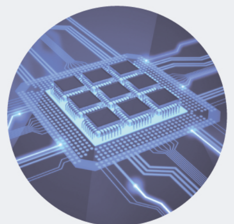
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

MESTRADO EM ENGENHARIA DE SISTEMAS COMPUTACIONAIS CRÍTICOS



Monitorização Integrada para Cibersegurança em Cenários Residênciais

MÁRIO MIGUEL SILVA DE SÁ CARNEIRO
outubro de 2024

P.PORTO

Integrated Monitoring for Cyber-Security in Residential Scenarios

Mário Miguel Silva Sá Carneiro

Dissertation submitted in partial fulfilment of the requirements for the Master's degree in Critical Computing Systems Engineering

Supervisor: Luis Miguel Moreira Lino Ferreira
Co-Supervisor: Ricardo Augusto Rodrigues da Silva Severino

Evaluation Committee:
President:
Luis Miguel Pinho

Members:
Daniel Costa
Luis Lino Ferreira

Porto, October 11, 2024

Statement of Integrity

I hereby declare having conducted this academic work with integrity. I have not plagiarised or applied any form of undue use of information or falsification of results along the process leading to its elaboration.

I declare that the work presented in this document is original and my own, and has not previously been used for any other purpose.

I further declare that I have fully followed the Code of Good Practices and Conduct of the Polytechnic Institute of Porto.

ISEP, Porto, October 11, 2024

Abstract

The increasing number of Internet of Things devices and the increasing adoption of smart homes have led to an increase in risk related to cybersecurity. The purpose of this dissertation is to examine these limitations and propose innovative solutions for anomaly detection using machine learning (ML) methods.

The objectives and motivations for this work, which focuses on improving home network security, are explained in Chapter 1, which also provides a review of smart homes and their connections with cybersecurity issues.

In Chapter 2, "State of the Art," the rise of IoT in everyday use and related safety concerns are addressed. In addition, it covers over basic concepts like machine learning strategies and the way these interact with intrusion detection systems (IDS). In order to mitigate increasing threats it considers that integrating ML with cybersecurity in IoT systems is important.

The machine learning techniques selected for this project are presented in Chapter 3, with a focus on creating a reliable anomaly detection pipeline.Comprehensive data pre-processing, including cleaning, merging, normalization, and analysis, ensures sure the data is suitable for model training.

In Chapter 4, training individual and ensemble models will be addressed along with an analysis of performance metrics in scenarios using binary and multi-class classification.Z-Score normalization is one strategy that is frequently used to handle unbalanced datasets. It has been demonstrated to perform better than Min-Max, especially when applied to the UNSW-NB15 dataset.

The implementation of APIs using Streamlit for real-time visualisation and FastAPI for back-end integration with ML models will be discussed in Chapter 5.This combination enables the ability to anticipate cyberattacks and visually represent anomalies in an effective way.

The testbed built to automate cyberattacks and extract important features for model training will be discussed in detail in Chapter 6.

The results of the evaluations, that compare the performance of the individual models and the ensemble, are presented in Chapter 7. The ensemble performed better than expected, especially when it came to identifying anomalies in multi-class environments. It achieved this with high accuracy and a significant reduction in false positives and negatives.

Chapter 8 ends with an overview of the project's conclusions and contributions.The main findings emphasize the significance of selecting normalization strategies and the advantages of using ensemble models to improve attack detection.

Keywords: IoT Security, Anomaly Detection, Cybersecurity, Machine Learning, Ensemble Models, Intrusion Detection Systems (IDS), API Integration

Resumo

O número crescente de dispositivos da *Internet das Coisas* e a crescente adoção de casas inteligentes conduziram a um aumento do risco relacionado com a cibersegurança. O objetivo desta dissertação é examinar estas limitações e propor soluções inovadoras para a deteção de anomalias utilizando métodos de machine learning (ML).

