of Intrusion Detection with the algorithm that we chose being tuned to reduce false positive alerts as much as possible. It is also built to tolerate a large number of network traffic data and make sure that the anomalies are detected in a timely manner.

## Outline of the report

| **Chapter Title** | **Description** |
| --- | --- |
| **Graduation Project 1** | |
| **Chapter 1:**  **Introduction** | Provides a detailed overview of the project idea, the problem statement, and significance of the study, study objectives, study contribution and the targeted audience for this project**.** |
| **Chapter 2:**  **Project plan** | A detailed description of the project's flow, deliverables, and project management tools required, and the tasks distributed amongst team members. It also contains the risk assessment for each task and the project's cost estimation. |
| **Chapter 3:**  **Literature Review and Related Work** | Shows the related work and the knowledge gap for this project. |
| **Chapter4:**  **Requirements Specification** | Presents the functional and non-functional requirements and required software specifications with the declaration of stakeholders. |
| **Chapter 5:**  **System Design** | Shows the logical model design, physical model design and architectural design. |
| **Graduation Project 2** | |
| **Chapter 6:**  **Implementation** | Provides a general, pipeline implementation description, model implementation, and additional implementation details. |
| **Chapter 7:**  **Testing** | Shows the testing approach and the testing results with the discussion. |
| **Chapter 8:**  **Conclusion and Future work** | Summarises any results achieved in this project and discusses our next step in the future. |

**Table 2: Outline of the Report**

# Chapter 2

# Project Plan

## 2.1 Project Deliverables

The table below provides an overview of the deliverables associated with our project. Each deliverable has a brief description, to understand the purpose and scope of each deliverable:

| **Deliverable** | **Description** |
| --- | --- |
| **Dataset** | Data required for testing and operating the machine learning model. |
| **Diagrams** | Visual representations of the work such as flowcharts and use case diagrams. |
| **Progress Report** | Report describing the progress of the project and a timeline that shows the amount of work done for each week. |
| **Research Report** | A documentation report summarizing the research. |
| **User Interface** | An eye caching user interface dashboard design that displays the output of the system. |

**Table 3: Project Deliverables**

## 2.2 Project Tasks

Our project is divided into three high-level tasks:

1. **Analysis Phase:**

This covers the requirements identification, development strategy, requirement modelling and the issues that we will solve by implementing the project.

| **Task Number** | **Task Name** | **Description** | **Duration** | **Dependencies** | **Progress** |
| --- | --- | --- | --- | --- | --- |
| **RA1** | Define project Scope | Choose the project’s idea. | 14 days | \_\_\_\_\_\_ | 100% |
| **RA2** | Feasibility analysis | Review/study of relevant systems and technologies. | 8 days | RA1 | 100% |
| **RA3** | Dataset | Finding a dataset for our project. | 3 days | RA2 | 100% |
| **RA4** | Project description | Providing a precise description of our idea. | 5 days | RA3 | 100% |
| **RA5** | Requirement Modelling and Algorithms | Determining the requirements that our system needs to meet in order to solve our issue. | 5 days | RA4 | 100% |
| **RA6** | Development strategy | Deciding the best approach to accomplish our goals. | 6 days | RA5 | 100% |

**Table 4: Analysis Phase in Project Tasks**

1. **Design Phase:**

Designing all the designs and diagrams that determine our development plan.

| **Task Number** | **Task Name** | **Description** | **Duration** | **Dependencies** | **Progress** |
| --- | --- | --- | --- | --- | --- |
| **D1** | Data preparation | Preparing the right data for the design | 20 days | RA6 | 100% |
| **D2** | System components | Designing system components | 18 days | RA5 | 100% |
| **D3** | System Architecture | Enhancement of the system architecture. | 10 days | RA5 | 100% |
| **D4** | System Design | Finalizing the system design. | 30 days | RA5 | 100% |

**Table 5: Design Phase in Project Tasks**

1. **Implementation phase:**

This covers the coding part, testing and documentation. We implement our idea into

a real-life application.

| **Task Number** | **Task Name** | **Description** | **Dependencies** |
| --- | --- | --- | --- |
| **I1** | Implementation | Developing the system | RA6 |
| **I2** | dashboard | Designing and implementing a dashboard for the system | I1 |
| **I3** | Testing | Test the system | I1 |
| **I4** | Documentation | Preparing the documentation for the whole project during the entire semester | ALL |

**Table 6: Implementation Phase in Project Tasks**

## 2.3 Roles and Responsibilities

The table below describes the roles and responsibilities of each member in the project:

| **Member** | **Role** | **Responsibilities** |
| --- | --- | --- |
| **Dr. Mu'awya Al-Dala'ien** | **Supervisor** | -Providing guidance regarding best practices and technical aspects for the team.  -Providing mentorship. |
| **Fatima** | **-Team leader**  -Researcher  -Developer  -Programmer  -Security  -Analyst | -Assignment of Tasks.  -Tracking of progress.  -Overall project leadership.  -Researching security issues.  -Developing the system code |
| **Sereen** | -Researcher  -Developer  -Programmer  -Security Analyst | -Researching efforts and threat analysis.  -Implementing code for the system.    -Suggesting security enhancements.  -Analysing security features. |
| **Ayat** | -Researcher  -Developer  -Programmer  -Security Analyst | -Researching security issues.  -Developing code for the implementation.  -Analysing and enhancing security features. |
| **Rasha** | -Researcher  -Developer  -Programmer  -Security Analyst | -Researching efforts and threat analysis.  -Coding for the implementation of the system  -Analysing security measures. |

**Table 7: Roles and Responsibilities**

## 2.4 Risk Assessment

This table shows the risks that we can face while doing the project and their impact.

| **Risk Number** | **Risk Description** | **Task Affected** | **Risk Probability** | **Risk Impact** |
| --- | --- | --- | --- | --- |
| **R1** | Not being able to find a proper dataset that will cover all necessary features. | RA3 | **Medium** | **High** |
| **R2** | Time constraints, where we wouldn’t be able to find the best optimal solution for the problem. | RA6 | **High** | **Medium** |
| **R3** | The possibility of some features not being fulfilled and for our project not to be fully complete. | D2 | **Medium** | **Medium** |
| **R4** | The timeline might fail us to deliver everything clearly. | I1 | **Low** | **High** |
| **R5** | Focusing on specific features by overworking them, and neglecting others. | I1 | **Low** | **Medium** |
| **R6** | External events that could unexpectedly pause the progress. | All | **Medium** | **High** |
| **R7** | Not being able to reduce the false positive and not increasing the accuracy of the anomaly detection system | All-I | **Medium** | **High** |

**Table 8: Risk Assessment**

## 2.5 Cost Estimation

This table shows the estimated costs of the tools that we are using in our project.

| **Tools** | **Services and** | **Estimated cost** |
| --- | --- | --- |
| **Figma** | UI/UX Design | $15 per month |
| **Google Colab** | For Collaborating on our code. | $49.99 per month |

**Table 9: Cost Estimation**

## 2.6 Project Management Tools

Those are the tools that we used for version control software and project management software.

| **Tool** | **Purpose** |
| --- | --- |
| **Google Documents** | Documenting and editing the project. |
| **Google Drive** | To store files online and access them anywhere using the cloud. |
| **Canadian Dataset** | Finding datasets. |
| **Tshark** | Network monitoring tool. |
| **Figma** | Dashboard and interface design (UI/UX Design). |
| **Qt Designer** | GUI development tool. |
| **Visual Studio** | Code Editor (Compiler) |
| **Drawio** | Tool for creating diagrams. |

**Table 10: Project Management Tools**

# Chapter 3

# Literature Review and Related Work

This chapter provides a comprehensive analysis of existing research on malware detection in network traffic. It encompasses various methodologies, data collection techniques, feature selection strategies, machine learning algorithms, accuracy measures, and underlying technologies. Drawing from a multitude of sources, the review highlights the diverse approaches employed in previous studies while identifying common challenges and constraints encountered. Emphasis is placed on understanding the best-performing methods and technologies, with a focus on their accuracy and performance in detecting anomalies in network traffic. Through this exploration, the chapter aims to provide insights for advancing the field of malware detection and shaping future research directions.

## 3.1 Related Work

### 3.1.1 Introduction

Anomaly detection is a critical component of cybersecurity, focusing on identifying unusual patterns that deviate from the norm and potentially indicate malicious activity. This section reviews existing literature on anomaly detection, particularly in the context of network intrusion detection systems (IDS) and the application of machine learning techniques. The review covers various methods, their effectiveness, and the challenges they face.

### 3.1.2 Anomaly Detection

Anomaly detection involves identifying deviations from expected behaviour in data. Traditional methods rely on statistical approaches, which are effective but often limited in handling complex, high-dimensional data. Recent advancements in machine learning have significantly improved the ability to detect anomalies by learning from vast amounts of data and identifying intricate patterns.

### 3.1.3 Machine Learning

Machine learning has revolutionized anomaly detection by enabling systems to learn from data and improve their detection capabilities over time. Techniques such as supervised, unsupervised, and semi-supervised learning are employed to enhance the detection of unknown threats.

* **Supervised Learning:**

Supervised learning involves training models on labelled datasets where anomalies are predefined. This approach requires a comprehensive dataset with examples of both normal and anomalous behaviour. Common algorithms include Support Vector Machines (**SVM**) and Random Forests.

* **Unsupervised Learning:**   
  Unsupervised learning does not require labelled data and identifies anomalies based on the inherent structure of the data. Techniques such as K-Means Clustering and Isolation Forests are commonly used. These methods are advantageous for detecting new and previously unseen threats.
* **Semi-Supervised Learning:**

Semi-supervised learning combines both labelled and unlabelled data to improve model accuracy. It is particularly useful when labelled data is scarce, leveraging the vast amounts of unlabelled data to enhance learning.

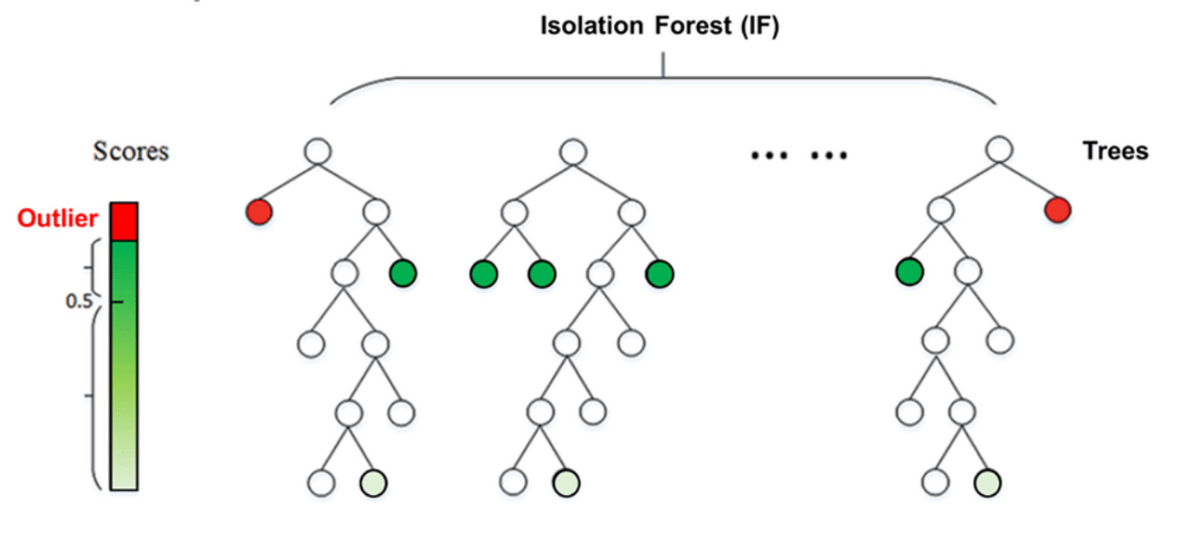
### 3.1.4 Data Collection

Data collection is a crucial step in anomaly detection, providing the foundation upon which models are built. Effective data collection involves capturing relevant features that can indicate anomalous behaviour. Network traffic data, system logs, and user behaviour data are commonly used sources.

### 3.1.5 Machine Learning Algorithms

Various machine learning algorithms have been employed for anomaly detection, each with its strengths and limitations.

* **Support Vector Machines (SVM):** Effective in high-dimensional spaces and when the number of dimensions exceeds the number of samples. SVMs are robust but can be computationally expensive for large datasets.
* **Random Forests:** An ensemble learning method that builds multiple decision trees and merges their predictions. Random Forests handle large datasets well and provide high accuracy, though they can be resource-intensive.
* **K-Means Clustering**: A simple and scalable unsupervised learning algorithm that groups data into clusters. It is sensitive to the initial choice of cluster centres and can struggle with varying cluster sizes and densities.
* **Isolation Forest:** Efficient for high-dimensional data, isolating observations by randomly selecting features and splitting values. It requires fewer splits to isolate anomalies, making it effective and fast.



**Figure 3.1: Isolation forest algorithm.**

### 3.1.6 Machine Learning Feature Selection

Feature selection is vital for improving model performance and reducing computational complexity. It involves selecting the most relevant features that contribute to identifying anomalies. Techniques such as Principal Component Analysis (PCA) and Recursive Feature Elimination (RFE) are commonly used.

### 3.1.7 Anomaly Detection Methods

Anomaly detection methods can be broadly categorized into statistical methods, machine learning techniques, and hybrid models:

* **Statistical Methods:** Include techniques such as Z-Score Analysis, PCA, and Gaussian Mixture Models (GMM). These methods analyse data distributions and identify deviations indicating anomalies.
* **Machine Learning Techniques:** Encompass supervised, unsupervised, and semi-supervised learning approaches, as detailed earlier.
* **Hybrid Models:** Combine statistical and machine learning methods to leverage their strengths. For instance, combining PCA with a machine learning classifier can enhance detection accuracy and reduce false positives.

### 3.1.8 Dataset Collection

Effective anomaly detection relies on comprehensive datasets that capture a wide range of normal and anomalous behaviours. Publicly available datasets such as the **Canadian Dataset** widely used for training and evaluating anomaly detection models. These datasets provide a benchmark for comparing different methods and ensuring reproducibility in research.

### 3.1.9 Feature Selection

Selecting the right features is crucial for building effective anomaly detection models. Features such as network flow statistics, packet-level data, and user behaviour metrics are commonly used. Feature selection techniques help in identifying the most relevant features, improving model accuracy, and reducing computational overhead .

### 3.1.10 Real-World Applications and Case Studies

Reviewing case studies and real-world applications of anomaly detection systems can provide practical insights and highlight the effectiveness of various approaches in different environments. Examples include the implementation of anomaly detection in critical infrastructure, financial systems, and cloud environments.

### 3.1.11 Evaluation Metrics and Benchmarks

Discussing various evaluation metrics used to assess the performance of anomaly detection models is crucial. Common metrics include precision, recall, F1-score.

### 3.1.12 Packet Analysis

Packet analysis involves capturing and examining network packets to identify anomalies. Tools like Wireshark and tcpdump are widely used for deep packet inspection. This method allows for detailed inspection of packet headers and payloads, making it possible to detect suspicious activities and patterns.

### 3.1.13 Intrusion Detection Systems (IDS)

IDS can be classified into two main types:

* **Signature-based IDS:** These systems detect known threats by comparing network traffic against a database of known attack signatures. Snort is a popular open-source signature-based IDS.
* **Anomaly-based IDS**: These systems detect deviations from established normal behavior. Examples include Bro (now known as Zeek) and Suricata, which can perform both signature-based and anomaly-based detection.

## 3.2 Knowledge Gap

Despite significant advancements, several gaps remain in the field of anomaly detection:

**High False Positive Rates**: Many anomaly detection systems suffer from high false positive rates, overwhelming security teams and reducing overall effectiveness. There is a need for more precise models that can accurately distinguish between benign anomalies and true threats.

**Scalability Issues:** Handling large volumes of data in real-time remains a challenge. Many existing solutions struggle with the scalability needed for real-time anomaly detection in large networks.

**Adaptability:** As cyber threats evolve, detection systems need to adapt quickly to new patterns of behaviour. Many current models lack the flexibility required to adapt to new and emerging threats.

Our project aims to address these gaps by optimizing the Isolation Forest algorithm to reduce false positives and enhance scalability. By implementing a more efficient data processing pipeline and leveraging high-performance computing resources, our system can handle large datasets and provide real-time anomaly detection.

# Chapter 4

# Requirements Specification

## 4.1 Stakeholders

The table below shows the stakeholders of our system and their responsibilities.

| **Stakeholders** | **Responsibilities** |
| --- | --- |
| **Dr. Mu'awya Al-Dala'ien** | Supervisor of the project. |
| **Programmers** | The developers of the system (Fatima,Ayat, Rasha and Sereen) |
| **Network and system administrator, Data Analysts and researchers** | Use the dashboard that is connected to the system. |

**Table 11: Stakeholders**

## 4.2 Platform Requirements

1. **Hardware requirements:**

Required RAM and CPU power.

Sufficient storage space for storing and processing the data.

1. **Software requirements:**

Operating System.

Dataset.

Anomaly Detection Algorithms.

Machine Learning Algorithms.

Monitoring Tools.

Interface Design & it’s Code.

Programming Language.

## 4.3 Functional Requirements

The following is a detailed table of all the functional requirements of the system. Functional Requirements.

| **No.** | **Requirement** | **Description** | **Importance** |
| --- | --- | --- | --- |
| FR1 | User’s ability to upload file/dataset | An input box that allows the user to input either a CSV or PCAP file they want to check | Essential |
| FR2 | File format conversion | If the input file was a PCAP format file, convert it to CSV format | Essential |
| FR3 | Anomaly detection using machine learning algorithm | The system should feed the input file into the  machine learning algorithm | Essential |
| FR4 | Output | The dashboard should output the percentage and the appropriate colour for the malicious traffic, and benign traffic | Essential |
| FR5 | Danger level | The system should output the danger level (the level of malware in the dataset) to the user, using different colours | Recommended |
| FR6 | Preparing the dataset | Apply the necessary preprocessing and feature selection to the dataset | Essential |
| FR7 | Training the Model | Train the Isolation Forest using the dataset and tune the hyperparameters | Essential |

**Table 12: Functional Requirements**

## 4.4 Non-Functional Requirements

The following is a table for each non-functional requirements in our system

| **No.** | **Non-Functional Requirements** | **Description** |
| --- | --- | --- |
| 1 | User friendly UI/UX | The dashboard provides a clear overview of the network status, and real time alerts. |
| 2 | Performance | The system should be fast responding and give high accuracy results. |
| 3 | Effectiveness | Ability of the system to perform the functions necessary to achieve the best accuracy. |

**Table 13: Non-Functional Requirements**

## 4.5 Other Requirements

1. **Testing and quality assurance:**

Preform regular code reviews and audits.

1. **Documentation:**

Clear documentation of the setup, configuration, and usage instructions for the system. This documentation is important for both end users and programmers.

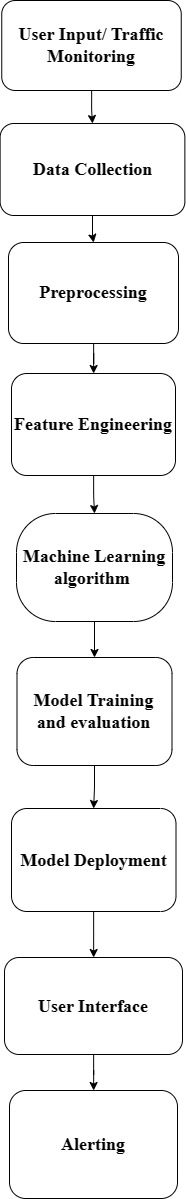
1. **Security Measures:**

We might consider including any necessary security measures, such as firewalls, encryption protocols, and authentication mechanisms to protect the system and data.

# Chapter 5

# System Design

## 5.1 Architectural Design

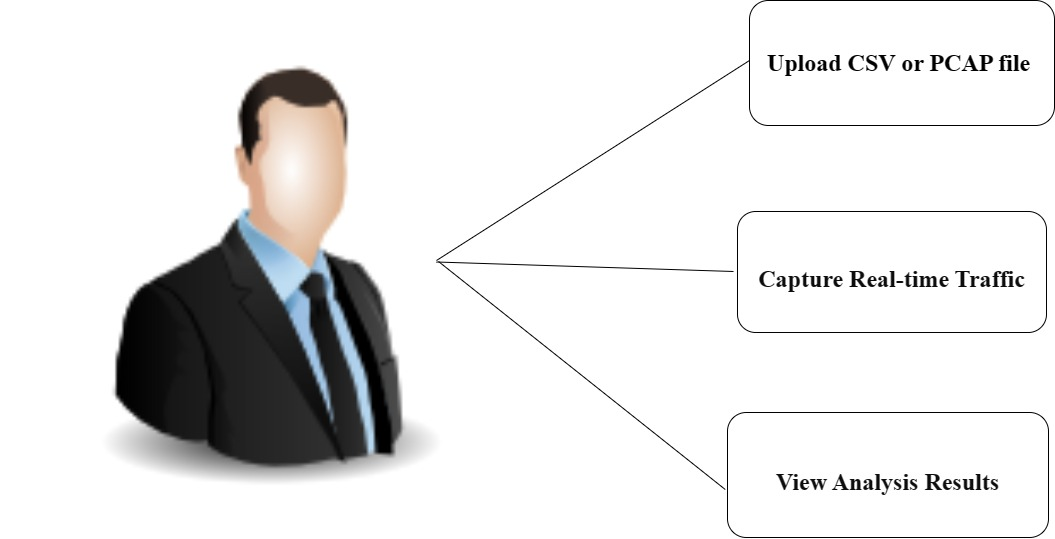


**Figure 5.1: Architecture of system.**

The architecture diagram outlines the Anomalyzer dashboard interface, enabling user interaction by uploading datasets or capturing real-time traffic. Data is pre-processed for analysis through cleaning, normalization, and feature extraction. Machine learning models are trained on unlabelled data to detect normal network behaviour and evaluate performance metrics. Deployed models enable real-time traffic analysis. The system continuously monitors traffic patterns, generating alerts upon anomaly detection for administrator action.

## 5.2 Logical Model Design

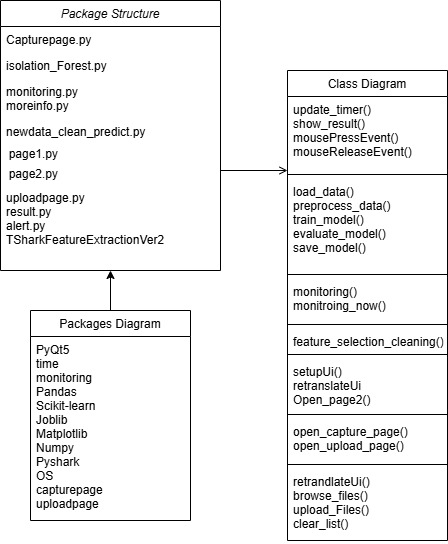
* **Use Case Diagrams:**



**Figure ‎5.2: Use Case Diagram.**

The above diagram describes the interactions between end users and the system. Here it shows the user can Upload Dataset, Capture Real-time Traffic and View Analysis Result.

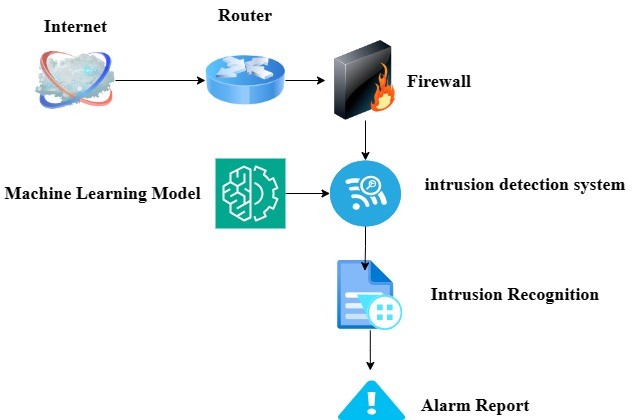
* **Package and Class Diagrams.**



**Figure ‎5.3: Package and Class Diagrams.**

This diagram describes the packages and the classes that we are going to use it in the implementation phase

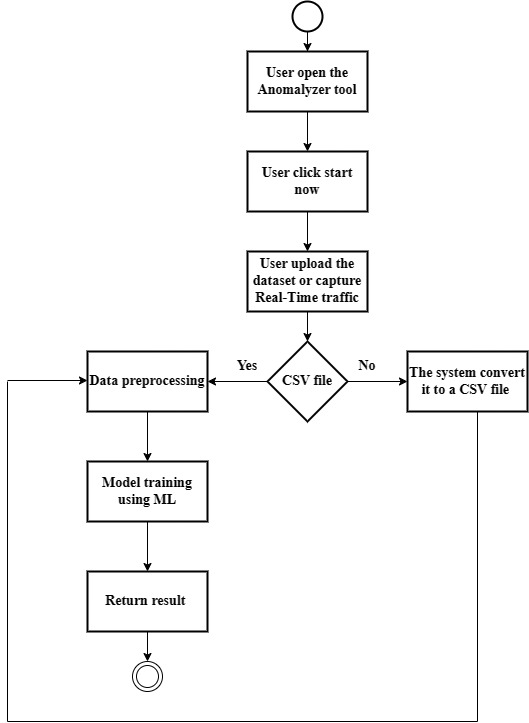
* **Deployment Diagram:**



**Figure ‎5.4: Deployment Diagram.**

The above diagram shows the physical deployment of software components across hardware nodes in the system. This diagram helps visualize the distribution of software components and their dependencies in the system environment.

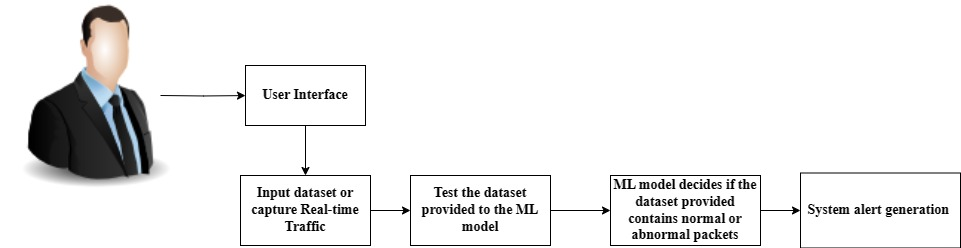
* **Activity Diagram:**



**Figure ‎5.5: Activity Diagram.**

The Anomalyzer tool is designed to make it easy for users by identifying network threats and to improve the results, reduced false positives and increased the accuracy by using Machine Learning. The Anomalyzer tool Activity Diagram shows the process when the user opens the tool, choose to uploads the dataset file or Real-Time traffic monitoring , if it's not a CSV file the tool convert it to CSV then it pre-process the dataset and train the Machine Learning model and returns the result.

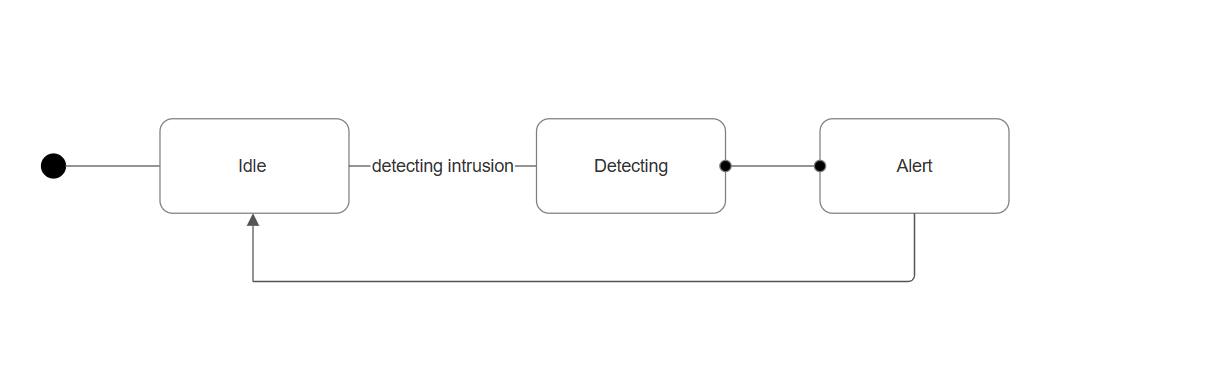
* **Sequence Diagram:**



**Figure ‎5.6: Sequence Diagram.**

The end user accesses the system with  the user friendly interface then the user inputs the dataset or Capture Real-time Traffic, the dataset is then tested using the ML model ,the model decides if the dataset contains anomaly, then it send an alert to clarify the state of this dataset.

* **State Transition Diagram:**

**Figure ‎5.7: State Transition Diagram.**

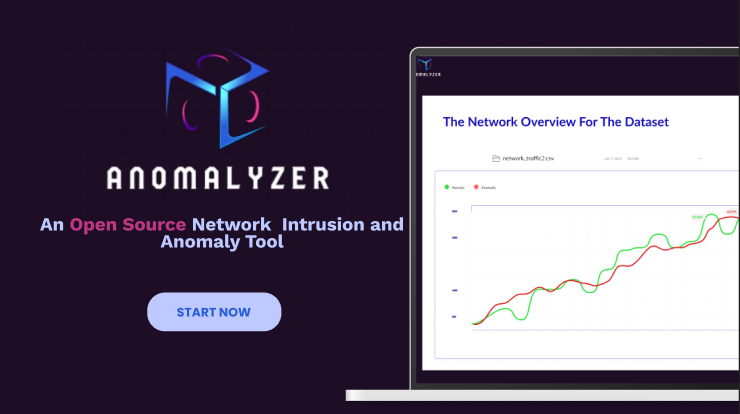
The state diagram above shows the transition in the system. First The initial state where the IDS is waiting for network traffic, The IDS is actively monitoring and analysing network traffic to detect any intrusion attempts, if an intrusion is detected, the system transitions to the Alert state, where it generates an alert or takes some action to notify administrators or respond to the intrusion, after handling the alert, the system returns to the Idle state to continue monitoring

## 5.2 Physical Model Design

### 5.2.1 User Interface Design

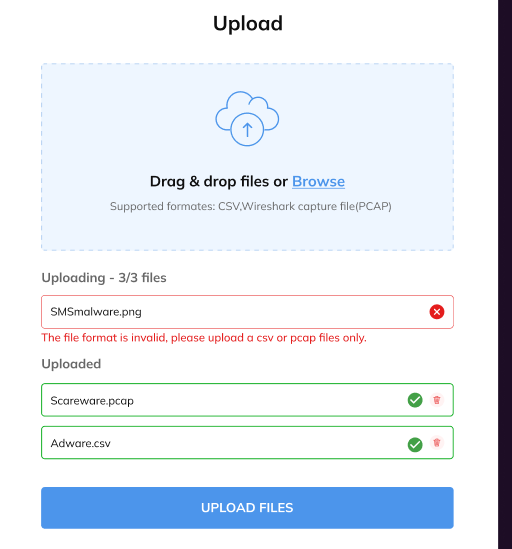
**Anomalyzer Dashboard**

Users can effortlessly test the network security with anmolayzer user-friendly interface.



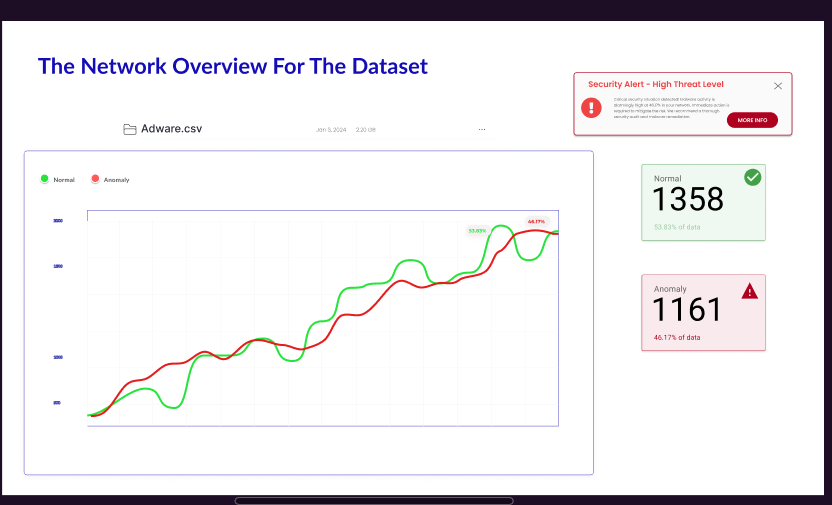
**Figure 5.8: Amomalyzer Dashboard**

The result of our tool **(Anomalyzer)** are shown on the user interface:



**Figure 5.9: Upload the dataset**

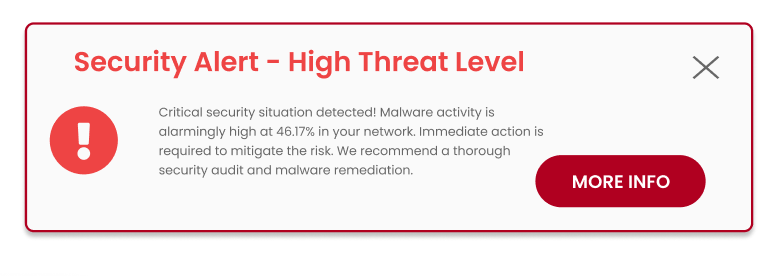
Amomalyzer tool allows users to upload datasets to their network, whether they are pcap files from Wireshark, and then the tool converts them to CSV files or ready-made CSV files, with a notification displayed when data sets or files in an unacceptable format are uploaded.



**Figure 5.10: Zoom in for the uploading screen**

The result screen shows the file name, type, and size, and an easy graph showing whether the data is normal (green) or anomaly data (red), along with the number of normal and anomaly data and their percentage of the original amount of data, as well as a notification describing the state of the data and the extent of its seriousness, with recommendations for dealing with it.

### 5.2.2 Reports Design



**Figure 5.11: Anomalyzer Alert**

Clarifying the form of the notification that explains the Threat level of the data and recommendations for dealing with it.

You can click on More Info to understand the meaning of the poem and the reason behind it.



**Figure 5.11: Anomalyzer Alert Meaning**

Based on the percentage of anomaly (abnormal) data that appeared, this screen explains the meaning of low, moderate, and high levels of the threat on the network, with recommendations for dealing with each of them.

# Chapter 6

# Implementation

## 6.1 General Implementation Description

* **Python:** Is a flexible and widely-used language with extensive libraries for data analysis, machine learning, and GUI development.
* **General Overview about Libraries and APIs Used:**

**PyQt5**: For creating the GUI components of the application (QtWidgets, QtCore, QtGui).

**Pyshark:** For packet capturing and network analysis.

**Pickle and Joblib:** For serialising and deserializing Python object structures.

**Pandas:** For data manipulation and analysis.

**Numpy:** For numerical operations.

**Matplotlib:** For plotting and visualising data.

**Scikit-learn:** For machine learning algorithms and preprocessing.

**OS Module:** Used for interacting with the operating system, such as file path manipulations and executing shell commands.

* **General Overview about Classes:**

**LoadingPage in capturepage.py:** Handles the loading screen and the monitoring process.

**Ui\_MainWindow in page1.py:** Set up the main window of the application.

**Ui\_SecondWindow in page2.py:** Set up the second window, including navigation to capture and upload pages.

**Ui\_MainWindow in uploadpage.py:** Handles file uploads and triggers the data cleaning and prediction script.

* **General Overview about Functions:**

**Monitoring Functions:** monitoring and monitoring\_now in monitoring.py: Captures network traffic and triggers feature extraction.

**Data Processing Functions:** feature\_selection\_cleaning and main in **newdata\_clean\_predict.py** clean and preprocess data, and perform predictions using a pre-trained model.

**Model Functions:** Various functions in isolation\_forest.py used to train and evaluate the Isolation Forest model for anomaly detection.

**Scripts: capturepage.py,page1.py, page2.py, uploadpage.py, monitoring.py, newdata\_clean\_predict.py, and isolation\_forest.py.**

**Lines of Code:** Approximate total lines of code across all files: 1000+ lines.

* **General Overview about GUI Pages:**

**Main Page (page1.py):** Main interface for starting the process.

from PyQt5 import QtCore, QtGui, QtWidgets

import page2  # Importing page2.py module

class Ui\_MainWindow(object):

def setupUi(self, MainWindow):

MainWindow.setObjectName("MainWindow")

MainWindow.resize(785, 493)

MainWindow.setStyleSheet("background-color: rgb(29, 14, 37);")

self.centralwidget = QtWidgets.QWidget(MainWindow)

self.centralwidget.setObjectName("centralwidget")

self.background = QtWidgets.QFrame(self.centralwidget)

self.background.setGeometry(QtCore.QRect(0, 0, 941, 561))

self.background.setCursor(QtGui.QCursor(QtCore.Qt.ArrowCursor))

self.background.setToolTipDuration(2)

self.background.setAutoFillBackground(False)

self.background.setFrameShape(QtWidgets.QFrame.StyledPanel)

self.background.setFrameShadow(QtWidgets.QFrame.Raised)

self.background.setObjectName("background")

self.logoandoutput = QtWidgets.QWidget(self.background)

self.logoandoutput.setGeometry(QtCore.QRect(20, 20, 751, 471))

self.logoandoutput.setObjectName("logoandoutput")

self.background\_label = QtWidgets.QLabel(self.logoandoutput)

self.background\_label.setGeometry(QtCore.QRect(0, 0, 751, 471))

self.background\_label.setPixmap(QtGui.QPixmap('page1.png'))

self.background\_label.setScaledContents(True)

self.background\_label.setObjectName("background\_label")

self.pushButton\_3 = QtWidgets.QPushButton(self.logoandoutput)

self.pushButton\_3.setGeometry(QtCore.QRect(140, 360, 141, 51))

font = QtGui.QFont()

font.setPointSize(11)

font.setBold(True)

font.setWeight(75)

self.pushButton\_3.setFont(font)

self.pushButton\_3.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))

self.pushButton\_3.setStyleSheet("""

QPushButton {

background-color: #BDC9FF;

color: #285FD7;

border-radius: 23px;

padding: 10px 20px;

}

""")

self.pushButton\_3.setObjectName("pushButton\_3")

self.pushButton\_3.clicked.connect(self.open\_page2)  # Connect clicked signal to open\_page2 method

self.pushButton\_3.raise\_()

MainWindow.setCentralWidget(self.centralwidget)

self.statusbar = QtWidgets.QStatusBar(MainWindow)

self.statusbar.setObjectName("statusbar")

MainWindow.setStatusBar(self.statusbar)

self.retranslateUi(MainWindow)

QtCore.QMetaObject.connectSlotsByName(MainWindow)

def retranslateUi(self, MainWindow):

\_translate = QtCore.QCoreApplication.translate

MainWindow.setWindowTitle(\_translate("MainWindow", "MainWindow"))

self.pushButton\_3.setText(\_translate("MainWindow", "Start Now"))

def open\_page2(self):

self.window = QtWidgets.QMainWindow()

self.ui = page2.Ui\_SecondWindow()  # Initialize the UI from page2.py

self.ui.setupUi(self.window)

self.window.show()

if \_\_name\_\_ == "\_\_main\_\_":

import sys

app = QtWidgets.QApplication(sys.argv)

MainWindow = QtWidgets.QMainWindow()

ui = Ui\_MainWindow()

ui.setupUi(MainWindow)

MainWindow.show()

sys.exit(app.exec\_())

**Capture Page (capturepage.py):** Displays the status of traffic capture.

from PyQt5 import QtCore, QtGui, QtWidgets

import subprocess

class LoadingPage(QtWidgets.QWidget):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.setWindowTitle("Capturing...")

        self.resize(300, 150)

        self.setStyleSheet("background-color: rgb(29, 14, 37); color: white;")

        layout = QtWidgets.QVBoxLayout(self)

        self.timer\_label = QtWidgets.QLabel("", self)

        self.timer\_label.setAlignment(QtCore.Qt.AlignCenter)

        layout.addWidget(self.timer\_label)

        self.loading\_label = QtWidgets.QLabel("", self)

        self.loading\_label.setAlignment(QtCore.Qt.AlignCenter)

        layout.addWidget(self.loading\_label)

        subprocess.run(["python", "monitoring.py"])

        # Start the timer for 10 seconds

        self.timer = QtCore.QTimer(self)

        self.timer.timeout.connect(self.update\_timer)

        self.timer\_interval = 1000  # 1 second

        self.timer\_count = 5  # Timer count set to 10 seconds

        self.timer.start(self.timer\_interval)

        # Button for showing result

        self.result\_button = QtWidgets.QPushButton("Result", self)

        self.result\_button.setStyleSheet("""

            QPushButton {

                background-color: #BDC9FF;

                color: blue;

                border-radius: 30px;

                padding: 10px 20px;

                border: none;

                font-weight: bold;

                font-size: 14px;

            }

        """)

        self.result\_button.clicked.connect(self.show\_result)

        layout.addWidget(self.result\_button)

        # Hide the result button initially

        self.result\_button.hide()

    def update\_timer(self):

        self.timer\_count -= 1

        self.timer\_label.setText(f"Time remaining: {self.timer\_count} seconds")

        if self.timer\_count <= 0:

            self.timer.stop()

            self.loading\_label.setText("Capturing Complete!")

            self.result\_button.show()

    def show\_result(self):

        # Open result.py script using subprocess

        subprocess.run(["python", "result.py"]) #update the path

    def mousePressEvent(self, event):

        if event.button() == QtCore.Qt.LeftButton:

            QtWidgets.QApplication.setOverrideCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))

    def mouseReleaseEvent(self, event):

        QtWidgets.QApplication.restoreOverrideCursor()

if \_\_name\_\_ == "\_\_main\_\_":

    import sys

    app = QtWidgets.QApplication(sys.argv)

    loading\_page = LoadingPage()

    loading\_page.show()

    sys.exit(app.exec\_())

**Upload Page (uploadpage.py):** Allows users to upload PCAP files for analysis.

from PyQt5 import QtCore, QtGui, QtWidgets

import os

from PyQt5.QtGui import QDesktopServices, QIcon

from PyQt5.QtCore import QUrl

import subprocess

class Ui\_MainWindow(object):

    def setupUi(self, MainWindow):

        MainWindow.setObjectName("MainWindow")

        MainWindow.resize(821, 576)

        self.centralwidget = QtWidgets.QWidget(MainWindow)

        self.centralwidget.setObjectName("centralwidget")

        self.bg = QtWidgets.QFrame(self.centralwidget)

        self.bg.setGeometry(QtCore.QRect(20, -10, 791, 571))

        self.bg.setStyleSheet("background-color: rgb(29, 13, 37);")

        self.bg.setFrameShape(QtWidgets.QFrame.StyledPanel)

        self.bg.setFrameShadow(QtWidgets.QFrame.Raised)

        self.bg.setObjectName("bg")

        self.loge = QtWidgets.QFrame(self.bg)

        self.loge.setGeometry(QtCore.QRect(40, 90, 651, 421))

        self.loge.setStyleSheet("background-color: rgb(255, 255, 255);")

        self.loge.setFrameShape(QtWidgets.QFrame.StyledPanel)

        self.loge.setFrameShadow(QtWidgets.QFrame.Raised)

        self.loge.setObjectName("loge")

        self.uploadscreen = QtWidgets.QFrame(self.loge)

        self.uploadscreen.setGeometry(QtCore.QRect(70, 30, 541, 201))

        self.uploadscreen.setStyleSheet("border-color: rgb(85, 0, 255);\n"

                                         "background-color: rgb(238, 246, 255);")

        self.uploadscreen.setFrameShape(QtWidgets.QFrame.StyledPanel)

        self.uploadscreen.setFrameShadow(QtWidgets.QFrame.Raised)

        self.uploadscreen.setObjectName("uploadscreen")

        self.label\_3 = QtWidgets.QLabel(self.uploadscreen)

        self.label\_3.setGeometry(QtCore.QRect(160, -20, 221, 151))

        self.label\_3.setAutoFillBackground(False)

        self.label\_3.setText("")

        self.label\_3.setPixmap(QtGui.QPixmap("../../Pictures/Screenshots/Screenshot 2024-04-12 145829.png"))

        self.label\_3.setScaledContents(True)

        self.label\_3.setObjectName("label\_3")

        self.label = QtWidgets.QLabel(self.uploadscreen)

        self.label.setGeometry(QtCore.QRect(30, 70, 551, 41))

        self.label.setStyleSheet("color: rgb(186, 186, 186); font: 75 14pt \'MS Shell Dlg 2\';")

        self.label.setObjectName("label")

        # Browse Button

        self.browse = QtWidgets.QPushButton(self.uploadscreen)

        self.browse.setGeometry(QtCore.QRect(200, 120, 150, 60))  # Larger size

        self.browse.setStyleSheet("QPushButton {background-color: #BDC9FF; font: bold 12pt \"MS Shell Dlg 2\"; color: #285FD7; border-radius: 8px; padding: 15px 25px;}"

                                  "QPushButton:hover {background-color:#79BAEC; cursor: hand;}"

                                  "QPushButton:pressed {background-color: #79BAEC;}")

        self.browse.setObjectName("browse")

        self.browse.setText("Browse")