Os objetivos e as motivações deste trabalho, que se centra na melhoria da segurança das redes domésticas, são explicados no Capítulo 1, que também apresenta uma análise das casas inteligentes e das suas ligações a questões de cibersegurança.

No Capítulo 2, "Estado da arte", é abordado o aumento da IoT na utilização quotidiana e as preocupações de segurança relacionadas. Além disso, aborda conceitos básicos como estratégias de machine learning e a forma como interagem com os sistemas de deteção de intrusões (IDS).

As técnicas de machine learning selecionadas para este projeto são apresentadas no Capítulo 3, com o objetivo de criar um pipeline de deteção de anomalias fiável. Um pré-processamento abrangente dos dados, incluindo limpeza, fusão, normalização e análise, garante que os dados são adequados para o treino do modelo.

No Capítulo 4, será abordado o treino de modelos individuais e de conjuntos, bem como uma análise das métricas de desempenho em cenários que utilizam a classificação binária e multiclasse. A normalização do Z-Score é uma estratégia frequentemente utilizada para lidar com conjuntos de dados desequilibrados.

A implementação de APIs utilizando Streamlit para visualização em tempo real e FastAPI para integração de back-end com modelos de ML será discutida no Capítulo 5. Esta combinação permite a capacidade de antecipar ciberataques e representar visualmente as anomalias de uma forma eficaz.

A bancada de ensaio construída para automatizar ciberataques e extrair caraterísticas importantes para o treino de modelos será analisada em pormenor no Capítulo 6.

Os resultados das avaliações, que comparam o desempenho dos modelos individuais e do conjunto, são apresentados no Capítulo 7. O conjunto teve um desempenho melhor do que o esperado, especialmente quando se tratou de identificar anomalias em ambientes multiclasse.

O capítulo 8 termina com uma síntese das conclusões e contributos do projeto. As principais conclusões salientam a importância da seleção de estratégias de normalização e as vantagens da utilização de modelos de conjunto para melhorar a deteção de ataques.

Acknowledgement

To begin with, I would like to express my gratitude to Professors Luís Ferreira and Ricardo Severino for their important guidance, challenging conversations, and ongoing support during this research. Their collaboration was essential in facilitating the research's progress and giving me valuable and in-depth knowledge.

I also want to express my gratitude to my family, especially to my parents, for their continuous love, support, and understanding throughout this challenging time in my life. It would never have been possible to accomplish this milestone without their help.

I would also like to express my gratitude to my friends and coworkers who have supported and assisted me in different ways throughout the project's steps, whether it be by offering guidance, encouragement, or hands-on assistance.

Lastly, I would like to thank everyone who has helped me grow and develop in both my personal and professional life by having an impact on my academic and personal journey.

Mário Sá Carneiro

Contents

List of Figures xvii

List of Tables xix

List of Abbreviations xxi

List of Symbols xxiii

List of Acronyms xxv

1 Introduction 1
- 1.1 Contextualisation . . . 1
- 1.2 Problem Statement . . . 2
- 1.3 Motivation . . . 3
- 1.4 Thesis objectives . . . 4
- 1.5 Document Structure . . . 4

2 State of the Art 5
- 2.1 IoT and Cybersecurity . . . 5
  - 2.1.1 Fundamental Concepts in Cybersecurity . . . 6
    - Confidentiality, Integrity, Availability (CIA) . . . 7
    - Encryption . . . 8
  - 2.1.2 Existing Approaches to Residential Cybersecurity . . . 9
    - Network Segmentation . . . 10
  - 2.1.3 Types of Attacks . . . 11
- 2.2 Machine Learning (ML) and Artificial Intelligence (AI) . . . 13
  - 2.2.1 Recent Advances and Future Trends in AI and ML . . . 14
  - 2.2.2 Types of ML Algorithms: . . . 14
  - 2.2.3 Machine Learning for Intrusion Detection System (IDS) in Embedded Systems . . . 15
    - Challenges and Constraints of Embedded Systems . . . 16
    - ML Libraries in Low Level Languages . . . 16
- 2.3 Data Preparation and Preprocessing . . . 17
  - 2.3.1 The Importance of Preprocessing . . . 17
  - 2.3.2 Data Normalization . . . 17
- 2.4 Variable Encoding and Feature Selection . . . 19
  - 2.4.1 One-Hot Encoding . . . 19
  - 2.4.2 Feature Selection . . . 20
  - 2.4.3 Significance of Dimensionality Reduction . . . 20
- 2.5 Binary vs. Multi-Class Classification . . . 20
  - 2.5.1 Differences between Binary Classification and Multiclass . . . 20