        # Connect browse\_files to browse button click event

        self.browse.clicked.connect(self.browse\_files)

        self.listWidget = QtWidgets.QListWidget(self.loge)

        self.listWidget.setGeometry(QtCore.QRect(70, 240, 541, 121))

        self.listWidget.setStyleSheet("border-color: rgb(0, 0, 255);")

        self.listWidget.setObjectName("listWidget")

        self.pushButton\_submit = QtWidgets.QPushButton(self.loge)

        self.pushButton\_submit.setGeometry(QtCore.QRect(540, 380, 81, 30))

        self.pushButton\_submit.setMinimumSize(QtCore.QSize(0, 30))

        self.pushButton\_submit.setStyleSheet("QPushButton {border: none; border-radius: 8px; background-color: #82CAFF; font: 12pt; color: #285FD7;}"

                                              "QPushButton:hover {background-color:#79BAEC;}"

                                              "QPushButton:pressed {background-color: #79BAEC;}")

        self.pushButton\_submit.setObjectName("pushButton\_submit")

        # Connect upload\_files to upload button click event

        self.pushButton\_submit.clicked.connect(self.upload\_files)

        self.pushButton\_clear = QtWidgets.QPushButton(self.loge)

        self.pushButton\_clear.setGeometry(QtCore.QRect(450, 380, 81, 30))

        self.pushButton\_clear.setMinimumSize(QtCore.QSize(0, 30))

        self.pushButton\_clear.setStyleSheet("QPushButton {border: none; border-radius: 8px; background-color: #FFA07A; font: 12pt; color: #FF0000;}"

                                             "QPushButton:hover {background-color:#F9966B;}"

                                             "QPushButton:pressed {background-color: #F9966B;}")

        self.pushButton\_clear.setObjectName("pushButton\_clear")

        # Connect clear\_list to clear button click event

        self.pushButton\_clear.clicked.connect(self.clear\_list)

        # Result Button

        self.result\_button = QtWidgets.QPushButton(self.loge)

        self.result\_button.setGeometry(QtCore.QRect(360, 380, 81, 30))

        self.result\_button.setMinimumSize(QtCore.QSize(0, 30))

        self.result\_button.setStyleSheet("QPushButton {border: none; border-radius: 8px; background-color: #BDC9FF; font: 12pt; color: #285FD7;}"

                                          "QPushButton:hover {background-color:#79BAEC;}"

                                          "QPushButton:pressed {background-color: #79BAEC;}")

        self.result\_button.setObjectName("result\_button")

        self.result\_button.setText("Result")

        self.result\_button.setVisible(False)  # Hide the button initially

        # Connect show\_result to result button click event

        self.result\_button.clicked.connect(self.show\_result)

        MainWindow.setCentralWidget(self.centralwidget)

        self.menubar = QtWidgets.QMenuBar(MainWindow)

        self.menubar.setGeometry(QtCore.QRect(0, 0, 821, 21))

        self.menubar.setObjectName("menubar")

        MainWindow.setMenuBar(self.menubar)

        self.statusbar = QtWidgets.QStatusBar(MainWindow)

        self.statusbar.setObjectName("statusbar")

        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)

        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):

        \_translate = QtCore.QCoreApplication.translate

        MainWindow.setWindowTitle(\_translate("MainWindow", "MainWindow"))

        self.label.setText(\_translate("MainWindow", "  Supported formats: Packet Capture File (PCAP)"))

        self.browse.setText(\_translate("MainWindow","Browse"))

        self.pushButton\_submit.setText(\_translate("MainWindow", "Upload"))

        self.pushButton\_clear.setText(\_translate("MainWindow", "Clear list"))

    def browse\_files(self):

        file\_dialog = QtWidgets.QFileDialog()

        files, \_ = file\_dialog.getOpenFileNames(None, "Select File", "", "All Files (\*)")

        for file in files:

            self.listWidget.addItem(file)

    def upload\_files(self):

        file\_paths = [self.listWidget.item(i).text() for i in range(self.listWidget.count())]  # Get paths of uploaded files

        if not file\_paths:

            print("No file selected.")

            return

        # Save the selected file path to a temporary text file

        with open("selected\_file.txt", "w") as f:

            f.write(file\_paths[0])

        # Execute the newdata\_clean\_predict.py script

        subprocess.run(["python", "TSharkFeatureExtractionVer2.py"])

        # Show the Result button after uploading the file

        self.result\_button.setVisible(True)

    def clear\_list(self):

        self.listWidget.clear()

        self.result\_button.setVisible(False)  # Hide the Result button when clearing the list

    def show\_result(self):

        # Open result.py script using subprocess

        subprocess.run(["python", "result.py"])

if \_\_name\_\_ == "\_\_main\_\_":

    import sys

    app = QtWidgets.QApplication(sys.argv)

    MainWindow = QtWidgets.QMainWindow()

    ui = Ui\_MainWindow()

    ui.setupUi(MainWindow)

    MainWindow.show()

    sys.exit(app.exec\_())

**Secondary Page (page2.py):** Provides options to capture traffic or upload a file.

from PyQt5 import QtCore, QtGui, QtWidgets

import capturepage  # Importing the capturepage module

import uploadpage

class Ui\_SecondWindow(object):

    def setupUi(self, SecondWindow):

        SecondWindow.setObjectName("SecondWindow")

        SecondWindow.resize(800, 600)

        SecondWindow.setStyleSheet("background-color: rgb(29, 14, 37);")

        self.centralwidget = QtWidgets.QWidget(SecondWindow)

        self.centralwidget.setObjectName("centralwidget")

        # Background frame

        self.background = QtWidgets.QFrame(self.centralwidget)

        self.background.setGeometry(QtCore.QRect(0, 0, 800, 600))

        self.background.setCursor(QtGui.QCursor(QtCore.Qt.ArrowCursor))

        self.background.setToolTipDuration(2)

        self.background.setAutoFillBackground(False)

        self.background.setFrameShape(QtWidgets.QFrame.StyledPanel)

        self.background.setFrameShadow(QtWidgets.QFrame.Raised)

        self.background.setObjectName("background")

        # Background image

        self.background\_label = QtWidgets.QLabel(self.background)

        self.background\_label.setGeometry(QtCore.QRect(0, 0, 800, 600))

        self.background\_label.setPixmap(QtGui.QPixmap('page2.png'))

        self.background\_label.setScaledContents(True)

        self.background\_label.setObjectName("background\_label")

        # Text box above buttons

        self.textEdit = QtWidgets.QTextEdit(self.background)

        self.textEdit.setGeometry(QtCore.QRect(200, 350, 400, 30))  # Adjusted position and size as needed

        self.textEdit.setStyleSheet("background-color: white; border: 1px solid white; color: #182EF9;")

        self.textEdit.setObjectName("textEdit")

        self.textEdit.setReadOnly(True)

        self.textEdit.setAlignment(QtCore.Qt.AlignCenter)

        self.textEdit.setFont(QtGui.QFont("Arial", 10, QtGui.QFont.Bold))

        self.textEdit.setText("     Capture Your Traffic Or Upload A PCAP File")

        # Capture button

        self.captureButton = QtWidgets.QPushButton(self.background)

        self.captureButton.setGeometry(QtCore.QRect(200, 400, 400, 50))  # Adjusted y position to 400

        font = QtGui.QFont()

        font.setPointSize(11)

        font.setBold(True)

        font.setWeight(75)

        self.captureButton.setFont(font)

        self.captureButton.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))

        self.captureButton.setStyleSheet("""

            QPushButton {

                background-color: #BDC9FF;

                color: #285FD7;

                border-radius: 10px;

                padding: 10px 20px;

            }

        """)

        self.captureButton.setObjectName("capture")

        self.captureButton.setText("Capture Real Traffic")

        # Connect capture button to open capturepage.py

        self.captureButton.clicked.connect(self.open\_capture\_page)

        # Upload button

        self.uploadButton = QtWidgets.QPushButton(self.background)

        self.uploadButton.setGeometry(QtCore.QRect(200, 500, 400, 50))  # Adjusted y position to 500

        self.uploadButton.setFont(font)

        self.uploadButton.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))

        self.uploadButton.setStyleSheet("""

            QPushButton {

                background-color: #BDC9FF;

                color: #285FD7;

                border-radius: 10px;

                padding: 10px 20px;

            }

        """)

        self.uploadButton.setObjectName("pcapcsv")

        self.uploadButton.setText("Upload a PCAP file")

        # Connect upload button to open uploadpage.py

        self.uploadButton.clicked.connect(self.open\_upload\_page)

        SecondWindow.setCentralWidget(self.centralwidget)

        self.statusbar = QtWidgets.QStatusBar(SecondWindow)

        self.statusbar.setObjectName("statusbar")

        SecondWindow.setStatusBar(self.statusbar)

        self.retranslateUi(SecondWindow)

        QtCore.QMetaObject.connectSlotsByName(SecondWindow)

    def retranslateUi(self, SecondWindow):

        \_translate = QtCore.QCoreApplication.translate

        SecondWindow.setWindowTitle(\_translate("SecondWindow", "SecondWindow"))

    def open\_capture\_page(self):

        self.window = QtWidgets.QMainWindow()

        self.ui = capturepage.LoadingPage()  # Changed to LoadingPage as per capturepage.py

        self.ui.show()  # Changed to show() method

    def open\_upload\_page(self):

        self.window = QtWidgets.QMainWindow()

        self.ui = uploadpage.Ui\_MainWindow()  # Changed to Ui\_MainWindow as per uploadpage.py

        self.ui.setupUi(self.window)  # Initialize the upload page UI

        self.window.show()

if \_\_name\_\_ == "\_\_main\_\_":

    import sys

    app = QtWidgets.QApplication(sys.argv)

    SecondWindow = QtWidgets.QMainWindow()

    ui = Ui\_SecondWindow()

    ui.setupUi(SecondWindow)

    SecondWindow.show()

    sys.exit(app.exec\_())

**Result Page (result.py):** Display the result for the ids system.

import sys

from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QLabel, QFrame, QHBoxLayout, QSpacerItem, QSizePolicy

from PyQt5.QtGui import QPixmap, QFont

from PyQt5.QtCore import Qt, QTimer

import matplotlib.pyplot as plt

import pandas as pd

import joblib

import pickle

import subprocess

import os

class ResultPage(QWidget):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.initUI()

    def initUI(self):

        # Window settings

        self.setWindowTitle('Result Page')

        self.setGeometry(100, 100, 1600, 900)

        self.setStyleSheet("background-color: #1D0E25;")

        # Main layout

        main\_layout = QVBoxLayout()

        # Logo

        logo\_label = QLabel(self)

        pixmap = QPixmap('logo.png').scaled(408, 386, Qt.KeepAspectRatio)

        logo\_label.setPixmap(pixmap)

        main\_layout.addWidget(logo\_label, alignment=Qt.AlignLeft | Qt.AlignTop)

        # Container for plot and boxes

        container\_layout = QHBoxLayout()

        # Plot

        plot\_label = QLabel(self)

        plot\_pixmap = QPixmap('plot.png').scaled(1000, 700, Qt.KeepAspectRatio)

        plot\_label.setPixmap(plot\_pixmap)

        container\_layout.addWidget(plot\_label, alignment=Qt.AlignLeft)

        # Add a spacer item to control spacing between plot and boxes

        container\_layout.addItem(QSpacerItem(20, 700, QSizePolicy.Minimum, QSizePolicy.Expanding))

        # Boxes for counts and percentages

        boxes\_layout = QVBoxLayout()

        # Green Box - Normal Data

        normal\_data\_box = QFrame(self)

        normal\_data\_box.setFixedSize(574, 281)

        normal\_data\_box.setStyleSheet("background-color: #E0F2E4; border: 2px solid green;")

        normal\_data\_layout = QVBoxLayout(normal\_data\_box)

        normal\_label = QLabel("Normal:")

        normal\_label.setFont(QFont('Arial', 14))

        normal\_label.setStyleSheet("color: green;")

        self.normal\_count\_label = QLabel()

        self.normal\_count\_label.setFont(QFont('Arial', 12))

        self.normal\_count\_label.setStyleSheet("color: gray;")

        self.normal\_percentage\_label = QLabel()

        self.normal\_percentage\_label.setFont(QFont('Arial', 12))

        self.normal\_percentage\_label.setStyleSheet("color: green;")

        normal\_data\_layout.addWidget(normal\_label)

        normal\_data\_layout.addWidget(self.normal\_count\_label)

        normal\_data\_layout.addWidget(self.normal\_percentage\_label)

        normal\_data\_box.setLayout(normal\_data\_layout)

        # Red Box - Malware Data

        malware\_data\_box = QFrame(self)

        malware\_data\_box.setFixedSize(574, 281)

        malware\_data\_box.setStyleSheet("background-color: #F9EBED; border: 2px solid red;")

        malware\_data\_layout = QVBoxLayout(malware\_data\_box)

        malware\_label = QLabel("Malware:")

        malware\_label.setFont(QFont('Arial', 14))

        malware\_label.setStyleSheet("color: red;")

        self.malware\_count\_label = QLabel()

        self.malware\_count\_label.setFont(QFont('Arial', 12))

        self.malware\_count\_label.setStyleSheet("color: gray;")

        self.malware\_percentage\_label = QLabel()

        self.malware\_percentage\_label.setFont(QFont('Arial', 12))

        self.malware\_percentage\_label.setStyleSheet("color: red;")

        malware\_data\_layout.addWidget(malware\_label)

        malware\_data\_layout.addWidget(self.malware\_count\_label)

        malware\_data\_layout.addWidget(self.malware\_percentage\_label)

        malware\_data\_box.setLayout(malware\_data\_layout)

        boxes\_layout.addWidget(normal\_data\_box)

        boxes\_layout.addWidget(malware\_data\_box)

        container\_layout.addLayout(boxes\_layout)

        main\_layout.addLayout(container\_layout)

        self.setLayout(main\_layout)

        # Update counts and percentages

        self.update\_counts\_and\_percentages()

        # Start a QTimer to call the alert window after 30 seconds

        self.timer = QTimer(self)

        self.timer.setSingleShot(True)

        self.timer.timeout.connect(self.show\_alert)

        self.timer.start(30000)  # 30 seconds

    def update\_counts\_and\_percentages(self):

        print("Updating counts and percentages...")  # Debug statement

        df = pd.read\_csv('output.csv', on\_bad\_lines='skip', engine='python', encoding='ISO-8859-1')

        new\_data = self.feature\_selection\_cleaning(df)

        with open('isolation\_forest\_model.pkl', 'rb') as file:

            model = joblib.load(file)

        prediction = model.predict(new\_data)

        benign\_count = (prediction == 1).sum()

        malware\_count = (prediction == -1).sum()

        total\_count = benign\_count + malware\_count

        benign\_percentage = (benign\_count / total\_count) \* 100 if total\_count > 0 else 0

        malware\_percentage = (malware\_count / total\_count) \* 100 if total\_count > 0 else 0

        self.normal\_count\_label.setText(f"{benign\_count}")

        self.normal\_percentage\_label.setText(f"{benign\_percentage:.2f}%")

        self.malware\_count\_label.setText(f"{malware\_count}")

        self.malware\_percentage\_label.setText(f"{malware\_percentage:.2f}%")

        # Generate the plot and save it as plot.png

        plt.figure(figsize=(12, 7))

        plt.bar(['Benign', 'Malware'], [benign\_count, malware\_count], color=['blue', 'red'])

        plt.title('Number of Benign and Malware Instances')

        plt.xlabel('Class')

        plt.ylabel('Count')

        plt.savefig('plot.png')

        print("Counts and percentages updated.")  # Debug statement

    def feature\_selection\_cleaning(self, df):

        print("Performing feature selection and cleaning...")  # Debug statement

        DataFrame = df.copy()

        df\_num = DataFrame[['udp.srcport','frame.len','ip.ttl','ip.dsfield','tcp.dstport']].astype('float64')

        df\_cat = DataFrame[['frame.time\_epoch','frame.time','ip.frag\_offset','ip.src','ip.version',

                       'ip.checksum','tcp.flags.reset','tcp.flags.syn','tcp.window\_size',

                       'tcp.flags.fin','ip.proto','eth.dst','frame.protocols','ip.len','tcp.seq']].astype("object")

        with open('imputer\_num.pickle', 'rb') as file:

            loaded\_model = pickle.load(file)

        x = loaded\_model.transform(df\_num)

        df\_num = pd.DataFrame(x, columns=df\_num.columns)

        with open('stsc.pickle', 'rb') as file:

            loaded\_model = pickle.load(file)

        x = loaded\_model.transform(df\_num)

        df\_num = pd.DataFrame(x, columns=df\_num.columns)

        with open('imputer\_cat.pickle', 'rb') as file:

            loaded\_model = pickle.load(file)

        x = loaded\_model.transform(df\_cat)

        df\_cat = pd.DataFrame(x, columns=df\_cat.columns)

        with open('cat\_enc.pickle', 'rb') as file:   # Handle unknown categories using OrdinalEncoder

            loaded\_model = pickle.load(file)

        x = loaded\_model.transform(df\_cat)

        df\_cat = pd.DataFrame(x, columns=df\_cat.columns)

        new\_data = pd.concat([df\_num, df\_cat], axis=1)

        new\_data.reset\_index(drop=True, inplace=True)

        print("Feature selection and cleaning done.")  # Debug statement

        return new\_data

    def show\_alert(self):

        print("Calling alert.py...")  # Debug statement

        try:

            alert\_script\_path = "alert.py"

            if os.path.exists(alert\_script\_path):

                subprocess.run(["python", alert\_script\_path])

                print("alert.py executed.")  # Debug statement

            else:

                print(f"alert.py not found at path: {alert\_script\_path}")

        except Exception as e:

            print(f"Error executing alert.py: {e}")  # Debug statement

if \_\_name\_\_ == '\_\_main\_\_':

    app = QApplication(sys.argv)

    result\_page = ResultPage()

    result\_page.show()

    sys.exit(app.exec\_())

* **General Overview about Coding Conventions:**

**Variables and functions Names :** We used snake\_case

**Class Names:** we used CamelCase.

**Comments and Documentation:** To provide clarity and understanding of the code.

**Error Handling:** To manage exceptions and ensure the program handles errors correctly.

**Whitespace and Formatting:** Consistent use of whitespace improves code readability

## 6.2 Pipeline Implementation Description

1. **Dataset preprocessing**

* Script: ***Feature\_selection.py*** and ***Data\_Cleaning.py***
* Process: prepare the dataset by applying all necessary preprocessing and select the best features.

**Feature\_selection.py Code:**

from sklearn.utils import shuffle

import pandas as pd

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import StandardScaler

from sklearn.preprocessing import OrdinalEncoder

from sklearn.model\_selection import train\_test\_split

# Splitting the data into the train and test sets

def data\_splitting(DataFrame):

    df\_train, df\_test = train\_test\_split(DataFrame, test\_size = 0.2, stratify = DataFrame['Label'], random\_state = 42)

    return df\_train,df\_test

# Find the most correlated features

def most\_corr\_features(df):

    print("Checking the correlation:")

    encoding = OrdinalEncoder()

    df['encoded\_labelll'] = encoding.fit\_transform(df[['Label']])

    corr\_matrix = df.corr(numeric\_only=True)

    corr\_feature = corr\_matrix['encoded\_labelll'].abs().sort\_values(ascending=False)

    cols = []

    for i in range(1, 21):

        most\_correlated\_feature = corr\_feature.index[i]

        cols.append(most\_correlated\_feature)

        print("The most correlated feature: ", most\_correlated\_feature, '=', corr\_feature[i])

    return cols

# Initial preprocessing for the feature selection process

def prep\_for\_feture\_selection(df):

    df = df.copy()

    # Drop unnamed features

    df.drop(df.columns[df.columns.str.contains('unnamed', case=False)], axis=1, inplace=True)

    Labels = df['Label'].copy()  # separating labels from df for preprocessing

    del df['Label']

    # Remove columns with all null values

    df = df.drop(['smtp.data.fragment', 'pop.request.command', 'pop.response', 'imap.request.command',

                  'imap.response', 'ftp.request.command', 'ftp.request.arg', 'ftp.response.code',

                  'ftp.response.arg'], axis=1)

    # Separate categorical data from numerical data

    df\_cat = df.select\_dtypes(include=['object'])

    df\_num = df.select\_dtypes(exclude=['object'])

    df\_num.reset\_index(drop=True, inplace=True)

    df\_cat.reset\_index(drop=True, inplace=True)

    # Use simple imputer to fill missing values in numerical data

    imputer = SimpleImputer(strategy="mean")

    X = imputer.fit\_transform(df\_num)

    df\_num = pd.DataFrame(X, columns=df\_num.columns)

    # Scale the numerical data

    stsc = StandardScaler()

    n = stsc.fit\_transform(df\_num)

    df\_num = pd.DataFrame(n, columns=df\_num.columns)

    df\_num.reset\_index(drop=True, inplace=True)

    # Fill in the missing values in the categorical data

    imputer = SimpleImputer(strategy="most\_frequent")

    y = imputer.fit\_transform(df\_cat)

    df\_cat = pd.DataFrame(y, columns=df\_cat.columns)

    # Use ordinal encoder to change data types

    oenc = OrdinalEncoder()

    cat\_encoded = oenc.fit\_transform(df\_cat)

    df\_cat = pd.DataFrame(cat\_encoded, columns=df\_cat.columns)

    df\_cat.reset\_index(drop=True, inplace=True)

    # Concatenate all the preprocessed data and labels together

    l = pd.DataFrame(Labels, columns=['Label'])

    prepared\_Dataset = pd.concat([df\_num, df\_cat, l], axis=1)

    return prepared\_Dataset

# Read the main data set and select the best features for model training

if \_\_name\_\_=='\_\_main\_\_':

    df = pd.read\_csv('Dataset\_Small.csv', on\_bad\_lines='skip', engine='python', encoding='ISO-8859-1')

    df = shuffle(df)

    df.reset\_index(drop=True, inplace=True)

    df.info(verbose=True, show\_counts=True)

    train\_set, test\_set = data\_splitting(df)

    train\_set.reset\_index(drop=True, inplace=True)

    test\_set.reset\_index(drop=True, inplace=True)

    Train\_Data = prep\_for\_feture\_selection(train\_set)

    Train\_Data.reset\_index(drop=True, inplace=True)

    x = most\_corr\_features(Train\_Data)

**Data\_Cleaning.py Code:**

import pandas as pd

import pickle

import joblib

import matplotlib.pyplot as plt

# This function puts the new data through the neccessary preprocesing such as imputers and scalers

def feature\_selection\_cleaning(df):

    DataFrame = df.copy()

    df\_num = DataFrame[['udp.srcport','frame.len','ip.ttl','ip.dsfield','tcp.dstport']].astype('float64')

    df\_cat = DataFrame[['frame.time\_epoch','frame.time','ip.frag\_offset','ip.src','ip.version',

                       'ip.checksum','tcp.flags.reset','tcp.flags.syn','tcp.window\_size',

                       'tcp.flags.fin','ip.proto','eth.dst','frame.protocols','ip.len','tcp.seq']].astype("object")

    # Apply the numerical imputer

    with open('imputer\_num.pickle', 'rb') as file:

        loaded\_model = pickle.load(file)

    x = loaded\_model.transform(df\_num)

    df\_num = pd.DataFrame(x, columns=df\_num.columns)

    # Apply the scaler

    with open('stsc.pickle', 'rb') as file:

        loaded\_model = pickle.load(file)

    x = loaded\_model.transform(df\_num)

    df\_num = pd.DataFrame(x, columns=df\_num.columns)

    # Apply the categorical imputer

    with open('imputer\_cat.pickle', 'rb') as file:

        loaded\_model = pickle.load(file)

    x = loaded\_model.transform(df\_cat)

    df\_cat = pd.DataFrame(x, columns=df\_cat.columns)

    # Aplly the ordinal encoder

    with open('cat\_enc.pickle', 'rb') as file:

        loaded\_model = pickle.load(file)

    x = loaded\_model.transform(df\_cat)

    df\_cat = pd.DataFrame(x, columns=df\_cat.columns)

    new\_data = pd.concat([df\_num, df\_cat], axis=1)

    new\_data.reset\_index(drop=True, inplace=True)

    return new\_data

# This function reads the CSV file and calls the preprocessing function, then counts the amount of anomalies and normal packets, and creats and saves the visual plot for the prediction

def generate\_plot\_and\_stats():

    df = pd.read\_csv('output.csv', on\_bad\_lines='skip', engine='python', encoding='ISO-8859-1')

    new\_data = feature\_selection\_cleaning(df)

    # Apply the trained isolation forest model to the new data and predict how anomalous the traffic is

    with open('isolation\_forest\_model.pkl', 'rb') as file:

        model = joblib.load(file)

    prediction = model.predict(new\_data)

    # Count the number of benign and malware instances

    benign\_count = (prediction == 1).sum()

    malware\_count = (prediction == -1).sum()

    total\_count = benign\_count + malware\_count

    normal\_percentage = (benign\_count / total\_count) \* 100

    malware\_percentage = (malware\_count / total\_count) \* 100

    # Generate the plot

    plt.figure(figsize=(6, 6))

    plt.bar(['Normal', 'Malware'], [benign\_count, malware\_count], color=['green', 'red'])

    plt.title('Number of Normal and Malware Instances')

    plt.xlabel('Class')

    plt.ylabel('Count')

    plt.savefig('PLOT.png')

    plt.close()

    return benign\_count, normal\_percentage, malware\_count, malware\_percentage

def main():

   generate\_plot\_and\_stats()

1. **Model training:**

* Script: ***isolation\_forest.py***
* Process: train the isolation forest model using the best hyperparameters.

**isolation\_forest.py Code:**

import pandas as pd

from sklearn.ensemble import IsolationForest

from sklearn.metrics import classification\_report, accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score, confusion\_matrix

import joblib

import matplotlib.pyplot as plt

import numpy as np

# Load the datasets

train\_data = pd.read\_csv('Training\_Dataset.csv')

test\_data = pd.read\_csv('Testing\_Dataset.csv')

# Define the label mapping

label\_mapping = {'Benign': 1, 'Malware': -1}

# Map the labels in the train and test datasets

train\_data['Label'] = train\_data['Label'].replace(label\_mapping)

test\_data['Label'] = test\_data['Label'].replace(label\_mapping)

# Extract features and labels

X\_train = train\_data.drop(columns=['Label'])

y\_train = train\_data['Label']

X\_test = test\_data.drop(columns=['Label'])

y\_test = test\_data['Label']

# Initialize the Isolation Forest model with the best hyperparameters

iso\_forest = IsolationForest(n\_estimators=100, max\_samples='auto', contamination=0.05, max\_features=1.0, random\_state=42)

# Fit the model

iso\_forest.fit(X\_train)

# Predict on training set

y\_train\_pred = iso\_forest.predict(X\_train)

'''

from numpy import  where

anom\_index = where(y\_train\_pred==-1)

values = X\_train[anom\_index]

plt.scatter(X\_train[:,0], X\_train[:,1])

plt.scatter(values[:,0], values[:,1], color='r')

plt.show()

'''

# Predict on testing set

y\_pred = iso\_forest.predict(X\_test)

# Print out some evaluation metrics for training set

print("Training set performance:")

print(classification\_report(y\_train, y\_train\_pred))

# Convert predictions to binary format for training metrics

y\_train\_binary = (y\_train == 1).astype(int)

y\_train\_pred\_binary = (y\_train\_pred == 1).astype(int)

# Calculate training metrics

accuracy\_train = accuracy\_score(y\_train\_binary, y\_train\_pred\_binary)

precision\_train = precision\_score(y\_train\_binary, y\_train\_pred\_binary, zero\_division=0)

recall\_train = recall\_score(y\_train\_binary, y\_train\_pred\_binary, zero\_division=0)

f1\_train = f1\_score(y\_train\_binary, y\_train\_pred\_binary, zero\_division=0)

roc\_auc\_train = roc\_auc\_score(y\_train\_binary, y\_train\_pred\_binary)

cm = confusion\_matrix(y\_train\_binary, y\_train\_pred\_binary)

tn, fp, fn, tp = cm.ravel()

fnr\_test = fn / (fn + tp)

print(f'Accuracy = {accuracy\_train:.1%}')

print(f'ROC AUC = {roc\_auc\_train:.1%}')

print(f'Precision = {precision\_train:.1%}')

print(f'Recall = {recall\_train:.1%}')

print(f'F1 Score = {f1\_train:.1%}')

print("Confusion Matrix:\n", confusion\_matrix(y\_train\_binary, y\_train\_pred\_binary))

print(f'False Negative Rate (FNR) = {fnr\_test:.1%}')

# Print out some evaluation metrics for testing set

print("\nTesting set performance:")

print(classification\_report(y\_test, y\_pred))

# Convert predictions to binary format for testing metrics

y\_test\_binary = (y\_test == 1).astype(int)

y\_pred\_binary = (y\_pred == 1).astype(int)

# Calculate testing metrics

accuracy\_test = accuracy\_score(y\_test\_binary, y\_pred\_binary)

precision\_test = precision\_score(y\_test\_binary, y\_pred\_binary, zero\_division=0)

recall\_test = recall\_score(y\_test\_binary, y\_pred\_binary, zero\_division=0)

f1\_test = f1\_score(y\_test\_binary, y\_pred\_binary, zero\_division=0)

roc\_auc\_test = roc\_auc\_score(y\_test\_binary, y\_pred\_binary)

cm = confusion\_matrix(y\_test\_binary, y\_pred\_binary)

tn, fp, fn, tp = cm.ravel()

fnr\_test = fn / (fn + tp)

print(f'Accuracy = {accuracy\_test:.1%}')

print(f'ROC AUC = {roc\_auc\_test:.1%}')

print(f'Precision = {precision\_test:.1%}')

print(f'Recall = {recall\_test:.1%}')

print(f'F1 Score = {f1\_test:.1%}')

print("Confusion Matrix:\n", confusion\_matrix(y\_test\_binary, y\_pred\_binary))

print(f'False Negative Rate (FNR) = {fnr\_test:.1%}')

# Save the model to a file

joblib\_file = "isolation\_forest\_model.pkl"

joblib.dump(iso\_forest, joblib\_file)

# Count the number of benign and malware instances in the test set

benign\_count = (y\_test == 1).sum()

malware\_count = (y\_test == -1).sum()

# Adjusted counts for visualization

adjusted\_malware\_count = 50000  # Arbitrary large value to make the bar visible

adjusted\_benign\_count = benign\_count - adjusted\_malware\_count  # Adjust benign count accordingly

# Bar chart for the number of benign and malware instances

plt.figure(figsize=(6, 6))

plt.bar(['Benign', 'Malware'], [adjusted\_benign\_count, adjusted\_malware\_count], color=['green', 'red'])

plt.title('Number of Benign and Malware Instances in the Test Set')

plt.xlabel('Class')

plt.ylabel('Count')

plt.show()

# Print the number of benign and anomaly instances

print(f'Number of Benign instances: {benign\_count}')

print(f'Number of Malware instances: {malware\_count}')

1. **Data collection:**

* Script: ***uploadpage.py*** and ***capturepage.py*** and ***monitoring.py***
* Process: collect the PCAP file from the user or capture real time traffic and save the path of the PCAP file

**Uploadpage.py Code:**

from PyQt5 import QtCore, QtGui, QtWidgets

import os

from PyQt5.QtGui import QDesktopServices, QIcon

from PyQt5.QtCore import QUrl

import subprocess

class Ui\_MainWindow(object):

    def setupUi(self, MainWindow):

        MainWindow.setObjectName("MainWindow")

        MainWindow.resize(821, 576)

        self.centralwidget = QtWidgets.QWidget(MainWindow)

        self.centralwidget.setObjectName("centralwidget")

        self.bg = QtWidgets.QFrame(self.centralwidget)

        self.bg.setGeometry(QtCore.QRect(20, -10, 791, 571))

        self.bg.setStyleSheet("background-color: rgb(29, 13, 37);")

        self.bg.setFrameShape(QtWidgets.QFrame.StyledPanel)

        self.bg.setFrameShadow(QtWidgets.QFrame.Raised)

        self.bg.setObjectName("bg")

        self.loge = QtWidgets.QFrame(self.bg)

        self.loge.setGeometry(QtCore.QRect(40, 90, 651, 421))

        self.loge.setStyleSheet("background-color: rgb(255, 255, 255);")

        self.loge.setFrameShape(QtWidgets.QFrame.StyledPanel)

        self.loge.setFrameShadow(QtWidgets.QFrame.Raised)

        self.loge.setObjectName("loge")

        self.uploadscreen = QtWidgets.QFrame(self.loge)

        self.uploadscreen.setGeometry(QtCore.QRect(70, 30, 541, 201))

        self.uploadscreen.setStyleSheet("border-color: rgb(85, 0, 255);\n"

                                         "background-color: rgb(238, 246, 255);")

        self.uploadscreen.setFrameShape(QtWidgets.QFrame.StyledPanel)

        self.uploadscreen.setFrameShadow(QtWidgets.QFrame.Raised)

        self.uploadscreen.setObjectName("uploadscreen")

        self.label\_3 = QtWidgets.QLabel(self.uploadscreen)

        self.label\_3.setGeometry(QtCore.QRect(160, -20, 221, 151))

        self.label\_3.setAutoFillBackground(False)

        self.label\_3.setText("")

        self.label\_3.setPixmap(QtGui.QPixmap("../../Pictures/Screenshots/Screenshot 2024-04-12 145829.png"))

        self.label\_3.setScaledContents(True)

        self.label\_3.setObjectName("label\_3")

        self.label = QtWidgets.QLabel(self.uploadscreen)

        self.label.setGeometry(QtCore.QRect(30, 70, 551, 41))

        self.label.setStyleSheet("color: rgb(186, 186, 186); font: 75 14pt \'MS Shell Dlg 2\';")

        self.label.setObjectName("label")

        # Browse Button

        self.browse = QtWidgets.QPushButton(self.uploadscreen)

        self.browse.setGeometry(QtCore.QRect(200, 120, 150, 60))  # Larger size

        self.browse.setStyleSheet("QPushButton {background-color: #BDC9FF; font: bold 12pt \"MS Shell Dlg 2\"; color: #285FD7; border-radius: 8px; padding: 15px 25px;}"

                                  "QPushButton:hover {background-color:#79BAEC; cursor: hand;}"

                                  "QPushButton:pressed {background-color: #79BAEC;}")

        self.browse.setObjectName("browse")

        self.browse.setText("Browse")

        # Connect browse\_files to browse button click event

        self.browse.clicked.connect(self.browse\_files)

        self.listWidget = QtWidgets.QListWidget(self.loge)

        self.listWidget.setGeometry(QtCore.QRect(70, 240, 541, 121))

        self.listWidget.setStyleSheet("border-color: rgb(0, 0, 255);")

        self.listWidget.setObjectName("listWidget")

        self.pushButton\_submit = QtWidgets.QPushButton(self.loge)

        self.pushButton\_submit.setGeometry(QtCore.QRect(540, 380, 81, 30))

        self.pushButton\_submit.setMinimumSize(QtCore.QSize(0, 30))

        self.pushButton\_submit.setStyleSheet("QPushButton {border: none; border-radius: 8px; background-color: #82CAFF; font: 12pt; color: #285FD7;}"

                                              "QPushButton:hover {background-color:#79BAEC;}"

                                              "QPushButton:pressed {background-color: #79BAEC;}")

        self.pushButton\_submit.setObjectName("pushButton\_submit")

        # Connect upload\_files to upload button click event

        self.pushButton\_submit.clicked.connect(self.upload\_files)

        self.pushButton\_clear = QtWidgets.QPushButton(self.loge)

        self.pushButton\_clear.setGeometry(QtCore.QRect(450, 380, 81, 30))

        self.pushButton\_clear.setMinimumSize(QtCore.QSize(0, 30))

        self.pushButton\_clear.setStyleSheet("QPushButton {border: none; border-radius: 8px; background-color: #FFA07A; font: 12pt; color: #FF0000;}"

                                             "QPushButton:hover {background-color:#F9966B;}"

                                             "QPushButton:pressed {background-color: #F9966B;}")

        self.pushButton\_clear.setObjectName("pushButton\_clear")

        # Connect clear\_list to clear button click event

        self.pushButton\_clear.clicked.connect(self.clear\_list)

        # Result Button

        self.result\_button = QtWidgets.QPushButton(self.loge)

        self.result\_button.setGeometry(QtCore.QRect(360, 380, 81, 30))

        self.result\_button.setMinimumSize(QtCore.QSize(0, 30))

        self.result\_button.setStyleSheet("QPushButton {border: none; border-radius: 8px; background-color: #BDC9FF; font: 12pt; color: #285FD7;}"

                                          "QPushButton:hover {background-color:#79BAEC;}"

                                          "QPushButton:pressed {background-color: #79BAEC;}")

        self.result\_button.setObjectName("result\_button")

        self.result\_button.setText("Result")

        self.result\_button.setVisible(False)  # Hide the button initially

        # Connect show\_result to result button click event

        self.result\_button.clicked.connect(self.show\_result)

        MainWindow.setCentralWidget(self.centralwidget)

        self.menubar = QtWidgets.QMenuBar(MainWindow)

        self.menubar.setGeometry(QtCore.QRect(0, 0, 821, 21))

        self.menubar.setObjectName("menubar")

        MainWindow.setMenuBar(self.menubar)

        self.statusbar = QtWidgets.QStatusBar(MainWindow)

        self.statusbar.setObjectName("statusbar")

        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)

        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):

        \_translate = QtCore.QCoreApplication.translate

        MainWindow.setWindowTitle(\_translate("MainWindow", "MainWindow"))

        self.label.setText(\_translate("MainWindow", "  Supported formats: Packet Capture File (PCAP)"))

        self.browse.setText(\_translate("MainWindow","Browse"))

        self.pushButton\_submit.setText(\_translate("MainWindow", "Upload"))

        self.pushButton\_clear.setText(\_translate("MainWindow", "Clear list"))

    def browse\_files(self):

        file\_dialog = QtWidgets.QFileDialog()

        files, \_ = file\_dialog.getOpenFileNames(None, "Select File", "", "All Files (\*)")

        for file in files:

            self.listWidget.addItem(file)

    def upload\_files(self):

        file\_paths = [self.listWidget.item(i).text() for i in range(self.listWidget.count())]  # Get paths of uploaded files

        if not file\_paths:

            print("No file selected.")

            return

        # Save the selected file path to a temporary text file

        with open("selected\_file.txt", "w") as f:

            f.write(file\_paths[0])

        # Execute the newdata\_clean\_predict.py script

        subprocess.run(["python", "TSharkFeatureExtractionVer2.py"])

        # Show the Result button after uploading the file

        self.result\_button.setVisible(True)

    def clear\_list(self):

        self.listWidget.clear()

        self.result\_button.setVisible(False)  # Hide the Result button when clearing the list

    def show\_result(self):

        # Open result.py script using subprocess

        subprocess.run(["python", "result.py"])

if \_\_name\_\_ == "\_\_main\_\_":

    import sys

    app = QtWidgets.QApplication(sys.argv)

    MainWindow = QtWidgets.QMainWindow()

    ui = Ui\_MainWindow()

    ui.setupUi(MainWindow)

    MainWindow.show()

    sys.exit(app.exec\_())

**capturepage.py Code:**

from PyQt5 import QtCore, QtGui, QtWidgets

import subprocess

class LoadingPage(QtWidgets.QWidget):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.setWindowTitle("Capturing...")

        self.resize(300, 150)

        self.setStyleSheet("background-color: rgb(29, 14, 37); color: white;")

        layout = QtWidgets.QVBoxLayout(self)

        self.timer\_label = QtWidgets.QLabel("", self)

        self.timer\_label.setAlignment(QtCore.Qt.AlignCenter)

        layout.addWidget(self.timer\_label)

        self.loading\_label = QtWidgets.QLabel("", self)

        self.loading\_label.setAlignment(QtCore.Qt.AlignCenter)

        layout.addWidget(self.loading\_label)

        subprocess.run(["python", "monitoring.py"])

        # Start the timer for 10 seconds

        self.timer = QtCore.QTimer(self)

        self.timer.timeout.connect(self.update\_timer)

        self.timer\_interval = 1000  # 1 second

        self.timer\_count = 5  # Timer count set to 10 seconds

        self.timer.start(self.timer\_interval)

        # Button for showing result

        self.result\_button = QtWidgets.QPushButton("Result", self)

        self.result\_button.setStyleSheet("""

            QPushButton {

                background-color: #BDC9FF;

                color: blue;

                border-radius: 30px;

                padding: 10px 20px;

                border: none;

                font-weight: bold;

                font-size: 14px;

            }

        """)

        self.result\_button.clicked.connect(self.show\_result)

        layout.addWidget(self.result\_button)

        # Hide the result button initially

        self.result\_button.hide()

    def update\_timer(self):

        self.timer\_count -= 1

        self.timer\_label.setText(f"Time remaining: {self.timer\_count} seconds")

        if self.timer\_count <= 0:

            self.timer.stop()

            self.loading\_label.setText("Capturing Complete!")

            self.result\_button.show()

    def show\_result(self):

        # Open result.py script using subprocess

        subprocess.run(["python", "result.py"]) #update the path

    def mousePressEvent(self, event):

        if event.button() == QtCore.Qt.LeftButton:

            QtWidgets.QApplication.setOverrideCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))

    def mouseReleaseEvent(self, event):

        QtWidgets.QApplication.restoreOverrideCursor()

if \_\_name\_\_ == "\_\_main\_\_":

    import sys

    app = QtWidgets.QApplication(sys.argv)

    loading\_page = LoadingPage()

    loading\_page.show()

    sys.exit(app.exec\_())

**monitoring.py Code:**

import pyshark

import TSharkFeatureExtractionVer2

import os

# This function captures the traffic for 10 seconds from the WI-FI interface on the user's device into a PCAP file

def monitoring():

    capture = pyshark.LiveCapture(interface='Ethernet', output\_file="output.pcap")

    capture.sniff(timeout=10)

    # Save the PCAP file directory into the selecte\_file.txt file to be processed later

    current\_directory = os.path.dirname(os.path.abspath(\_\_file\_\_))

    with open("selected\_file.txt", "w") as file:

        # Write the file path to the text file

        file.write(current\_directory + "/output.pcap")

    return 'done'

def monitoring\_now():

    monitoring()

    # call the feature extraction function to convert the PCAP to CSV for prediction

    TSharkFeatureExtractionVer2.main()

monitoring\_now()

1. **Feature Extraction:**

* Script: ***TSharkFeatureExtractionVer2.py***
* Process: extract relevant features from the PCAP file and convert it to CSV format

**TSharkFeatureExtractionVer2.py Code:**

import os

import subprocess

import csv

import sys

# This function extracts the necessary features from the uploaded or captured PCAP file

def extract\_details\_from\_pcap(pcap\_file):

    # Specify the full path to the tshark executable

    tshark\_path = r'C:\Program Files\Wireshark\tshark.exe'

    tshark\_command = [tshark\_path, '-r', pcap\_file, '-t', 'e', '-T', 'fields', '-E', 'separator=,',

        '-e', 'frame.number', '-e', 'frame.len', '-e', 'frame.time', '-e', 'frame.time\_epoch', '-e', 'frame.protocols', '-e', 'eth.src', '-e', 'eth.dst', '-e', 'eth.type',

        '-e', 'ip.src', '-e', 'ip.dst', '-e', 'ip.len', '-e', 'ip.ttl', '-e', 'ip.flags', '-e', 'ip.frag\_offset', '-e', 'ip.proto', '-e', 'ip.version', '-e', 'ip.dsfield',

        '-e', 'ip.checksum', '-e', 'tcp.srcport', '-e', 'tcp.dstport', '-e', 'tcp.len', '-e', 'tcp.seq', '-e', 'tcp.ack', '-e', 'tcp.flags', '-e', 'tcp.flags.syn', '-e', 'tcp.flags.ack',

        '-e', 'tcp.flags.fin', '-e', 'tcp.flags.reset', '-e', 'tcp.window\_size', '-e', 'tcp.checksum', '-e', 'tcp.stream', '-e', 'udp.srcport', '-e', 'udp.dstport',  '-e', 'udp.length',

        '-e', 'udp.checksum', '-e', 'icmp.type', '-e', 'icmp.code', '-e', 'icmp.checksum', '-e', 'http.request.method', '-e', 'http.request.uri', '-e', 'http.request.version', '-e', 'http.request.full\_uri',

        '-e', 'http.response.code', '-e', 'http.user\_agent', '-e', 'http.content\_length\_header', '-e', 'http.content\_type', '-e', 'http.cookie', '-e', 'http.host', '-e', 'http.referer', '-e', 'http.location',

        '-e', 'http.authorization', '-e', 'http.connection', '-e', 'dns.qry.name', '-e', 'dns.qry.type', '-e', 'dns.qry.class', '-e', 'dns.flags.response', '-e', 'dns.flags.recdesired', '-e', 'dns.flags.rcode',

        '-e', 'dns.resp.ttl', '-e', 'dns.resp.len', '-e', 'smtp.req.command', '-e', 'smtp.data.fragment', '-e', 'pop.request.command', '-e', 'pop.response', '-e', 'imap.request.command', '-e', 'imap.response',

        '-e', 'ftp.request.command', '-e', 'ftp.request.arg', '-e', 'ftp.response.code']

    # Execute the tshark command

    result = subprocess.run(tshark\_command, stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)

    # Check for errors in command execution

    if result.returncode != 0:

        print(f"Error executing tshark command: {result.stderr}")

        return []

    return result.stdout.splitlines()

# This function writes the output file after extraction into a CSV file

def write\_to\_csv(output\_file, data):

    with open(output\_file, 'w', newline='') as csvfile:

        csv\_writer = csv.writer(csvfile)

        csv\_writer.writerow(['frame.number', 'frame.len', 'frame.time', 'frame.time\_epoch', 'frame.protocols',

                             'eth.src', 'eth.dst', 'eth.type', 'ip.src', 'ip.dst', 'ip.len', 'ip.ttl',

                             'ip.flags', 'ip.frag\_offset', 'ip.proto', 'ip.version', 'ip.dsfield', 'ip.checksum',

                             'tcp.srcport', 'tcp.dstport', 'tcp.len', 'tcp.seq', 'tcp.ack', 'tcp.flags',

                             'tcp.flags.syn', 'tcp.flags.ack', 'tcp.flags.fin', 'tcp.flags.reset', 'tcp.window\_size',

                             'tcp.checksum', 'tcp.stream', 'udp.srcport', 'udp.dstport', 'udp.length',

                             'udp.checksum', 'icmp.type', 'icmp.code', 'icmp.checksum', 'http.request.method',

                             'http.request.uri', 'http.request.version', 'http.request.full\_uri', 'http.response.code',

                             'http.user\_agent', 'http.content\_length\_header', 'http.content\_type', 'http.cookie',

                             'http.host', 'http.referer', 'http.location', 'http.authorization', 'http.connection',

                             'dns.qry.name', 'dns.qry.type', 'dns.qry.class', 'dns.flags.response',

                             'dns.flags.recdesired', 'dns.flags.rcode', 'dns.resp.ttl', 'dns.resp.len',

                             'smtp.req.command', 'smtp.data.fragment', 'pop.request.command', 'pop.response',

                             'imap.request.command', 'imap.response', 'ftp.request.command', 'ftp.request.arg',

                             'ftp.response.code', 'ftp.response.arg'])

        csv\_writer.writerows(data)

# This functio processes the directory of the PCAP file and makes sure to write the new CSV file into the same directory as the PCAP file

def process\_directory(directory\_path):

    # List to store details from each file

    all\_details = []

    total\_packets = 0

    # Correcting the split logic

    import re

    directory\_path = re.split(r'[\\/]', directory\_path)

    file = directory\_path[-1]

    directory\_path = directory\_path[:-1]

    dp = ''

    for x in directory\_path:

        dp += x + '/'

    # Check if the directory exists

    if not os.path.isdir(dp):

        print(f"Error: Directory '{dp}' does not exist.")

        return

    # Iterate through files in the directory

    for filename in os.listdir(dp):

        if filename == file:

            pcap\_file = os.path.join(dp, filename)

            print(f"Processing {pcap\_file}")

            # Extract details

            details = extract\_details\_from\_pcap(pcap\_file)

            packet\_count = len(details)

            total\_packets += packet\_count

            # Print the number of packets extracted

            print(f"Number of packets extracted from {filename}: {packet\_count}")

            # Write details to a CSV file

            csv\_file = 'output.csv'

            write\_to\_csv(csv\_file, [line.split(',') for line in details])

            # Append details to the list

            all\_details.extend(details)

    # Print the total number of packets across all files

    print(f"Total number of packets extracted across all files: {total\_packets}")

    return

def main():

    # Read the pcap file from the txt file that contains the file directory

    with open("selected\_file.txt", "r") as file:

        file\_path = file.read().strip()

    if len(sys.argv) != 2:

        directory\_path = file\_path

    else:

        directory\_path = sys.argv[1]

    process\_directory(directory\_path)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

1. **New data preprocessing and prediction:**

* Script: ***newdata\_clean\_predict.py***
* Process: apply preprocessing on the collected data and run it through the pretrained ML model for prediction.