2.6 ML Algorithms for Intrusion Detection 21
2.6.1 Algorithms Used in Project 21
2.6.2 Model Comparison 22
Performance Metrics 22
2.7 Model Combination and Ensemble Learning 23
2.7.1 Ensemble Learning 23
2.8 APIs for Integrating ML Models 24
2.8.1 FastAPI: Backend Interaction and Integration 24
FastAPI's advantages 24
FastAPI alternatives 24
2.8.2 Streamlit: Visualisation and Interaction with the Model 25
Streamlit's advantages 25
Streamlit Alternatives 25
2.9 State of the Art Conclusion 25

3 ML-Based Algorithm 27
3.1 Workflow of the Anomaly Detection Models 27
3.1.1 Data Collection 29
Explored Datasets 29
Selection of Datasets 29
Creating a Combined Dataset 30
Initial Data Cleaning 30
Data Cleaning Summary 34
3.1.2 One-Hot Encoding 35
Application of One-Hot Encoding 35
3.1.3 Data Normalization 35
Min-Max Scaling 35
Z-Score Normalization (Standardization) 36
Comparing Techniques 36
3.1.4 Label Encoding 37
Binary Encoding 37
Multi-class encoding 37
3.1.5 Feature Selection 37
Feature Selection for Binary Classification 38
Feature Selection for Multi-class Classification 38

4 Model Training 39
4.1 Individual Model Training 39
4.1.1 Data Preparation 39
4.1.2 Selected Algorithms 39
4.1.3 Evaluation Metrics 40
4.1.4 Results and Metrics 40
Individual Model Binary Evaluation 40
Individual Model Multi-Class Evaluation 42
4.2 Model Combination with Bagging 42
4.2.1 Binary Bagging 43
Ensembled Binary Model Metrics and Results 43
4.2.2 Multi-Class Bagging 44
Ensembled Multi-Class Model Metrics and Results 44

5 Implementation of APIs for Visualisation and Interaction **47**
5.1 Data Format 47
5.2 FastAPI: Backend Interaction and Model Integration 48
5.2.1 API Workflow 48
5.3 Streamlit: Frontend and Real-time Dashboard 49
5.3.1 Dashboard Features 49
5.3.2 Communication with FastAPI 50
5.3.3 Final Layout and Usability Enhancements 51
5.4 Conclusion 51

6 Feature Engineering and Data Preparation **53**
6.1 Testbed for Cyberattack Automation and Feature Extraction 53
6.1.1 General Objectives 53
6.1.2 Structure of *Testbed* 53
6.1.3 Technical details 54
Directory Structure 54
Packet Capture Process 54
Attack Execution 54
Detailed Event Log 55
6.1.4 Conclusion of the Testbed 55

7 Results and Discussion **57**
7.1 Introduction 57
7.2 Evaluation of Model Performance 57
7.2.1 Individual Models 57
Binary Classification 57
Multi-Class Classification 60
7.2.2 Ensembled Models 62
7.3 Testbed Evaluation 64
7.4 Application Input 65
7.4.1 Input Data Format 65
7.4.2 Data Analysis Process 66
7.4.3 User Interface 66
7.4.4 Correlated Feature Analysis for Binary and Multi-Class Classification 67
Binary Classification 68
Multi-Class Classification 73
MultiClass Prediction Tests 78
7.5 Limitations 78
7.5.1 Dataset Selection 78
7.5.2 Attack Type Options 78
7.5.3 Problems with Merging Datasets 78
7.5.4 Data Cleaning 78
7.5.5 Data Visualization 79
7.5.6 No integration with the Cloud 79
7.5.7 Live Detection Limitations 79
7.5.8 Overfitting and Incorrect Prediction in the API 79
7.5.9 Impossibility of Model Retraining on the Testbed 79
7.5.10 Time limit for in-depth Research 79
7.5.11 Using Python instead of C 79