**newdata\_clean\_predict.py Code:**

import pandas as pd

import pickle

import joblib

import matplotlib.pyplot as plt

# This function puts the new data through the neccessary preprocesing such as imputers and scalers

def feature\_selection\_cleaning(df):

    DataFrame = df.copy()

    df\_num = DataFrame[['udp.srcport','frame.len','ip.ttl','ip.dsfield','tcp.dstport']].astype('float64')

    df\_cat = DataFrame[['frame.time\_epoch','frame.time','ip.frag\_offset','ip.src','ip.version',

                       'ip.checksum','tcp.flags.reset','tcp.flags.syn','tcp.window\_size',

                       'tcp.flags.fin','ip.proto','eth.dst','frame.protocols','ip.len','tcp.seq']].astype("object")

    # Apply the numerical imputer

    with open('imputer\_num.pickle', 'rb') as file:

        loaded\_model = pickle.load(file)

    x = loaded\_model.transform(df\_num)

    df\_num = pd.DataFrame(x, columns=df\_num.columns)

    # Apply the scaler

    with open('stsc.pickle', 'rb') as file:

        loaded\_model = pickle.load(file)

    x = loaded\_model.transform(df\_num)

    df\_num = pd.DataFrame(x, columns=df\_num.columns)

    # Apply the categorical imputer

    with open('imputer\_cat.pickle', 'rb') as file:

        loaded\_model = pickle.load(file)

    x = loaded\_model.transform(df\_cat)

    df\_cat = pd.DataFrame(x, columns=df\_cat.columns)

    # Aplly the ordinal encoder

    with open('cat\_enc.pickle', 'rb') as file:

        loaded\_model = pickle.load(file)

    x = loaded\_model.transform(df\_cat)

    df\_cat = pd.DataFrame(x, columns=df\_cat.columns)

    new\_data = pd.concat([df\_num, df\_cat], axis=1)

    new\_data.reset\_index(drop=True, inplace=True)

    return new\_data

# This function reads the CSV file and calls the preprocessing function, then counts the amount of anomalies and normal packets, and creats and saves the visual plot for the prediction

def generate\_plot\_and\_stats():

    df = pd.read\_csv('output.csv', on\_bad\_lines='skip', engine='python', encoding='ISO-8859-1')

    new\_data = feature\_selection\_cleaning(df)

    # Apply the trained isolation forest model to the new data and predict how anomalous the traffic is

    with open('isolation\_forest\_model.pkl', 'rb') as file:

        model = joblib.load(file)

    prediction = model.predict(new\_data)

    # Count the number of benign and malware instances

    benign\_count = (prediction == 1).sum()

    malware\_count = (prediction == -1).sum()

    total\_count = benign\_count + malware\_count

    normal\_percentage = (benign\_count / total\_count) \* 100

    malware\_percentage = (malware\_count / total\_count) \* 100

    # Generate the plot

    plt.figure(figsize=(6, 6))

    plt.bar(['Normal', 'Malware'], [benign\_count, malware\_count], color=['green', 'red'])

    plt.title('Number of Normal and Malware Instances')

    plt.xlabel('Class')

    plt.ylabel('Count')

    plt.savefig('PLOT.png')

    plt.close()

    return benign\_count, normal\_percentage, malware\_count, malware\_percentage

def main():

   generate\_plot\_and\_stats()

1. **Visualise results:**

* Script: ***result.py***
* Process: output a plot showing the ratio of normal traffic to anomaly traffic and calculate the danger level.

**result.py Code:**

import sys

from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QLabel, QFrame, QHBoxLayout, QSpacerItem, QSizePolicy

from PyQt5.QtGui import QPixmap, QFont

from PyQt5.QtCore import Qt, QTimer

import matplotlib.pyplot as plt

import pandas as pd

import joblib

import pickle

import subprocess

import os

class ResultPage(QWidget):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.initUI()

    def initUI(self):

        # Window settings

        self.setWindowTitle('Result Page')

        self.setGeometry(100, 100, 1600, 900)

        self.setStyleSheet("background-color: #1D0E25;")

        # Main layout

        main\_layout = QVBoxLayout()

        # Logo

        logo\_label = QLabel(self)

        pixmap = QPixmap('logo.png').scaled(408, 386, Qt.KeepAspectRatio)

        logo\_label.setPixmap(pixmap)

        main\_layout.addWidget(logo\_label, alignment=Qt.AlignLeft | Qt.AlignTop)

        # Container for plot and boxes

        container\_layout = QHBoxLayout()

        # Plot

        plot\_label = QLabel(self)

        plot\_pixmap = QPixmap('plot.png').scaled(1000, 700, Qt.KeepAspectRatio)

        plot\_label.setPixmap(plot\_pixmap)

        container\_layout.addWidget(plot\_label, alignment=Qt.AlignLeft)

        # Add a spacer item to control spacing between plot and boxes

        container\_layout.addItem(QSpacerItem(20, 700, QSizePolicy.Minimum, QSizePolicy.Expanding))

        # Boxes for counts and percentages

        boxes\_layout = QVBoxLayout()

        # Green Box - Normal Data

        normal\_data\_box = QFrame(self)

        normal\_data\_box.setFixedSize(574, 281)

        normal\_data\_box.setStyleSheet("background-color: #E0F2E4; border: 2px solid green;")

        normal\_data\_layout = QVBoxLayout(normal\_data\_box)

        normal\_label = QLabel("Normal:")

        normal\_label.setFont(QFont('Arial', 14))

        normal\_label.setStyleSheet("color: green;")

        self.normal\_count\_label = QLabel()

        self.normal\_count\_label.setFont(QFont('Arial', 12))

        self.normal\_count\_label.setStyleSheet("color: gray;")

        self.normal\_percentage\_label = QLabel()

        self.normal\_percentage\_label.setFont(QFont('Arial', 12))

        self.normal\_percentage\_label.setStyleSheet("color: green;")

        normal\_data\_layout.addWidget(normal\_label)

        normal\_data\_layout.addWidget(self.normal\_count\_label)

        normal\_data\_layout.addWidget(self.normal\_percentage\_label)

        normal\_data\_box.setLayout(normal\_data\_layout)

        # Red Box - Malware Data

        malware\_data\_box = QFrame(self)

        malware\_data\_box.setFixedSize(574, 281)

        malware\_data\_box.setStyleSheet("background-color: #F9EBED; border: 2px solid red;")

        malware\_data\_layout = QVBoxLayout(malware\_data\_box)

        malware\_label = QLabel("Malware:")

        malware\_label.setFont(QFont('Arial', 14))

        malware\_label.setStyleSheet("color: red;")

        self.malware\_count\_label = QLabel()

        self.malware\_count\_label.setFont(QFont('Arial', 12))

        self.malware\_count\_label.setStyleSheet("color: gray;")

        self.malware\_percentage\_label = QLabel()

        self.malware\_percentage\_label.setFont(QFont('Arial', 12))

        self.malware\_percentage\_label.setStyleSheet("color: red;")

        malware\_data\_layout.addWidget(malware\_label)

        malware\_data\_layout.addWidget(self.malware\_count\_label)

        malware\_data\_layout.addWidget(self.malware\_percentage\_label)

        malware\_data\_box.setLayout(malware\_data\_layout)

        boxes\_layout.addWidget(normal\_data\_box)

        boxes\_layout.addWidget(malware\_data\_box)

        container\_layout.addLayout(boxes\_layout)

        main\_layout.addLayout(container\_layout)

        self.setLayout(main\_layout)

        # Update counts and percentages

        self.update\_counts\_and\_percentages()

        # Start a QTimer to call the alert window after 30 seconds

        self.timer = QTimer(self)

        self.timer.setSingleShot(True)

        self.timer.timeout.connect(self.show\_alert)

        self.timer.start(30000)  # 30 seconds

    def update\_counts\_and\_percentages(self):

        print("Updating counts and percentages...")  # Debug statement

        df = pd.read\_csv('output.csv', on\_bad\_lines='skip', engine='python', encoding='ISO-8859-1')

        new\_data = self.feature\_selection\_cleaning(df)

        with open('isolation\_forest\_model.pkl', 'rb') as file:

            model = joblib.load(file)

        prediction = model.predict(new\_data)

        benign\_count = (prediction == 1).sum()

        malware\_count = (prediction == -1).sum()

        total\_count = benign\_count + malware\_count

        benign\_percentage = (benign\_count / total\_count) \* 100 if total\_count > 0 else 0

        malware\_percentage = (malware\_count / total\_count) \* 100 if total\_count > 0 else 0

        self.normal\_count\_label.setText(f"{benign\_count}")

        self.normal\_percentage\_label.setText(f"{benign\_percentage:.2f}%")

        self.malware\_count\_label.setText(f"{malware\_count}")

        self.malware\_percentage\_label.setText(f"{malware\_percentage:.2f}%")

        # Generate the plot and save it as plot.png

        plt.figure(figsize=(12, 7))

        plt.bar(['Benign', 'Malware'], [benign\_count, malware\_count], color=['blue', 'red'])

        plt.title('Number of Benign and Malware Instances')

        plt.xlabel('Class')

        plt.ylabel('Count')

        plt.savefig('plot.png')

        print("Counts and percentages updated.")  # Debug statement

    def feature\_selection\_cleaning(self, df):

        print("Performing feature selection and cleaning...")  # Debug statement

        DataFrame = df.copy()

        df\_num = DataFrame[['udp.srcport','frame.len','ip.ttl','ip.dsfield','tcp.dstport']].astype('float64')

        df\_cat = DataFrame[['frame.time\_epoch','frame.time','ip.frag\_offset','ip.src','ip.version',

                       'ip.checksum','tcp.flags.reset','tcp.flags.syn','tcp.window\_size',

                       'tcp.flags.fin','ip.proto','eth.dst','frame.protocols','ip.len','tcp.seq']].astype("object")

        with open('imputer\_num.pickle', 'rb') as file:

            loaded\_model = pickle.load(file)

        x = loaded\_model.transform(df\_num)

        df\_num = pd.DataFrame(x, columns=df\_num.columns)

        with open('stsc.pickle', 'rb') as file:

            loaded\_model = pickle.load(file)

        x = loaded\_model.transform(df\_num)

        df\_num = pd.DataFrame(x, columns=df\_num.columns)

        with open('imputer\_cat.pickle', 'rb') as file:

            loaded\_model = pickle.load(file)

        x = loaded\_model.transform(df\_cat)

        df\_cat = pd.DataFrame(x, columns=df\_cat.columns)

        with open('cat\_enc.pickle', 'rb') as file:   # Handle unknown categories using OrdinalEncoder

            loaded\_model = pickle.load(file)

        x = loaded\_model.transform(df\_cat)

        df\_cat = pd.DataFrame(x, columns=df\_cat.columns)

        new\_data = pd.concat([df\_num, df\_cat], axis=1)

        new\_data.reset\_index(drop=True, inplace=True)

        print("Feature selection and cleaning done.")  # Debug statement

        return new\_data

    def show\_alert(self):

        print("Calling alert.py...")  # Debug statement

        try:

            alert\_script\_path = "alert.py"

            if os.path.exists(alert\_script\_path):

                subprocess.run(["python", alert\_script\_path])

                print("alert.py executed.")  # Debug statement

            else:

                print(f"alert.py not found at path: {alert\_script\_path}")

        except Exception as e:

            print(f"Error executing alert.py: {e}")  # Debug statement

if \_\_name\_\_ == '\_\_main\_\_':

    app = QApplication(sys.argv)

    result\_page = ResultPage()

    result\_page.show()

    sys.exit(app.exec\_())

1. **APIs Written to Complete the Pipeline:**

* Data Ingestion API: Handles the uploading and processing of PCAP files.
* Data Retrieval API: Provides endpoints to fetch processed data and results.
* Prediction API: Allows for real-time predictions using the trained model.

## 6.3 Model Implementation

### 6.3.1 Machine learning model choice and rationale for choice:

In our project, we chose to use the Isolation Forest model. Isolation Forest is a non-supervised machine learning algorithm that detects anomalies using binary trees. This algorithm has a linear time complexity and a low memory usage which makes it convenient to use in our project. It also seemed to show the best performance results when compared to OCSVM and we were able to achieve amazing results using this algorithm.

### 6.3.2 Model development and serving including hyper parameter tuning:

First, we worked on the dataset, we performed feature selection and ended up with 20 features that had the highest correlation with the label. Then we performed preprocessing such as Simple Imputer to fill in missing values, Ordinal Encoder to convert relevant categorical data to integer values, and Standard Scaler to have the data relatively have the same range. All the pre-processing steps were saved in pickle files to apply later to new data instances. Then we performed train-test splitting using the stratify parameter to have an evenly distributed split. After experimenting with the number of anomalies that should be in the dataset we ended up with a training dataset that had 80000 instances and testing dataset contain 20000 instances.

Before the training process, we mapped the labels in the datasets to (benign = 1, malware = -1). Then we created the isolation forest classifier with the final hyperparameters (n\_estimators = 100, max\_samples = ‘auto’, contamination = 0.05, max\_features = 1.0). The hyperparameter “contamination” indicates the ratio of anomalous instances to the normal traffic in the dataset and in our case it was around 0.05, so we didn’t change it. We manually tried different numbers of estimators, but when we went lower (50) or higher (200) the precision and accuracy would go down.

Then we fit the model with the final hyperparameters and tested the system on the training set and the testing set, the results of which you can see earlier in the document.

The trained model is served via Flask APIs, allowing for real-time predictions. In the case of a new data point (new traffic), whether its uploaded by the user or captured from the network, the system will apply the same preprocessing steps to this data and predict the number of normal and anomalous packets, and output the results on the “Result” page.

## 6.4 Additional Implementation Details

### 6.4.1 Code Functions in the Appendix

Each script provided will be included in the appendix, and references to specific lines of code will be made in the main document as we describe our data pipeline and other sections. Below are summaries of the scripts.

* **TSharkFeatureExtractionVer2.py**

**Functionality:** Extracts features from network traffic using TShark.

**Key Functions:**

**`extract\_details\_from\_pcap(pcap\_file)`:** Extracts features

**`write\_to\_csv()`:** Writes the extracted features to a CSV file

* **uploadpage.py**

**Functionality:** Handles file upload functionality in the application.

**Key Functions:**

**`upload\_file()`:** Handles file upload

**`browse\_files()`:** User selects a file from their own computer

* **page1.py**

**Functionality**: Renders the first page of the application.

**Key Functions:**

**`open\_page2(self)`:** Opens the second page of the application

* **page2.py**

**Functionality:** Renders the second page of the application.

**Key Functions:**

**`open\_capture\_page(self)`:** Opens the capture page

`**open\_upload\_page(self)`:** Opens the upload page

* **capturepage.py**

**Functionality:** Manages data capture for analysis.

**Key Functions:**

**`show\_result(self)`:** Gets the result from the monitoring function

* **isolation\_forest.py**

**Functionality:** Implements the Isolation Forest algorithm for anomaly detection.

* **monitoring.py**

**Functionality:** Monitors network traffic for anomalies.

**Key Functions:**

**`monitoring()`:** Monitors traffic

* **newdata\_clean\_predict.py**

**Functionality:** Cleans new data and makes predictions.

**Key Functions:**

**`generate\_plot\_and\_stats()`:** Plots the results

**`feature\_selection\_cleaning()`:** Cleans and preprocesses a data frame

### 6.4.2 Discussion of Main/Critical Algorithms/Techniques/Architecture

**Feature Extraction with TShark:**

**Reason for Choosing:** TShark is a powerful tool for detailed network traffic analysis and feature extraction, essential for training accurate anomaly detection models.

**Main Idea:** After capturing network traffic, it extracts relevant features that are processed for machine learning models.

**Pseudo-Code:**

function extract\_details\_from\_pcap(file):

run tshark on file

extract relevant fields

process and format fields

return features

function write\_to\_csv(output\_file, data):

Open output\_file in write mode as csvfile:

Create a csv writer object csv\_writer

For each row in data:

Write the row to csvfile using csv\_writer

Close the output\_file

**Anomaly Detection with Isolation Forest:**

**Reason for Choosing:** Isolation Forest is effective for anomaly detection, especially in high-dimensional datasets.

**Main Idea:** Uses an ensemble of trees to isolate anomalies in the dataset.

**Files:** **`newdata\_clean\_predict.py**` and **`isolation\_forest.py`**

**Pseudo-Code (isolation\_forest.py):**

Load datasets.

Map labels: 'Benign' -> 1, 'Malware' -> -1.

Extract features and labels.

Scale features.

Train Isolation Forest.

Predict on training and testing data for evaluation.

Evaluate and print metrics (Accuracy, Precision, Recall, F1 Score, ROC AUC, FNR).

Save model.

Count and plot benign vs. malware instances.

**Pseudo-Code (newdata\_clean\_predict.py):**

Define `feature\_selection\_cleaning` function to clean and preprocess data:

Split data into numerical and categorical features.

Load and apply numerical imputer, scaler.

Load and apply categorical imputer, encoder.

Combine processed numerical and categorical data.

Define `generate\_plot\_and\_stats` function:

Load data from CSV.

Clean and preprocess data using `feature\_selection\_cleaning`.

Load trained Isolation Forest model.

Predict using the model.

Count and calculate percentages of normal and malware instances.

Generate and save a bar plot of the results.

Define `main` function:

Call `generate\_plot\_and\_stats`.

### 6.4.3 System Architecture

**Components:**

**Frontend:** User interface for uploading files and viewing results.

**Backend:** Handles data processing, feature extraction, and anomaly detection.

### 6.4.4 Feature Implementation Table

| **Feature** | **Description** | **Status** |
| --- | --- | --- |
| **Real-time Monitoring** | Monitors network traffic in real-time | Implemented |
| **File Upload** | Allows users to upload traffic files | Implemented |
| **Feature Extraction** | Extracts features from network traffic | Implemented |
| **Anomaly Detection** | Detects anomalies in network traffic | Implemented |
| **User Interface** | Web interface for interacting with the tool | Implemented |
| **Historical Data Analysis** | Analyses past data for patterns | Deferred |
| **Alert System** | Sends alerts based on detected anomalies | Implemented |

**Table 14: Feature Implementation Table**

# Chapter 7

# Testing

## 7.1 Testing Approach

1. **Evaluation measures**

**Our performance measurements for the training set were as follows:**

* Accuracy = 95.0%
* Precision = 100.0%
* Recall = 95.0%
* F1 Score = 97.4%

**The performance measurements for the testing set were as follows:**

* Accuracy = 93.6%
* Precision = 99.5%
* Recall = 94.0%
* F1 Score = 96.7%
* False Negative Rate (FNR) = 6.0%

For an unsupervised algorithm these results are really great and the precision being over 99% is what we were aiming for to reduce the amount of false alarms as much as possible. And recall being at 94% indicates that more than 94% of the anomalies, regardless of their cause, are being detected.

1. **Model testing approach**

The model testing approach involves the following steps:

* **Isolation Forest Implementation:** The Isolation Forest algorithm is implemented in ***isolation\_forest.py.*** This file includes functions for model training, prediction, and evaluation.
* **Data Preprocessing:** Data cleaning and preparation are handled in ***newdata\_clean\_predict.py*** and ***TSharkFeatureExtractionVer2.py***. These scripts ensure the data is in the correct format and relevant features are extracted before feeding it into the model.
* **Prediction and Monitoring:** The ***monitoring.py*** script handles the real-time prediction and monitoring of network traffic, leveraging the trained Isolation Forest model.
* **User Interface Integration**: The ***uploadpage.py*** and capturepage.py files provide the user interface for uploading new data and capturing network traffic for analysis.

1. **Data splitting technique used**

The data can be split into training and testing sets using a standard technique such as train-test split or cross-validation. Since anomalies are rare, to accurately assess the model's performance it's important to ensure that the testing set contains a representative sample of anomalies. This helps in accurately evaluating the model's performance:

* **Train-Test Split:** The data is divided into training and testing sets, with careful consideration to include enough anomalies in the test set.
* **Cross-Validation:** To enhance the robustness of the evaluation. This technique ensures that the model's performance is validated across different subsets of the data.

1. **Pipeline testing approach**

The pipeline, including data preprocessing, model training, and prediction, it can be tested through a combination of unit tests and integration tests:

* **Unit Tests:** Each component of the pipeline is tested individually to ensure it functions correctly. This includes testing data preprocessing scripts for accurate feature extraction and data cleaning, as well as validating the Isolation Forest model's training and prediction functions.
* **Integration Tests:** The complete pipeline is tested to ensure smooth data flow and interaction between components. This includes uploading new data, preprocessing it, feeding it into the model, and monitoring predictions.

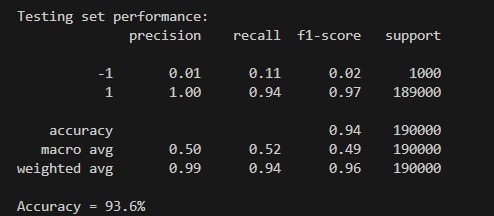
1. **Complete system testing approach**

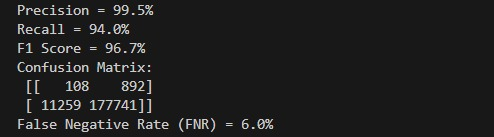
To make sure the system performs as planned, the entire system is put through an extensive testing process. This includes:

* **Simulating Network Traffic:** real data traffic with known anomalies is used to simulate network traffic. This helps in evaluating the system's ability to detect anomalies in a controlled environment.
* **Real-Time Monitoring:** The system's performance is monitored in real-time, ensuring that it accurately identifies anomalies and raising alerts as expected.
* **User Interface Testing:** The UI components are tested for functionality and user experience. This ensures that users can easily upload data and view results without encountering issues.

## 7.2 Testing Results

### 7.2.1 Testing Set Report





**Figure 7.2.1: Testing Set Report**

**1. Precision and Recall:**

The model demonstrates very high precision (99.5%) and high recall (94.0%), resulting in an excellent F1 score (96.7%). This indicates that the model is very good at identifying true positives with minimal false positives.

**2. Confusion Matrix:**

There are 108 true positives (correctly identified anomalies), 892 false positives (normal data misclassified as anomalies), 11259 false negatives (anomalies not detected), and 177741 true negatives (correctly identified normal data).

**3. False Negative Rate (FNR):**

The FNR is 6.0%, which is relatively low but indicates that there is still a small proportion of anomalies that the model fails to detect**.**

**4. Class Imbalance:**

The second image highlights a significant class imbalance, with only 1000 instances of class -1 (anomalies) compared to 189000 instances of class 1 (normal). Despite this imbalance, the model maintains a high recall and F1 score for class 1.

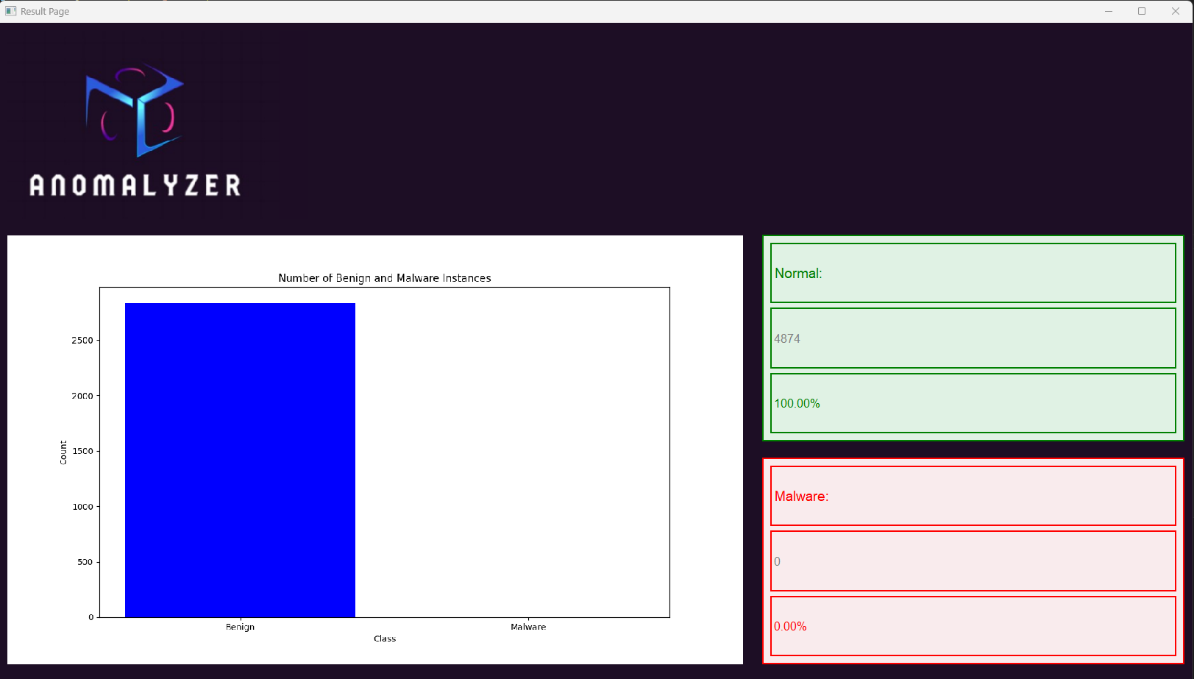
1. **Accuracy:**

The overall accuracy is 93.6%, which is very good given the class imbalance. However, the accuracy alone can be misleading due to the large number of normal instances.

1. **Macro vs. Weighted Average:**

The macro averages (precision, recall, F1 score) give equal weight to each class, highlighting the poor performance on the minority class (-1). In contrast, the weighted averages take into account the class imbalance, resulting in higher scores that reflect the model's overall performance better.

### 7.2.2 Capturing Output from Real Traffic



**Figure7.2.2: Capturing Output from Real Traffic**

The ANOMALYZER interface displays real traffic analysis results, indicating all detected instances are benign with a total count of 4874 and no malware detected, as shown in a bar chart and corresponding statistics panels. This result suggests an absence of malware in the analyzed traffic.

## 7.3 Discussion

We encountered challenges with the datasets during testing. Initially, we attempted to balance the dataset, but we realized that this approach was not ideal for anomaly detection. The Isolation Forest model is designed to detect outliers or rare instances, so balancing the dataset counteracted this purpose. Through continuous experimentation and adjustments, we ultimately created a dataset that yielded optimal results for our model.

# Chapter 8

# Conclusions and Future Work

## 8.1 Conclusions

Conclusively, the purpose of the project was to detect unlabelled normal and abnormal network traffic using machine learning. The primary achievements of this project are in the first stage of the project was to upload a dataset or monitor network traffic, the second stage was to pre-process and feature engineer the data, and the third stage was model development. In this phase various machine learning algorithms were evaluated, and the best-performing model that demonstrated high accuracy in detecting normal and abnormal behaviour in network traffic was the isolation forest model, the forth and the last step was to send the graph and alert to the user to visualise the traffic.

## 8.2 Future work

As for the future work, the upgrade that this design would face, we would enhance model training by incorporating more data and continuous learning, we would add advanced feature extraction like deep learning techniques for feature extraction, such as auto encoders or convolution neural networks, to capture more complex patterns in the data. By addressing these areas, we aim to enhance the efficiency and reliability of our network traffic monitoring system, ensuring it remains a robust tool for detecting network threats.

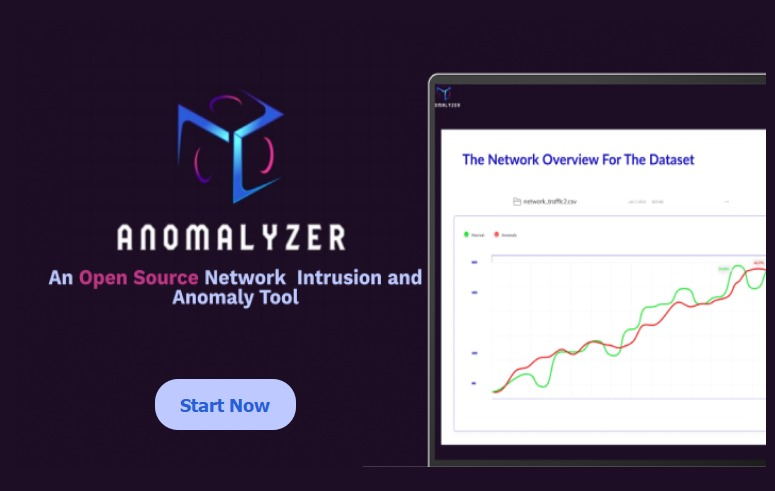
**References**

1. Chandola, V., Banerjee, A., & Kumar, V. (2022). Anomaly detection: A survey. *ACM Computing Surveys (CSUR),* 41(3), 1-58.
2. Zhang, L., & Liu, L. (2022). Data Anomaly Detection Based on Isolation Forest Algorithm*. International Conference on Computation, Big-Data and Engineering (ICCBE)*, 87-89.
3. Wang, X., Liu, J., & Zhang, Y. (2023). Deep learning for network anomaly detection: A survey. *IEEE Communications Surveys & Tutorials*, 25(1), 1-20.
4. Liu, F. T., Ting, K. M., & Zhou, Z. H. (2022). Isolation forest. In *Proceedings of the IEEE International Conference on Data Mining* (pp. 413-422). IEEE.
5. Oliveira, J. M., Almeida, J., Macedo, D., & Nogueira, J. M. (2023). Comparative Analysis of Unsupervised Machine Learning Algorithms for Anomaly Detection in Network Data. *IEEE Latin-American Conference on Communications (LATINCOM)*, 1-6.
6. Sahoo, S., Liu, C., & Hoi, S. C. H. (2022). Hybrid intrusion detection systems: A review of existing methods and challenges. *Journal of Network and Computer Applications,* 107, 102-123.
7. Marchal, S., Jiang, X., State, R., & Engel, T. (2023). A big data architecture for large scale security monitoring in IoT*. IEEE Internet of Things Journal*, 6(2), 2041-2050.
8. Kim, H., Kim, D., & Choi, J. (2022). Federated learning for anomaly detection in IoT networks. *Sensors,* 22(4), 1357.
9. Abuthawabeh, M. K., & Mahmoud, K. (2022). Android Malware Detection and Categorization Based on Conversation-level Network Traffic Features*. IEEE Access*, 10, 115354-115363.
10. Canadian Institute for Cybersecurity. (2022). CICAndMal2017 and CICAndMal2019 Datasets. Retrieved from [https://www.unb.ca/cic/datasets/andmal2017.html](https://www.unb.ca/cic/datasets/andmal2017.html).
11. Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys (CSUR),* 41(3), 1-58.
12. Sultana, N., Chilamkurti, N., & Peng, W. (2020). Survey on anomaly detection using data mining techniques. *Procedia Computer Science,* 144, 278-288.
13. Aggarwal, C. C. (2017). Outlier analysis. Springer.
14. Lakhina, A., Crovella, M., & Diot, C. (2004). Diagnosing network-wide traffic anomalies. In *ACM SIGCOMM Computer Communication Review* (Vol. 34, No. 4, pp. 219-230).
15. Sommer, R., & Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. *In 2010 IEEE Symposium on Security and Privacy (pp. 305-316).* IEEE.
16. Roesch, M. (1999). Snort - Lightweight Intrusion Detection for Networks. *In Proceedings of the 13th USENIX Conference on System Administration (LISA '99)* (pp. 229-238).
17. Hofstede, R., Celeda, P., Trammell, B., Drago, I., Sadre, R., Sperotto, A., & Pras, A. (2014). Flow monitoring explained: From packet capture to data analysis with NetFlow and IPFIX. *IEEE Communications Surveys & Tutorials, 16(4), 2037-2064.*

**Appendices**

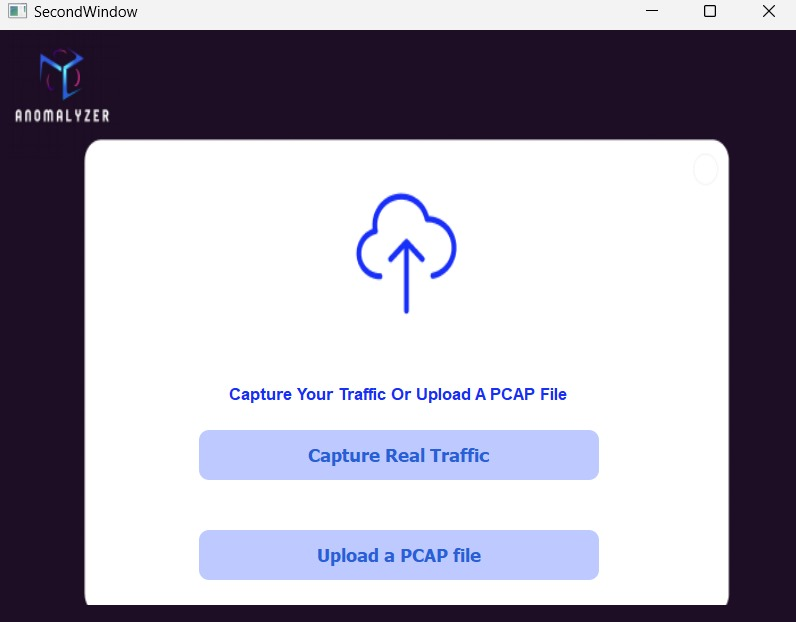
1. 1. **Users’ Manual**

**Start with running *Pag1.py* code:**

****

**Figure Appendix A.1: Start Page**

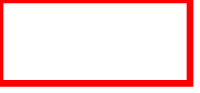
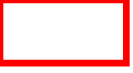
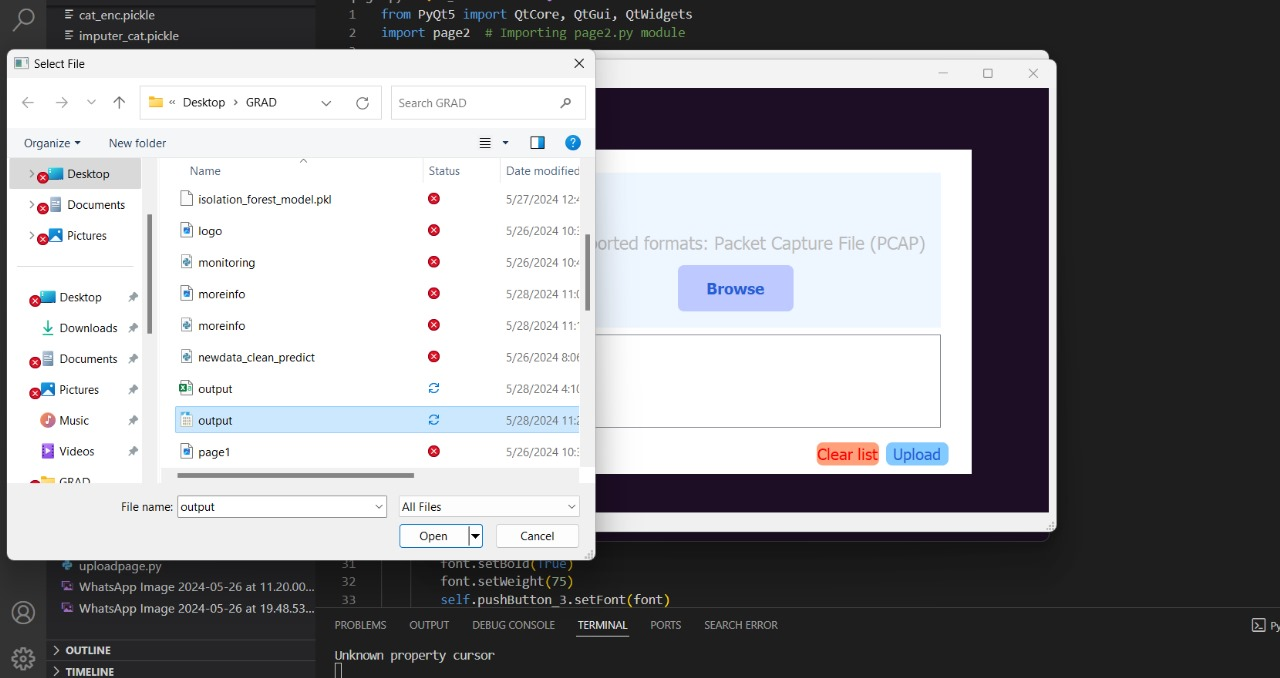
**The second page will appear:**



**Figure Appendix A.2: Uploading Page**

You can choose whether you want to capture a real traffilc capture or upload a pcap file

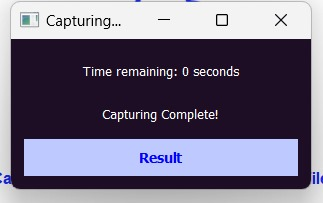
**Choosing upload a PCAP file:**



**Figure Appendix A.3: Upload the Dataset**

Amomalyzer tool allows users to upload datasets to their network, whether they are pcap files from Wireshark, and then the tool converts them to CSV, with a notification displayed when data sets or files in an unacceptable format are uploaded.

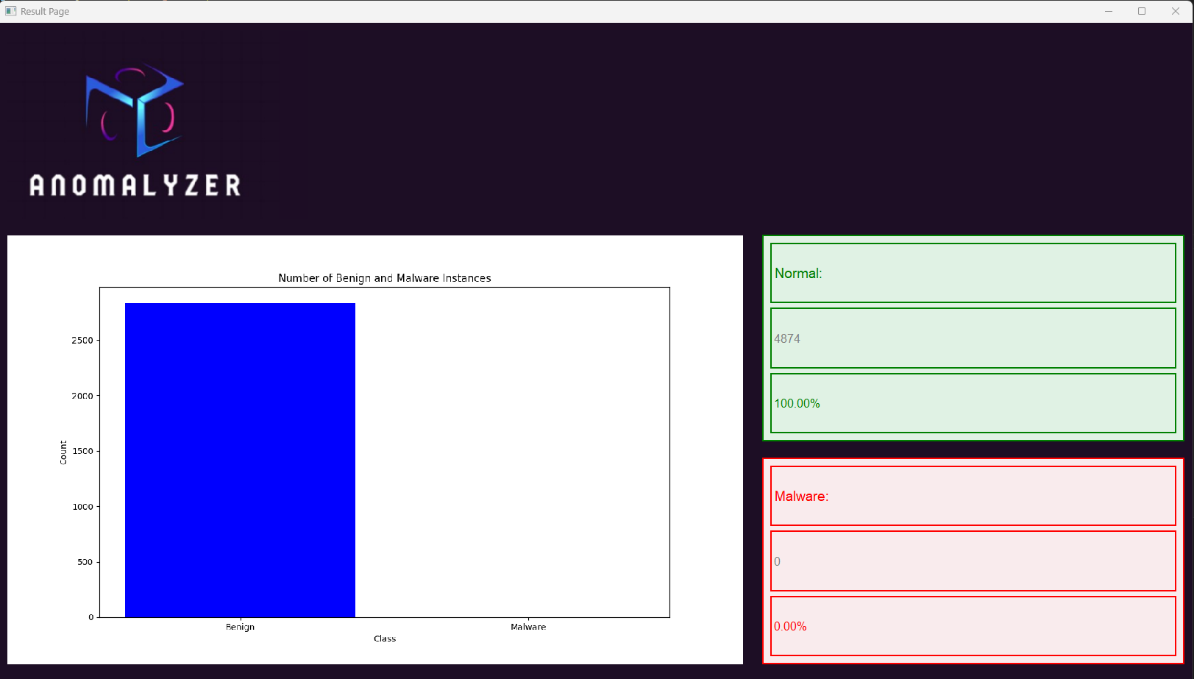
**Capture Real Traffic:**

****

**Figure Appendix A.4: Capture Real Traffic**

This pop-up window will appear and when the captcharing process is finished you can click on the “Result” button

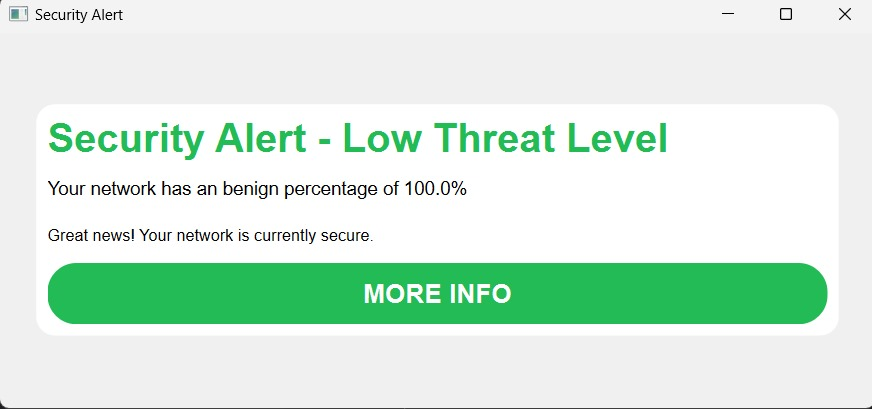
**Result Page:**

****

**Figure Appendix A.5: Result Page (Real Traffic Capturing)**

The result screen shows the file name, type, and size, and an easy graph showing whether the data is normal (green) or anomaly data (red), along with the number of normal and anomaly data and their percentage of the original amount of data.

### Alert Pop-up window:



**Figure Appendix A.6: Alert Pop-Up Window**

Clarifying the form of the notification that explains the Threat level of the data and recommendations for dealing with it.

You can click on More Info to understand the meaning of the poem and the reason behind it.

**Alert Meaning:**



**Figure Appendix A.7: Alert Meaning**

When you click on "MORE INFO" bottom , this screen will appear , the results is based on the percentage of anomaly (abnormal) data that appeared, this screen explains the meaning of low, moderate, and high levels of the threat on the network, with recommendations for dealing with each of them.

* 1. **ImpelemtnaioVideo https://youtu.be/n3O0\_DQYn9U?si=Xt9afxxz3kKM5lK2**

1. 1. **Document Changes**

**General differences:**

Our general idea differs a little bit from project 1. In this project we decided to create our own tool and capture and work with our own network traffic using T-shark, in comparison to project 1 where we wanted to rely exclusively on snort and improve its functionality. We also settled on a different ML algorithm since we researched more and got better results from the Isolation forest algorithm, in comparison to the OCSVM that we thought we'd use in project 1.

In the light of these major differences, here are the specific changes we made to the document.