7.5.12 Functions of Parallelism and Complexity 80

8 Discussion and Conclusion **81**
8.1 Summary of Findings 81
8.1.1 Key Insights 81
Meaningful Insights 81
8.1.2 Contributions to the Field 82
8.2 Implications 82
8.2.1 Practical Implications 82
8.2.2 Theoretical Implications 82
8.3 Future Work 83
8.3.1 Short-Term Improvements 83
8.3.2 Long-Term Research Directions 83

Bibliography **85**

A Project Structure and Code Overview **89**
A.1 API 89
A.2 Helpers Directory 89
A.2.1 Aux_Testbed_Pipeline 89
A.2.2 Aux_Scripts 90
A.3 Pipeline Directory 90
A.3.1 0.Cleaning 90
A.3.2 1.Training 90
A.4 Scripts Directory 90

B API Implementation **91**
B.1 FastAPI Application 91
B.2 Streamlit Dashboard 94

C Pipeline Implementation **99**
C.1 Overview 99
C.2 0.Cleaning 99
C.3 1.Training 103

D Testbed Implementation **115**
D.1 Testbed Script 115

A Training Evaluations **123**
A.1 Binary Classification - Min-Max-Score - UNSW 123
A.1.1 Label Correlation 123
A.1.2 Logistic Regression 123
A.1.3 K-Nearest Neighbors (KNN) 124
A.1.4 Decision Tree 124
A.1.5 Random Forest 125
A.1.6 MLP Classifier 125
A.1.7 SGD Classifier 126
A.1.8 Ensembled Model 126
A.2 Binary Classification - Z-Score - UNSW 126
A.2.1 Label Correlation 126

A.2.2 Logistic Regression 127
A.2.3 K-Nearest Neighbors (KNN) 127
A.2.4 Decision Tree 128
A.2.5 Random Forest 128
A.2.6 MLP Classifier 129
A.2.7 SGD Classifier 129
A.2.8 Ensembled Model 130
A.3 Multi-Class Classification - Min-Max-Score - UNSW 130
A.3.1 Label Correlation 130
A.3.2 Logistic Regression 130
A.3.3 K-Nearest Neighbors (KNN) 131
A.3.4 Decision Tree 131
A.3.5 Random Forest 132
A.3.6 MLP Classifier 132
A.3.7 SGD Classifier 133
A.3.8 Ensembled Model 133
A.4 Multi-Class Classification - Z-Score - UNSW 134
A.4.1 Label Correlation 134
A.4.2 Logistic Regression 134
A.4.3 K-Nearest Neighbors (KNN) 135
A.4.4 Decision Tree 135
A.4.5 Random Forest 136
A.4.6 MLP Classifier 136
A.4.7 SGD Classifier 137
A.4.8 Ensembled Model 137

List of Figures

1.1 IoT Vulnerabilities linking with layered IoT architecture across several layers. 2

2.1 Home Automation Market Size, 2022 to 2032 Prediction (In USD). 5
2.2 CIA Triad 7
2.3 Cyber Attacks Over the years. 11
2.4 Amount of Losses of Cyberattacks last twelve years (2010-2021), U.S.A. . 13
2.5 First AI Game of Checkers. 13

3.1 Workflow of the Anomaly Detection System 28
3.2 Distribution of Normal and Abnormal Labels in the UNSW-NB15 Dataset . 31
3.3 Distribution of Normal and Abnormal Labels in the CTU-13 Dataset 32
3.4 Distribution of Normal and Abnormal Labels in the CTU-UNSW Dataset . 32
3.5 Distribution of Attack Categories in the UNSW-NB15 Dataset 33
3.6 Distribution of Attack Categories in the CTU-UNSW Dataset 34

4.1 Accuracy results for Min-Max Normalization for Binary Classification 43
4.2 Accuracy results for Z-Score Normalization for Binary Classification 44
4.3 Accuracy results for Min-Max Normalization for Multi-Class Classification . 45
4.4 Accuracy results for Z-Score Normalization for Multi-Class Classification . . 45

5.1 Live Detection StreamlitAPI Screenshot 49
5.2 History Detection StreamlitAPI Screenshot 50
5.3 Data Analysis StreamlitAPI Screenshot 50

7.1 Performance of the KNN model on the UNSW-NB15 dataset for Binary Classification 58
7.2 Performance of the Random Forest model on the UNSW-NB15 dataset for Binary Classification 58
7.3 Performance of the Decision Tree model on the UNSW-NB15 dataset for Binary Classification 59
7.4 Performance of the MLP Classifier Model on the UNSW-NB15 dataset for Binary Classification 59
7.5 Performance of the KNN model on the UNSW-NB15 multi-class dataset. . 60
7.6 Performance of the Random Forest model on the UNSW-NB15 multi-class dataset. 61
7.7 Performance of the Decision Tree model on the UNSW-NB15 multi-class dataset. 61
7.8 Performance of the MLP Classifier on the UNSW-NB15 multi-class dataset. 62
7.9 Confusion Matrix of the Ensembled Model for Binary Classification 63
7.10 Confusion Matrix of the Ensembled Model for Multiclass Classification . . . 64
7.11 Streamlit's Frontend Tabs 66
7.12 Caption 67

7.13 Boxplots of variables for Label 0 - Part 1 68
7.14 Boxplots of variables for Label 0 - Part 2 68
7.15 Variable Distributions for Label 0 - Part 1 69
7.16 Variable Distributions for Label 0 - Part 2 69
7.17 Boxplots of features for Label 1 - Part 1 70
7.18 Boxplots of features for Label 1 - Part 2 71
7.19 Features Distributions for Label 1 - Part 1 71
7.20 Features Distributions for Label 1 - Part 2 72
7.21 Boxplot Normal Traffic-Binary Classification 74
7.22 Distribution of features for Normal Traffic 74
7.23 Boxplot DoS Attack 75
7.24 Distribution of features for DoS Attack 76
7.25 Boxplot Scanning Attack 76
7.26 Distribution of features for Scanning Traffic 77

A.1 Label Correlation for Binary Classification using Min-Max-Score 123
A.2 Logistic Regression Evaluation 123
A.3 K-Nearest Neighbors Evaluation 124
A.4 Decision Tree Evaluation 124
A.5 Random Forest Evaluation 125
A.6 MLP Classifier Evaluation 125
A.7 SGD Classifier Evaluation 126
A.8 Ensembled Model Evaluation 126
A.9 Label Correlation for Binary Classification using Z-Score 126
A.10 Logistic Regression Evaluation (Z-Score) 127
A.11 K-Nearest Neighbors Evaluation (Z-Score) 127
A.12 Decision Tree Evaluation (Z-Score) 128
A.13 Random Forest Evaluation (Z-Score) 128
A.14 MLP Classifier Evaluation (Z-Score) 129
A.15 SGD Classifier Evaluation (Z-Score) 129
A.16 Ensembled Model Evaluation (Z-Score) 130
A.17 Label Correlation for Multi-Class Classification using Min-Max-Score 130
A.18 Logistic Regression Evaluation (Multi-Class) 130
A.19 K-Nearest Neighbors Evaluation (Multi-Class) 131
A.20 Decision Tree Evaluation (Multi-Class) 131
A.21 Random Forest Evaluation (Multi-Class) 132
A.22 MLP Classifier Evaluation (Multi-Class) 132
A.23 SGD Classifier Evaluation (Multi-Class) 133
A.24 Ensembled Model Evaluation (Multi-Class) 133
A.25 Label Correlation for Multi-Class Classification using Min-Max-Score 134
A.26 Logistic Regression Evaluation (Multi-Class) 134
A.27 K-Nearest Neighbors Evaluation (Multi-Class) 135
A.28 Decision Tree Evaluation (Multi-Class) 135
A.29 Random Forest Evaluation (Multi-Class) 136
A.30 MLP Classifier Evaluation (Multi-Class) 136
A.31 SGD Classifier Evaluation (Multi-Class) 137
A.32 Ensembled Model Evaluation (Multi-Class) 137