**Title:** the new project title is more relevant to the changes that we had to our idea.

* 1. **Overview:**

The new overview is more cohesive and does a better job at describing our project and what we aim to achieve.

* 1. **Problem statement:**

The new problem statement makes our project more general and inclusive whereas the old one heavily focused on Snort's performance.

**3.1 related work:**

Added more related works that are relevant to out project scope with better description for better understanding of the research and thought processes.

**3.2 knowledge gap:**

Not focusing on the functionality or performance of snort anymore made the knowledge gap slightly more general, focusing more on anomaly detection using exclusively machine learning.

**4.5 other requirements:**

Creating a new tool from zero requires more maintenance and supervision, hence we added more requirements.

**5.1 architectural design:**

The process of implementation made us more familiar with the system structure and the tools necessary for actual implementation, therefore the diagrams are now more accurate and relevant to the project.

1. 1. **Code Documentation**

| Page1.py |
| --- |
| from PyQt5 import QtCore, QtGui, QtWidgets  import page2 # Importing page2.py module  class Ui\_MainWindow(object):  def setupUi(self, MainWindow):  MainWindow.setObjectName("MainWindow")  MainWindow.resize(785, 493)  MainWindow.setStyleSheet("background-color: rgb(29, 14, 37);")  self.centralwidget = QtWidgets.QWidget(MainWindow)  self.centralwidget.setObjectName("centralwidget")  self.background = QtWidgets.QFrame(self.centralwidget)  self.background.setGeometry(QtCore.QRect(0, 0, 941, 561))  self.background.setCursor(QtGui.QCursor(QtCore.Qt.ArrowCursor))  self.background.setToolTipDuration(2)  self.background.setAutoFillBackground(False)  self.background.setFrameShape(QtWidgets.QFrame.StyledPanel)  self.background.setFrameShadow(QtWidgets.QFrame.Raised)  self.background.setObjectName("background")  self.logoandoutput = QtWidgets.QWidget(self.background)  self.logoandoutput.setGeometry(QtCore.QRect(20, 20, 751, 471))  self.logoandoutput.setObjectName("logoandoutput")  self.background\_label = QtWidgets.QLabel(self.logoandoutput)  self.background\_label.setGeometry(QtCore.QRect(0, 0, 751, 471))  self.background\_label.setPixmap(QtGui.QPixmap('page1.png'))  self.background\_label.setScaledContents(True)  self.background\_label.setObjectName("background\_label")  self.pushButton\_3 = QtWidgets.QPushButton(self.logoandoutput)  self.pushButton\_3.setGeometry(QtCore.QRect(140, 360, 141, 51))  font = QtGui.QFont()  font.setPointSize(11)  font.setBold(True)  font.setWeight(75)  self.pushButton\_3.setFont(font)  self.pushButton\_3.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))  self.pushButton\_3.setStyleSheet("""  QPushButton {  background-color: #BDC9FF;  color: #285FD7;  border-radius: 23px;  padding: 10px 20px;  }  """)  self.pushButton\_3.setObjectName("pushButton\_3")  self.pushButton\_3.clicked.connect(self.open\_page2) # Connect clicked signal to open\_page2 method  self.pushButton\_3.raise\_()  MainWindow.setCentralWidget(self.centralwidget)  self.statusbar = QtWidgets.QStatusBar(MainWindow)  self.statusbar.setObjectName("statusbar")  MainWindow.setStatusBar(self.statusbar)  self.retranslateUi(MainWindow)  QtCore.QMetaObject.connectSlotsByName(MainWindow)  def retranslateUi(self, MainWindow):  \_translate = QtCore.QCoreApplication.translate  MainWindow.setWindowTitle(\_translate("MainWindow", "MainWindow"))  self.pushButton\_3.setText(\_translate("MainWindow", "Start Now"))  def open\_page2(self):  self.window = QtWidgets.QMainWindow()  self.ui = page2.Ui\_SecondWindow() # Initialize the UI from page2.py  self.ui.setupUi(self.window)  self.window.show()  if \_\_name\_\_ == "\_\_main\_\_":  import sys  app = QtWidgets.QApplication(sys.argv)  MainWindow = QtWidgets.QMainWindow()  ui = Ui\_MainWindow()  ui.setupUi(MainWindow)  MainWindow.show()  sys.exit(app.exec\_()) |

| Page2.py |
| --- |
| from PyQt5 import QtCore, QtGui, QtWidgets  import capturepage # Importing the capturepage module  import uploadpage  class Ui\_SecondWindow(object):  def setupUi(self, SecondWindow):  SecondWindow.setObjectName("SecondWindow")  SecondWindow.resize(800, 600)  SecondWindow.setStyleSheet("background-color: rgb(29, 14, 37);")  self.centralwidget = QtWidgets.QWidget(SecondWindow)  self.centralwidget.setObjectName("centralwidget")  # Background frame  self.background = QtWidgets.QFrame(self.centralwidget)  self.background.setGeometry(QtCore.QRect(0, 0, 800, 600))  self.background.setCursor(QtGui.QCursor(QtCore.Qt.ArrowCursor))  self.background.setToolTipDuration(2)  self.background.setAutoFillBackground(False)  self.background.setFrameShape(QtWidgets.QFrame.StyledPanel)  self.background.setFrameShadow(QtWidgets.QFrame.Raised)  self.background.setObjectName("background")  # Background image  self.background\_label = QtWidgets.QLabel(self.background)  self.background\_label.setGeometry(QtCore.QRect(0, 0, 800, 600))  self.background\_label.setPixmap(QtGui.QPixmap('page2.png'))  self.background\_label.setScaledContents(True)  self.background\_label.setObjectName("background\_label")  # Text box above buttons  self.textEdit = QtWidgets.QTextEdit(self.background)  self.textEdit.setGeometry(QtCore.QRect(200, 350, 400, 30)) # Adjusted position and size as needed  self.textEdit.setStyleSheet("background-color: white; border: 1px solid white; color: #182EF9;")  self.textEdit.setObjectName("textEdit")  self.textEdit.setReadOnly(True)  self.textEdit.setAlignment(QtCore.Qt.AlignCenter)  self.textEdit.setFont(QtGui.QFont("Arial", 10, QtGui.QFont.Bold))  self.textEdit.setText(" Capture Your Traffic Or Upload A PCAP File")  # Capture button  self.captureButton = QtWidgets.QPushButton(self.background)  self.captureButton.setGeometry(QtCore.QRect(200, 400, 400, 50)) # Adjusted y position to 400  font = QtGui.QFont()  font.setPointSize(11)  font.setBold(True)  font.setWeight(75)  self.captureButton.setFont(font)  self.captureButton.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))  self.captureButton.setStyleSheet("""  QPushButton {  background-color: #BDC9FF;  color: #285FD7;  border-radius: 10px;  padding: 10px 20px;  }  """)  self.captureButton.setObjectName("capture")  self.captureButton.setText("Capture Real Traffic")    # Connect capture button to open capturepage.py  self.captureButton.clicked.connect(self.open\_capture\_page)    # Upload button  self.uploadButton = QtWidgets.QPushButton(self.background)  self.uploadButton.setGeometry(QtCore.QRect(200, 500, 400, 50)) # Adjusted y position to 500  self.uploadButton.setFont(font)  self.uploadButton.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))  self.uploadButton.setStyleSheet("""  QPushButton {  background-color: #BDC9FF;  color: #285FD7;  border-radius: 10px;  padding: 10px 20px;  }  """)  self.uploadButton.setObjectName("pcapcsv")  self.uploadButton.setText("Upload a PCAP file")    # Connect upload button to open uploadpage.py  self.uploadButton.clicked.connect(self.open\_upload\_page)  SecondWindow.setCentralWidget(self.centralwidget)  self.statusbar = QtWidgets.QStatusBar(SecondWindow)  self.statusbar.setObjectName("statusbar")  SecondWindow.setStatusBar(self.statusbar)  self.retranslateUi(SecondWindow)  QtCore.QMetaObject.connectSlotsByName(SecondWindow)  def retranslateUi(self, SecondWindow):  \_translate = QtCore.QCoreApplication.translate  SecondWindow.setWindowTitle(\_translate("SecondWindow", "SecondWindow"))  def open\_capture\_page(self):  self.window = QtWidgets.QMainWindow()  self.ui = capturepage.LoadingPage() # Changed to LoadingPage as per capturepage.py  self.ui.show() # Changed to show() method  def open\_upload\_page(self):  self.window = QtWidgets.QMainWindow()  self.ui = uploadpage.Ui\_MainWindow() # Changed to Ui\_MainWindow as per uploadpage.py  self.ui.setupUi(self.window) # Initialize the upload page UI  self.window.show()  if \_\_name\_\_ == "\_\_main\_\_":  import sys  app = QtWidgets.QApplication(sys.argv)  SecondWindow = QtWidgets.QMainWindow()  ui = Ui\_SecondWindow()  ui.setupUi(SecondWindow)  SecondWindow.show()  sys.exit(app.exec\_()) |

| uploadpage.py |
| --- |
| from PyQt5 import QtCore, QtGui, QtWidgets  import capturepage # Importing the capturepage module  import uploadpage  class Ui\_SecondWindow(object):  def setupUi(self, SecondWindow):  SecondWindow.setObjectName("SecondWindow")  SecondWindow.resize(800, 600)  SecondWindow.setStyleSheet("background-color: rgb(29, 14, 37);")  self.centralwidget = QtWidgets.QWidget(SecondWindow)  self.centralwidget.setObjectName("centralwidget")  # Background frame  self.background = QtWidgets.QFrame(self.centralwidget)  self.background.setGeometry(QtCore.QRect(0, 0, 800, 600))  self.background.setCursor(QtGui.QCursor(QtCore.Qt.ArrowCursor))  self.background.setToolTipDuration(2)  self.background.setAutoFillBackground(False)  self.background.setFrameShape(QtWidgets.QFrame.StyledPanel)  self.background.setFrameShadow(QtWidgets.QFrame.Raised)  self.background.setObjectName("background")  # Background image  self.background\_label = QtWidgets.QLabel(self.background)  self.background\_label.setGeometry(QtCore.QRect(0, 0, 800, 600))  self.background\_label.setPixmap(QtGui.QPixmap('page2.png'))  self.background\_label.setScaledContents(True)  self.background\_label.setObjectName("background\_label")  # Text box above buttons  self.textEdit = QtWidgets.QTextEdit(self.background)  self.textEdit.setGeometry(QtCore.QRect(200, 350, 400, 30)) # Adjusted position and size as needed  self.textEdit.setStyleSheet("background-color: white; border: 1px solid white; color: #182EF9;")  self.textEdit.setObjectName("textEdit")  self.textEdit.setReadOnly(True)  self.textEdit.setAlignment(QtCore.Qt.AlignCenter)  self.textEdit.setFont(QtGui.QFont("Arial", 10, QtGui.QFont.Bold))  self.textEdit.setText(" Capture Your Traffic Or Upload A PCAP File")  # Capture button  self.captureButton = QtWidgets.QPushButton(self.background)  self.captureButton.setGeometry(QtCore.QRect(200, 400, 400, 50)) # Adjusted y position to 400  font = QtGui.QFont()  font.setPointSize(11)  font.setBold(True)  font.setWeight(75)  self.captureButton.setFont(font)  self.captureButton.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))  self.captureButton.setStyleSheet("""  QPushButton {  background-color: #BDC9FF;  color: #285FD7;  border-radius: 10px;  padding: 10px 20px;  }  """)  self.captureButton.setObjectName("capture")  self.captureButton.setText("Capture Real Traffic")    # Connect capture button to open capturepage.py  self.captureButton.clicked.connect(self.open\_capture\_page)    # Upload button  self.uploadButton = QtWidgets.QPushButton(self.background)  self.uploadButton.setGeometry(QtCore.QRect(200, 500, 400, 50)) # Adjusted y position to 500  self.uploadButton.setFont(font)  self.uploadButton.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))  self.uploadButton.setStyleSheet("""  QPushButton {  background-color: #BDC9FF;  color: #285FD7;  border-radius: 10px;  padding: 10px 20px;  }  """)  self.uploadButton.setObjectName("pcapcsv")  self.uploadButton.setText("Upload a PCAP file")    # Connect upload button to open uploadpage.py  self.uploadButton.clicked.connect(self.open\_upload\_page)  SecondWindow.setCentralWidget(self.centralwidget)  self.statusbar = QtWidgets.QStatusBar(SecondWindow)  self.statusbar.setObjectName("statusbar")  SecondWindow.setStatusBar(self.statusbar)  self.retranslateUi(SecondWindow)  QtCore.QMetaObject.connectSlotsByName(SecondWindow)  def retranslateUi(self, SecondWindow):  \_translate = QtCore.QCoreApplication.translate  SecondWindow.setWindowTitle(\_translate("SecondWindow", "SecondWindow"))  def open\_capture\_page(self):  self.window = QtWidgets.QMainWindow()  self.ui = capturepage.LoadingPage() # Changed to LoadingPage as per capturepage.py  self.ui.show() # Changed to show() method  def open\_upload\_page(self):  self.window = QtWidgets.QMainWindow()  self.ui = uploadpage.Ui\_MainWindow() # Changed to Ui\_MainWindow as per uploadpage.py  self.ui.setupUi(self.window) # Initialize the upload page UI  self.window.show()  if \_\_name\_\_ == "\_\_main\_\_":  import sys  app = QtWidgets.QApplication(sys.argv)  SecondWindow = QtWidgets.QMainWindow()  ui = Ui\_SecondWindow()  ui.setupUi(SecondWindow)  SecondWindow.show()  sys.exit(app.exec\_()) |

| capturepage.py |
| --- |
| from PyQt5 import QtCore, QtGui, QtWidgets  import capturepage # Importing the capturepage module  import uploadpage  class Ui\_SecondWindow(object):  def setupUi(self, SecondWindow):  SecondWindow.setObjectName("SecondWindow")  SecondWindow.resize(800, 600)  SecondWindow.setStyleSheet("background-color: rgb(29, 14, 37);")  self.centralwidget = QtWidgets.QWidget(SecondWindow)  self.centralwidget.setObjectName("centralwidget")  # Background frame  self.background = QtWidgets.QFrame(self.centralwidget)  self.background.setGeometry(QtCore.QRect(0, 0, 800, 600))  self.background.setCursor(QtGui.QCursor(QtCore.Qt.ArrowCursor))  self.background.setToolTipDuration(2)  self.background.setAutoFillBackground(False)  self.background.setFrameShape(QtWidgets.QFrame.StyledPanel)  self.background.setFrameShadow(QtWidgets.QFrame.Raised)  self.background.setObjectName("background")  # Background image  self.background\_label = QtWidgets.QLabel(self.background)  self.background\_label.setGeometry(QtCore.QRect(0, 0, 800, 600))  self.background\_label.setPixmap(QtGui.QPixmap('page2.png'))  self.background\_label.setScaledContents(True)  self.background\_label.setObjectName("background\_label")  # Text box above buttons  self.textEdit = QtWidgets.QTextEdit(self.background)  self.textEdit.setGeometry(QtCore.QRect(200, 350, 400, 30)) # Adjusted position and size as needed  self.textEdit.setStyleSheet("background-color: white; border: 1px solid white; color: #182EF9;")  self.textEdit.setObjectName("textEdit")  self.textEdit.setReadOnly(True)  self.textEdit.setAlignment(QtCore.Qt.AlignCenter)  self.textEdit.setFont(QtGui.QFont("Arial", 10, QtGui.QFont.Bold))  self.textEdit.setText(" Capture Your Traffic Or Upload A PCAP File")  # Capture button  self.captureButton = QtWidgets.QPushButton(self.background)  self.captureButton.setGeometry(QtCore.QRect(200, 400, 400, 50)) # Adjusted y position to 400  font = QtGui.QFont()  font.setPointSize(11)  font.setBold(True)  font.setWeight(75)  self.captureButton.setFont(font)  self.captureButton.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))  self.captureButton.setStyleSheet("""  QPushButton {  background-color: #BDC9FF;  color: #285FD7;  border-radius: 10px;  padding: 10px 20px;  }  """)  self.captureButton.setObjectName("capture")  self.captureButton.setText("Capture Real Traffic")    # Connect capture button to open capturepage.py  self.captureButton.clicked.connect(self.open\_capture\_page)    # Upload button  self.uploadButton = QtWidgets.QPushButton(self.background)  self.uploadButton.setGeometry(QtCore.QRect(200, 500, 400, 50)) # Adjusted y position to 500  self.uploadButton.setFont(font)  self.uploadButton.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))  self.uploadButton.setStyleSheet("""  QPushButton {  background-color: #BDC9FF;  color: #285FD7;  border-radius: 10px;  padding: 10px 20px;  }  """)  self.uploadButton.setObjectName("pcapcsv")  self.uploadButton.setText("Upload a PCAP file")    # Connect upload button to open uploadpage.py  self.uploadButton.clicked.connect(self.open\_upload\_page)  SecondWindow.setCentralWidget(self.centralwidget)  self.statusbar = QtWidgets.QStatusBar(SecondWindow)  self.statusbar.setObjectName("statusbar")  SecondWindow.setStatusBar(self.statusbar)  self.retranslateUi(SecondWindow)  QtCore.QMetaObject.connectSlotsByName(SecondWindow)  def retranslateUi(self, SecondWindow):  \_translate = QtCore.QCoreApplication.translate  SecondWindow.setWindowTitle(\_translate("SecondWindow", "SecondWindow"))  def open\_capture\_page(self):  self.window = QtWidgets.QMainWindow()  self.ui = capturepage.LoadingPage() # Changed to LoadingPage as per capturepage.py  self.ui.show() # Changed to show() method  def open\_upload\_page(self):  self.window = QtWidgets.QMainWindow()  self.ui = uploadpage.Ui\_MainWindow() # Changed to Ui\_MainWindow as per uploadpage.py  self.ui.setupUi(self.window) # Initialize the upload page UI  self.window.show()  if \_\_name\_\_ == "\_\_main\_\_":  import sys  app = QtWidgets.QApplication(sys.argv)  SecondWindow = QtWidgets.QMainWindow()  ui = Ui\_SecondWindow()  ui.setupUi(SecondWindow)  SecondWindow.show()  sys.exit(app.exec\_()) |

| monitoring.py |
| --- |
| import pyshark  import TSharkFeatureExtractionVer2  import os  # This function captures the traffic for 10 seconds from the WI-FI interface on the user's device into a PCAP file  def monitoring():  capture = pyshark.LiveCapture(interface='Ethernet', output\_file="output.pcap")  capture.sniff(timeout=10)  # Save the PCAP file directory into the selecte\_file.txt file to be processed later  current\_directory = os.path.dirname(os.path.abspath(\_\_file\_\_))  with open("selected\_file.txt", "w") as file:  # Write the file path to the text file  file.write(current\_directory + "/output.pcap")  return 'done'  def monitoring\_now():  monitoring()  # call the feature extraction function to convert the PCAP to CSV for prediction  TSharkFeatureExtractionVer2.main()  monitoring\_now() |

| Feature\_selection.py |
| --- |
| from sklearn.utils import shuffle  import pandas as pd  from sklearn.impute import SimpleImputer  from sklearn.preprocessing import StandardScaler  from sklearn.preprocessing import OrdinalEncoder  from sklearn.model\_selection import train\_test\_split  # Splitting the data into the train and test sets  def data\_splitting(DataFrame):  df\_train, df\_test = train\_test\_split(DataFrame, test\_size = 0.2, stratify = DataFrame['Label'], random\_state = 42)  return df\_train,df\_test  # Find the most correlated features  def most\_corr\_features(df):  print("Checking the correlation:")    encoding = OrdinalEncoder()  df['encoded\_labelll'] = encoding.fit\_transform(df[['Label']])  corr\_matrix = df.corr(numeric\_only=True)  corr\_feature = corr\_matrix['encoded\_labelll'].abs().sort\_values(ascending=False)    cols = []  for i in range(1, 21):  most\_correlated\_feature = corr\_feature.index[i]  cols.append(most\_correlated\_feature)  print("The most correlated feature: ", most\_correlated\_feature, '=', corr\_feature[i])    return cols  # Initial preprocessing for the feature selection process  def prep\_for\_feture\_selection(df):  df = df.copy()    # Drop unnamed features  df.drop(df.columns[df.columns.str.contains('unnamed', case=False)], axis=1, inplace=True)  Labels = df['Label'].copy() # separating labels from df for preprocessing  del df['Label']    # Remove columns with all null values  df = df.drop(['smtp.data.fragment', 'pop.request.command', 'pop.response', 'imap.request.command',  'imap.response', 'ftp.request.command', 'ftp.request.arg', 'ftp.response.code',  'ftp.response.arg'], axis=1)  # Separate categorical data from numerical data  df\_cat = df.select\_dtypes(include=['object'])  df\_num = df.select\_dtypes(exclude=['object'])  df\_num.reset\_index(drop=True, inplace=True)  df\_cat.reset\_index(drop=True, inplace=True)    # Use simple imputer to fill missing values in numerical data  imputer = SimpleImputer(strategy="mean")  X = imputer.fit\_transform(df\_num)  df\_num = pd.DataFrame(X, columns=df\_num.columns)  # Scale the numerical data  stsc = StandardScaler()  n = stsc.fit\_transform(df\_num)  df\_num = pd.DataFrame(n, columns=df\_num.columns)    df\_num.reset\_index(drop=True, inplace=True)  # Fill in the missing values in the categorical data  imputer = SimpleImputer(strategy="most\_frequent")  y = imputer.fit\_transform(df\_cat)  df\_cat = pd.DataFrame(y, columns=df\_cat.columns)  # Use ordinal encoder to change data types  oenc = OrdinalEncoder()  cat\_encoded = oenc.fit\_transform(df\_cat)  df\_cat = pd.DataFrame(cat\_encoded, columns=df\_cat.columns)  df\_cat.reset\_index(drop=True, inplace=True)    # Concatenate all the preprocessed data and labels together  l = pd.DataFrame(Labels, columns=['Label'])  prepared\_Dataset = pd.concat([df\_num, df\_cat, l], axis=1)    return prepared\_Dataset  # Read the main data set and select the best features for model training  if \_\_name\_\_=='\_\_main\_\_':  df = pd.read\_csv('Dataset\_Small.csv', on\_bad\_lines='skip', engine='python', encoding='ISO-8859-1')  df = shuffle(df)    df.reset\_index(drop=True, inplace=True)  df.info(verbose=True, show\_counts=True)  train\_set, test\_set = data\_splitting(df)  train\_set.reset\_index(drop=True, inplace=True)  test\_set.reset\_index(drop=True, inplace=True)    Train\_Data = prep\_for\_feture\_selection(train\_set)  Train\_Data.reset\_index(drop=True, inplace=True)  x = most\_corr\_features(Train\_Data) |