List of Tables

3.1 Summary of Dataset Statistics Before and After Cleaning 34

4.1 Model Performance using Min-Max and Z-Score Normalization on UNSW Binary Classification Dataset 41
4.2 Model Performance using Min-Max and Z-Score Normalization on CTU-13 Binary Classification Dataset 41
4.3 Model Performance using Min-Max and Z-Score Normalization on CTU-UNSW Binary Classification Dataset 42
4.4 EModel Performance using Min-Max and Z-Score Normalization on UNSW Dataset for Multi-Class Classification 42

5.1 Features and types of variables for binary and multiclass classification models. 48

7.1 Feature Value Ranges by Traffic Type - Binary Classification 73
7.2 Feature Value Ranges by Attack Type 78

List of Abbreviations

IoT	**I**nternet **o**f **T**hings
WSF	**W**hat (it) **S**tands **F**or
API	**A**pplication **P**rogramming **I**nterface
ML	**M**achine **L**earning
AI	**A**rtificial **I**ntelligence
IDS	**I**ntrusion **D**etection **S**ystem
SGD	**S**tochastic **G**radient **D**escent
DoS	**D**enial **o**f **S**ervice
DDoS	**D**istributed **D**enial **o**f **S**ervice
UDP	**U**ser **D**atagram **P**rotocol
TCP	**T**ransmission **C**ontrol **P**rotocol
XAI	**E**xplainable **AI**
CNN	**C**onvolutional **N**eural **N**etwork
RNN	**R**ecurrent **N**eural **N**etwork
PCA	**P**rincipal **C**omponent **A**nalysis
LDA	**L**inear **D**iscriminant **A**nalysis
SSL	**S**ecure **S**ockets **L**ayer
TLS	**T**ransport **L**ayer **S**ecurity
AES	**A**dvanced **E**ncryption **S**tandard
RSA	**R**ivest–**S**hamir–**A**dleman (cryptographic algorithm)

List of Symbols

a	Distance	m
P	Power	W ($\mathrm{J\,s^{-1}}$)
X	Original Value of the Variable	
X_{min}	Minimum Value of the Variable	
X_{max}	Maximum Value of the Variable	
Z	Variable value after Z-Score Normalization	
μ	Mean of the Variable	
σ	Standard Deviation of the Variable	
ω	Angular Frequency	$\mathrm{rad\,s^{-1}}$

List of Acronyms

AES	Advanced Encryption Standard.
AI	Artificial Intelligence.
API	Application Programming Interface.
CIA	Confidentiality, Integrity, Availability.
CMSIS-NN	Cortex Microcontroller Software Interface Standard Neural Network.
CNN	Convolutional Neural Network.
CTU-13	Czech Technical University - 13 Dataset.
CTU-UNSW	Combined Dataset from CTU and UNSW.
DDoS	Distributed Denial of Service.
DNS	Domain Name System.
DoS	Denial of Service.
DRF	Django Rest Framework.
DSN	Decimal Scaling Normalization.
FN	False Negative.
FP	False Positives.
GPS	Global Positioning System.
IDS	Intrusion Detection System.
IoT	Internet of Things.
KNN	K-Nearest Neighbors.
LDA	Linear Discriminant Analysis.
MAD	Median Absolute Deviation.
MAE	Mean Absolute Error.
MiTM	Man-in-the-Middle.
ML	Machine Learning.
MLP	Multilayer Perceptron.
MMADN	Median and Median Absolute Deviation Normalization.
MSE	Mean Squared Error.
NGFW	Next Generation Firewall.
NLP	Natural Language Processing.

PCA	Principal Component Analysis.
R^2 Score	Coefficient of Determination.
RMSE	Root Mean Squared Error.
RNN	Recurrent Neural Network.
RPL	Routing Protocol for Low-Power and Lossy Networks.
RSA	Rivest–Shamir–Adleman (cryptographic algorithm).
RTT	Round Trip Time.
SGD	Stochastic Gradient Descent.
SSL	Secure Sockets Layer.
SVM	Support Vector Machine.
TCP	Transmission Control Protocol.
TLS	Transport Layer Security.
TP	True Positives.
TTL	Time to Live.
UDP	User Datagram Protocol.
UNSW	University of New South Wales - NB15 Dataset.
USD	United States Dollar.
Wi-Fi	Wireless Fidelity.
WPA3	Wi-Fi Protected Access 3.
XAI	Explainable AI.

Chapter 1

Introduction

In an era characterised by an enormous dependency on technologies, the intersection of connectivity and comfort has given birth to new challenges, particularly in the world of cybersecurity. With the integration of new technologies, new systems and new devices into our daily lives, new risks in the cyber-attack scenario will appear. The emergence of Internet of Things (IoT) and Artificial Intelligence (AI) in an interconnected way has given rise to a new era of innovation, but with many new vulnerabilities.

The ongoing increase in these vulnerabilities highlights the need to implement more robust cybersecurity measures. The impact of a cybersecurity attack extends to all sectors, from healthcare to smart homes, putting the safety of those who are attacked at risk.

1.1 Contextualisation

In an era dominated by technology, the sudden addition of the Internet of Things (IoT) has changed the residential scene, turning conventional homes into dynamic ecosystems known as "smart homes". With this paradigm shift, various IoT devices have been added, creating a more comfortable, intelligent and efficient space.

On the other hand, with the evolution of residences to more technologically advanced environments, it is becoming crucial to address some concerns. One of the main concerns involves cybersecurity. With this in mind, it is important to understand the interaction between the technological innovation present in smart homes and the vulnerabilities in their devices and ecosystem.

Growing concerns about cybersecurity in residential contexts

The rise of smart homes brought in a new era of efficiency and comfort, but it has additionally presented a number of cybersecurity challenges. Research by *Kaspersky* shows that attacks on IoT devices doubled last year, with more than 1.5 billion attacks recorded in the first six months of 2021 alone, compared to 639 million in the previous six months [1]. This concerning pattern draws attention to the increasing risks that come with more connected devices in homes.

The number of IoT devices in smart homes has rapidly grown as a result, increasing the vulnerabilities connected to these innovations in technology. The rate at which IoT devices are growing is highlighted by the estimate of 127 new devices connected every second by *McKinsey* analysts [1]. An IoT device's security needs to be improved since it could be a point of entry for cyberattacks.

The fast growth of IoT devices connected to smart homes presents issues for the network that telecommunications service providers manage. IoT connections are predicted to increase significantly from 10.3 billion in 2018 to 25 billion globally by 2025, according to GSMA Intelligence [2]. The growing number of IoT devices requires more efficient data management from providers. The huge volume of data generated by these devices impacts the network infrastructure that is now in place, particularly since 5G technologies are required to ease communication between billions of sensors and devices.

Therefore, if data is managed less efficiently, it can not only affect the home environment but also the providers' operational performance.

To overcome these challenges, it's necessary to develop mitigation strategies, such as implementing measures to minimise the impact of cyberthreats, guaranteeing the stability of the network, or advanced intrusion detection systems with with the ability to identify more suspicious patterns, possibly with malicious purpose. By adding automatic learning algorithms to these measures, the system will be also able to adapt to the evolution of attacks, and is considered a fully dynamic system.