| Data\_cleaning.py |
| --- |
| import pandas as pd  from sklearn.utils import shuffle  from sklearn.preprocessing import OrdinalEncoder  from sklearn.impute import SimpleImputer  from sklearn.preprocessing import StandardScaler  from sklearn.preprocessing import OrdinalEncoder  from sklearn.model\_selection import train\_test\_split  import pickle  # Split the data set to train and test  def Splitting\_The\_Data(DataFrame):  df\_train, df\_test = train\_test\_split(DataFrame, test\_size=0.3, stratify=DataFrame['Label'], random\_state=42)  return df\_train,df\_test    # apply the preprocessing to the training set and save the imputers and scalers for later use  def Preprocessing\_train(train\_set):  DataFrame = train\_set.copy()  df\_num = DataFrame[['udp.srcport','frame.len','ip.ttl','ip.dsfield','tcp.dstport']].astype('float64')    df\_cat = DataFrame[['frame.time\_epoch','frame.time','ip.frag\_offset','ip.src','ip.version',  'ip.checksum','tcp.flags.reset','tcp.flags.syn','tcp.window\_size',  'tcp.flags.fin','ip.proto','eth.dst','frame.protocols','ip.len','tcp.seq']].astype("object")      Labeled\_Data = train\_set['Label'].copy()  del train\_set['Label']    df\_num.reset\_index(drop=True, inplace=True)  df\_cat.reset\_index(drop=True, inplace=True)  # apply simple imputer to numerical data using mean  imputer = SimpleImputer(strategy="mean")  X = imputer.fit\_transform(df\_num)  df\_num = pd.DataFrame(X, columns=df\_num.columns)  with open('imputer\_num.pickle', 'wb') as file:  pickle.dump(imputer, file)      # apply standard scaler  stsc = StandardScaler()  n = stsc.fit\_transform(df\_num)  df\_num = pd.DataFrame(n, columns=df\_num.columns)  with open('stsc.pickle', 'wb') as file:  pickle.dump(stsc, file)    df\_num.reset\_index(drop=True, inplace=True)    # apply simple imputer to categorical data using the most frequent value  imputer = SimpleImputer(strategy="most\_frequent")  Y=imputer.fit\_transform(df\_cat)  df\_cat = pd.DataFrame(Y, columns = df\_cat.columns)  with open('imputer\_cat.pickle', 'wb') as file:  pickle.dump(imputer, file)    # apply ordinal encoding  oenc= OrdinalEncoder(handle\_unknown='use\_encoded\_value', unknown\_value=-1)  cat\_encoded = oenc.fit\_transform(df\_cat)  df\_cat=pd.DataFrame(cat\_encoded, columns = df\_cat.columns)  with open('cat\_enc.pickle', 'wb') as file:  pickle.dump(oenc, file)      L = pd.DataFrame(Labeled\_Data,columns = ['Label'])  prepared\_Dataset = pd.concat([df\_num, df\_cat, L] , axis=1)  prepared\_Dataset.reset\_index(drop=True, inplace=True)  prepared\_Dataset.info(verbose=True,show\_counts=True)  return prepared\_Dataset  # apply the saved preprocessing onto the test set  def Preprocessing\_test(test\_set):  DataFrame = test\_set.copy()  Labeled\_Data = test\_set['Label'].copy()  test\_set.drop(columns=['Label'], inplace=True)  df\_num = DataFrame[['udp.srcport','frame.len','ip.ttl','ip.dsfield','tcp.dstport']].astype('float64')  df\_cat = DataFrame[['frame.time\_epoch','frame.time','ip.frag\_offset','ip.src','ip.version',  'ip.checksum','tcp.flags.reset','tcp.flags.syn','tcp.window\_size',  'tcp.flags.fin','ip.proto','eth.dst','frame.protocols','ip.len','tcp.seq']].astype("object")    with open('imputer\_num.pickle', 'rb') as file:  loaded\_model = pickle.load(file)  x = loaded\_model.transform(df\_num)  df\_num = pd.DataFrame(x, columns=df\_num.columns)  with open('stsc.pickle', 'rb') as file:  loaded\_model = pickle.load(file)  x = loaded\_model.transform(df\_num)  df\_num = pd.DataFrame(x, columns=df\_num.columns)  with open('imputer\_cat.pickle', 'rb') as file:  loaded\_model = pickle.load(file)  x = loaded\_model.transform(df\_cat)  df\_cat = pd.DataFrame(x, columns=df\_cat.columns)  with open('cat\_enc.pickle', 'rb') as file:  loaded\_model = pickle.load(file)    x = loaded\_model.transform(df\_cat)  df\_cat = pd.DataFrame(x, columns=df\_cat.columns)  L = pd.DataFrame(Labeled\_Data, columns=['Label'])  prepared\_Dataset = pd.concat([df\_num, df\_cat, L], axis=1)  prepared\_Dataset.dropna(subset=['Label'], inplace=True)  prepared\_Dataset.reset\_index(drop=True, inplace=True)  return prepared\_Dataset  if \_\_name\_\_=='\_\_main\_\_':  df = pd.read\_csv('Dataset\_Small.csv', on\_bad\_lines='skip', engine='python', encoding='ISO-8859-1')  df = shuffle(df)  df.reset\_index(drop=True, inplace=True)  #print(DataFrame['Labeledd'].value\_counts())    train\_set,test\_set = Splitting\_The\_Data(df)  train\_set.reset\_index(drop=True, inplace=True)  test\_set.reset\_index(drop=True, inplace=True)    #Preprocessing  Data\_Train = Preprocessing\_train(train\_set)  Data\_Test = Preprocessing\_test(test\_set)  Data\_Train.reset\_index(drop=True, inplace=True)  Data\_Test.reset\_index(drop=True, inplace=True) |

| Isolation\_forest.py |
| --- |
| import pandas as pd  from sklearn.ensemble import IsolationForest  from sklearn.metrics import classification\_report, accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score, confusion\_matrix  import joblib  import matplotlib.pyplot as plt  import numpy as np  # Load the datasets  train\_data = pd.read\_csv('Training\_Dataset.csv')  test\_data = pd.read\_csv('Testing\_Dataset.csv')  # Define the label mapping  label\_mapping = {'Benign': 1, 'Malware': -1}  # Map the labels in the train and test datasets  train\_data['Label'] = train\_data['Label'].replace(label\_mapping)  test\_data['Label'] = test\_data['Label'].replace(label\_mapping)  # Extract features and labels  X\_train = train\_data.drop(columns=['Label'])  y\_train = train\_data['Label']  X\_test = test\_data.drop(columns=['Label'])  y\_test = test\_data['Label']  # Initialize the Isolation Forest model with the best hyperparameters  iso\_forest = IsolationForest(n\_estimators=100, max\_samples='auto', contamination=0.05, max\_features=1.0, random\_state=42)  # Fit the model  iso\_forest.fit(X\_train)  # Predict on training set  y\_train\_pred = iso\_forest.predict(X\_train)  '''  from numpy import where  anom\_index = where(y\_train\_pred==-1)  values = X\_train[anom\_index]  plt.scatter(X\_train[:,0], X\_train[:,1])  plt.scatter(values[:,0], values[:,1], color='r')  plt.show()  '''  # Predict on testing set  y\_pred = iso\_forest.predict(X\_test)  # Print out some evaluation metrics for training set  print("Training set performance:")  print(classification\_report(y\_train, y\_train\_pred))  # Convert predictions to binary format for training metrics  y\_train\_binary = (y\_train == 1).astype(int)  y\_train\_pred\_binary = (y\_train\_pred == 1).astype(int)  # Calculate training metrics  accuracy\_train = accuracy\_score(y\_train\_binary, y\_train\_pred\_binary)  precision\_train = precision\_score(y\_train\_binary, y\_train\_pred\_binary, zero\_division=0)  recall\_train = recall\_score(y\_train\_binary, y\_train\_pred\_binary, zero\_division=0)  f1\_train = f1\_score(y\_train\_binary, y\_train\_pred\_binary, zero\_division=0)  roc\_auc\_train = roc\_auc\_score(y\_train\_binary, y\_train\_pred\_binary)  cm = confusion\_matrix(y\_train\_binary, y\_train\_pred\_binary)  tn, fp, fn, tp = cm.ravel()  fnr\_test = fn / (fn + tp)  print(f'Accuracy = {accuracy\_train:.1%}')  print(f'ROC AUC = {roc\_auc\_train:.1%}')  print(f'Precision = {precision\_train:.1%}')  print(f'Recall = {recall\_train:.1%}')  print(f'F1 Score = {f1\_train:.1%}')  print("Confusion Matrix:\n", confusion\_matrix(y\_train\_binary, y\_train\_pred\_binary))  print(f'False Negative Rate (FNR) = {fnr\_test:.1%}')  # Print out some evaluation metrics for testing set  print("\nTesting set performance:")  print(classification\_report(y\_test, y\_pred))  # Convert predictions to binary format for testing metrics  y\_test\_binary = (y\_test == 1).astype(int)  y\_pred\_binary = (y\_pred == 1).astype(int)  # Calculate testing metrics  accuracy\_test = accuracy\_score(y\_test\_binary, y\_pred\_binary)  precision\_test = precision\_score(y\_test\_binary, y\_pred\_binary, zero\_division=0)  recall\_test = recall\_score(y\_test\_binary, y\_pred\_binary, zero\_division=0)  f1\_test = f1\_score(y\_test\_binary, y\_pred\_binary, zero\_division=0)  roc\_auc\_test = roc\_auc\_score(y\_test\_binary, y\_pred\_binary)  cm = confusion\_matrix(y\_test\_binary, y\_pred\_binary)  tn, fp, fn, tp = cm.ravel()  fnr\_test = fn / (fn + tp)  print(f'Accuracy = {accuracy\_test:.1%}')  print(f'ROC AUC = {roc\_auc\_test:.1%}')  print(f'Precision = {precision\_test:.1%}')  print(f'Recall = {recall\_test:.1%}')  print(f'F1 Score = {f1\_test:.1%}')  print("Confusion Matrix:\n", confusion\_matrix(y\_test\_binary, y\_pred\_binary))  print(f'False Negative Rate (FNR) = {fnr\_test:.1%}')  # Save the model to a file  joblib\_file = "isolation\_forest\_model.pkl"  joblib.dump(iso\_forest, joblib\_file)  # Count the number of benign and malware instances in the test set  benign\_count = (y\_test == 1).sum()  malware\_count = (y\_test == -1).sum()  # Adjusted counts for visualization  adjusted\_malware\_count = 50000 # Arbitrary large value to make the bar visible  adjusted\_benign\_count = benign\_count - adjusted\_malware\_count # Adjust benign count accordingly  # Bar chart for the number of benign and malware instances  plt.figure(figsize=(6, 6))  plt.bar(['Benign', 'Malware'], [adjusted\_benign\_count, adjusted\_malware\_count], color=['green', 'red'])  plt.title('Number of Benign and Malware Instances in the Test Set')  plt.xlabel('Class')  plt.ylabel('Count')  plt.show()  # Print the number of benign and anomaly instances  print(f'Number of Benign instances: {benign\_count}')  print(f'Number of Malware instances: {malware\_count}') |

| Result.py |
| --- |
| import sys  from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QLabel, QFrame, QHBoxLayout, QSpacerItem, QSizePolicy  from PyQt5.QtGui import QPixmap, QFont  from PyQt5.QtCore import Qt, QTimer  import matplotlib.pyplot as plt  import pandas as pd  import joblib  import pickle  import subprocess  import os  class ResultPage(QWidget):  def \_\_init\_\_(self):  super().\_\_init\_\_()  self.initUI()  def initUI(self):  # Window settings  self.setWindowTitle('Result Page')  self.setGeometry(100, 100, 1600, 900)  self.setStyleSheet("background-color: #1D0E25;")    # Main layout  main\_layout = QVBoxLayout()    # Logo  logo\_label = QLabel(self)  pixmap = QPixmap('logo.png').scaled(408, 386, Qt.KeepAspectRatio)  logo\_label.setPixmap(pixmap)  main\_layout.addWidget(logo\_label, alignment=Qt.AlignLeft | Qt.AlignTop)    # Container for plot and boxes  container\_layout = QHBoxLayout()    # Plot  plot\_label = QLabel(self)  plot\_pixmap = QPixmap('plot.png').scaled(1000, 700, Qt.KeepAspectRatio)  plot\_label.setPixmap(plot\_pixmap)  container\_layout.addWidget(plot\_label, alignment=Qt.AlignLeft)    # Add a spacer item to control spacing between plot and boxes  container\_layout.addItem(QSpacerItem(20, 700, QSizePolicy.Minimum, QSizePolicy.Expanding))    # Boxes for counts and percentages  boxes\_layout = QVBoxLayout()    # Green Box - Normal Data  normal\_data\_box = QFrame(self)  normal\_data\_box.setFixedSize(574, 281)  normal\_data\_box.setStyleSheet("background-color: #E0F2E4; border: 2px solid green;")  normal\_data\_layout = QVBoxLayout(normal\_data\_box)    normal\_label = QLabel("Normal:")  normal\_label.setFont(QFont('Arial', 14))  normal\_label.setStyleSheet("color: green;")    self.normal\_count\_label = QLabel()  self.normal\_count\_label.setFont(QFont('Arial', 12))  self.normal\_count\_label.setStyleSheet("color: gray;")    self.normal\_percentage\_label = QLabel()  self.normal\_percentage\_label.setFont(QFont('Arial', 12))  self.normal\_percentage\_label.setStyleSheet("color: green;")    normal\_data\_layout.addWidget(normal\_label)  normal\_data\_layout.addWidget(self.normal\_count\_label)  normal\_data\_layout.addWidget(self.normal\_percentage\_label)  normal\_data\_box.setLayout(normal\_data\_layout)    # Red Box - Malware Data  malware\_data\_box = QFrame(self)  malware\_data\_box.setFixedSize(574, 281)  malware\_data\_box.setStyleSheet("background-color: #F9EBED; border: 2px solid red;")  malware\_data\_layout = QVBoxLayout(malware\_data\_box)    malware\_label = QLabel("Malware:")  malware\_label.setFont(QFont('Arial', 14))  malware\_label.setStyleSheet("color: red;")    self.malware\_count\_label = QLabel()  self.malware\_count\_label.setFont(QFont('Arial', 12))  self.malware\_count\_label.setStyleSheet("color: gray;")    self.malware\_percentage\_label = QLabel()  self.malware\_percentage\_label.setFont(QFont('Arial', 12))  self.malware\_percentage\_label.setStyleSheet("color: red;")    malware\_data\_layout.addWidget(malware\_label)  malware\_data\_layout.addWidget(self.malware\_count\_label)  malware\_data\_layout.addWidget(self.malware\_percentage\_label)  malware\_data\_box.setLayout(malware\_data\_layout)    boxes\_layout.addWidget(normal\_data\_box)  boxes\_layout.addWidget(malware\_data\_box)    container\_layout.addLayout(boxes\_layout)  main\_layout.addLayout(container\_layout)    self.setLayout(main\_layout)    # Update counts and percentages  self.update\_counts\_and\_percentages()    # Start a QTimer to call the alert window after 30 seconds  self.timer = QTimer(self)  self.timer.setSingleShot(True)  self.timer.timeout.connect(self.show\_alert)  self.timer.start(30000) # 30 seconds  def update\_counts\_and\_percentages(self):  print("Updating counts and percentages...") # Debug statement  df = pd.read\_csv('output.csv', on\_bad\_lines='skip', engine='python', encoding='ISO-8859-1')  new\_data = self.feature\_selection\_cleaning(df)    with open('isolation\_forest\_model.pkl', 'rb') as file:  model = joblib.load(file)  prediction = model.predict(new\_data)    benign\_count = (prediction == 1).sum()  malware\_count = (prediction == -1).sum()  total\_count = benign\_count + malware\_count    benign\_percentage = (benign\_count / total\_count) \* 100 if total\_count > 0 else 0  malware\_percentage = (malware\_count / total\_count) \* 100 if total\_count > 0 else 0  self.normal\_count\_label.setText(f"{benign\_count}")  self.normal\_percentage\_label.setText(f"{benign\_percentage:.2f}%")    self.malware\_count\_label.setText(f"{malware\_count}")  self.malware\_percentage\_label.setText(f"{malware\_percentage:.2f}%")    # Generate the plot and save it as plot.png  plt.figure(figsize=(12, 7))  plt.bar(['Benign', 'Malware'], [benign\_count, malware\_count], color=['blue', 'red'])  plt.title('Number of Benign and Malware Instances')  plt.xlabel('Class')  plt.ylabel('Count')  plt.savefig('plot.png')  print("Counts and percentages updated.") # Debug statement    def feature\_selection\_cleaning(self, df):  print("Performing feature selection and cleaning...") # Debug statement  DataFrame = df.copy()  df\_num = DataFrame[['udp.srcport','frame.len','ip.ttl','ip.dsfield','tcp.dstport']].astype('float64')  df\_cat = DataFrame[['frame.time\_epoch','frame.time','ip.frag\_offset','ip.src','ip.version',  'ip.checksum','tcp.flags.reset','tcp.flags.syn','tcp.window\_size',  'tcp.flags.fin','ip.proto','eth.dst','frame.protocols','ip.len','tcp.seq']].astype("object")  with open('imputer\_num.pickle', 'rb') as file:  loaded\_model = pickle.load(file)  x = loaded\_model.transform(df\_num)  df\_num = pd.DataFrame(x, columns=df\_num.columns)  with open('stsc.pickle', 'rb') as file:  loaded\_model = pickle.load(file)  x = loaded\_model.transform(df\_num)  df\_num = pd.DataFrame(x, columns=df\_num.columns)  with open('imputer\_cat.pickle', 'rb') as file:  loaded\_model = pickle.load(file)  x = loaded\_model.transform(df\_cat)  df\_cat = pd.DataFrame(x, columns=df\_cat.columns)  with open('cat\_enc.pickle', 'rb') as file: # Handle unknown categories using OrdinalEncoder  loaded\_model = pickle.load(file)  x = loaded\_model.transform(df\_cat)  df\_cat = pd.DataFrame(x, columns=df\_cat.columns)  new\_data = pd.concat([df\_num, df\_cat], axis=1)  new\_data.reset\_index(drop=True, inplace=True)  print("Feature selection and cleaning done.") # Debug statement  return new\_data  def show\_alert(self):  print("Calling alert.py...") # Debug statement  try:  alert\_script\_path = "alert.py"  if os.path.exists(alert\_script\_path):  subprocess.run(["python", alert\_script\_path])  print("alert.py executed.") # Debug statement  else:  print(f"alert.py not found at path: {alert\_script\_path}")  except Exception as e:  print(f"Error executing alert.py: {e}") # Debug statement  if \_\_name\_\_ == '\_\_main\_\_':  app = QApplication(sys.argv)  result\_page = ResultPage()  result\_page.show()  sys.exit(app.exec\_()) |

| Alert.py |
| --- |
| import sys  from PyQt5.QtWidgets import QApplication, QWidget, QLabel, QPushButton, QVBoxLayout  from PyQt5.QtGui import QFont  import subprocess  from newdata\_clean\_predict import generate\_plot\_and\_stats  class SecurityAlertWindow(QWidget):  def \_\_init\_\_(self, anomaly\_percentage):  super().\_\_init\_\_()  self.anomaly\_percentage = anomaly\_percentage  self.initUI()  def initUI(self):  self.setWindowTitle('Security Alert')  self.setGeometry(100, 100, 700, 300)  alert\_box = QWidget(self)  alert\_box.setGeometry(29, 57, 642, 185)  alert\_layout = QVBoxLayout()  if self.anomaly\_percentage <= 10:  alert\_box.setStyleSheet("border: 0px solid #22BB55; background-color: #FFFFFF; border-radius: 15px;")  title\_label = QLabel("Security Alert - Low Threat Level", self)  title\_label.setStyleSheet("color: #22BB55;")  message = "Great news! Your network is currently secure."  button\_color = "#22BB55"  elif 10 < self.anomaly\_percentage <= 45:  alert\_box.setStyleSheet("border: 0px solid #EEAA44; background-color: #FFFFFF; border-radius: 15px;")  title\_label = QLabel("Security Alert - Moderate Threat Level", self)  title\_label.setStyleSheet("color: #EEAA44;")  message = "We have detected an increase in malware presence on your network. Review and take necessary actions to enhance security."  button\_color = "#EEAA44"  else:  alert\_box.setStyleSheet("border: 0px solid red; background-color: #FFFFFF; border-radius: 15px;")  title\_label = QLabel("Security Alert - High Threat Level", self)  title\_label.setStyleSheet("color: red;")  message = "Immediate action is required. We recommend a thorough security audit and malware remediation."  button\_color = "#B00020"  title\_label.setFont(QFont('Arial', 24, QFont.Bold))  alert\_layout.addWidget(title\_label)  percentage\_label = QLabel(f"Your network has an anomaly percentage of {self.anomaly\_percentage}%", self)  percentage\_label.setFont(QFont('Arial', 11))  percentage\_label.setStyleSheet("color: black;")  alert\_layout.addWidget(percentage\_label)  message\_label = QLabel(message, self)  message\_label.setFont(QFont('Arial', 10))  message\_label.setStyleSheet("color: black;")  alert\_layout.addWidget(message\_label)  more\_info\_button = QPushButton('MORE INFO', self)  more\_info\_button.setFont(QFont('Arial', 16, QFont.Bold))  more\_info\_button.setStyleSheet(  f"background-color: {button\_color}; color: white; border-radius: 23px; width: 80px; height: 49px;")  more\_info\_button.clicked.connect(self.openMoreInfo)  alert\_layout.addWidget(more\_info\_button)  alert\_box.setLayout(alert\_layout)  def openMoreInfo(self):  subprocess.run(["python", "moreinfo.py"])  if \_\_name\_\_ == '\_\_main\_\_':  benign\_count, normal\_percentage, malware\_count, malware\_percentage = generate\_plot\_and\_stats()  anomaly\_percentage = malware\_percentage  app = QApplication(sys.argv)  ex = SecurityAlertWindow(anomaly\_percentage)  ex.show()  sys.exit(app.exec\_()) |

| Alertinfo.py |
| --- |
| import sys  from PyQt5.QtWidgets import QApplication, QWidget, QLabel  from PyQt5.QtGui import QPixmap  class MoreInfoWindow(QWidget):  def \_init(self): # Corrected from \_init to \_init\_  super().\_init\_()  self.initUI()  def initUI(self):  self.setWindowTitle('More Information')  self.setGeometry(100, 100, 800, 600)  self.setStyleSheet("background-color: white;")  label = QLabel(self)  label.setGeometry(0, 0, 800, 600)  pixmap = QPixmap("moreinfo.png")  label.setPixmap(pixmap)  label.setScaledContents(True)  if \_name\_ == '\_main': # Corrected from '\_main' to '\_main\_'  app = QApplication(sys.argv)  ex = MoreInfoWindow()  ex.show()  sys.exit(app.exec\_()) |