Therefore, with these growing concerns about cybersecurity, the vulnerabilities of IoT devices must be considered.At the same time, it is important to implement strong solutions so as not to risk the privacy of users or the integrity of devices.

1.2 Problem Statement

With the topic of smart homes and IoT devices, their integration causes a number of cybersecurity challenges to attract greater attention. This topic is paired with the lack of standardised security mechanisms, which makes smart homes prone to various cybersecurity threats, underlining the need to deal with various inherent vulnerabilities.

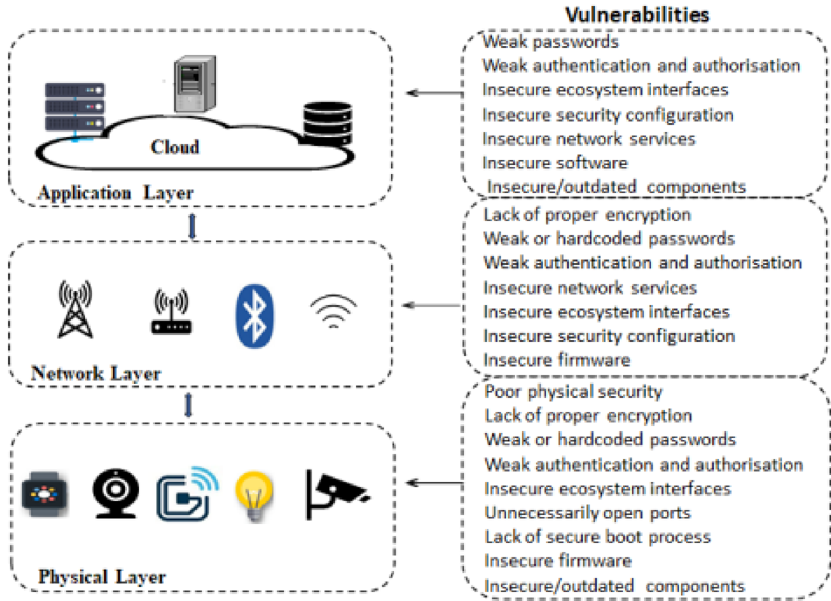


Figure 1.1: IoT Vulnerabilities linking with layered IoT architecture across several layers.

The multi-layered architecture of smart homes—comprising the physical, network, and application layers—introduces specific vulnerabilities at each level, as illustrated in Figure 1.1. These vulnerabilities include weak authentication, insecure configurations, and outdated components, all of which require robust security measures to mitigate.

The main problem that this thesis aims to solve is the lack of effective monitoring and user-protection against the threats of cybersecurity in IoT ecosystems. It is important to tighten security in smart homes, which include a number of devices that may be operating without strong security measures. These devices may have various vulnerabilities such as Distributed Denial of Service (DDoS) attacks or even software vulnerabilities, emphasising the need for strict and extensive monitoring.

With IoT devices becoming a key part of everyday life, the importance of protecting both users and devices has grown. The lack of standardised security protocols represents a risk to the three basic pillars of cybersecurity, the confidentiality, integrity and availability of user data in smart homes.

By focusing on the multi-layered nature of cyberthreats in IoT ecosystems, this thesis aims to contribute not only to understanding the problem, but also to the practical implementation of security measures.

1.3 Motivation

The main motivation behind this thesis comes from the need to protect users of smart home IoT devices from the increasing "cyberthreats". As such, there is an ongoing need to strengthen more conventional cybersecurity approaches. The motivation for this is based on a number of very important aspects.

Building a Future of Safer Smart Homes

The main motivation is directly related to the vision of a safer future in smarthomes. By improving the security of residential routers, there will be an environment in which all users will enjoy the benefits of IoT technologies without putting their security and privacy at risk.

The Emergence of Smart Homes

As mentioned above, the emergence of smart homes represents a revolutionary transformation in the way users interact with their living spaces. However, this change introduces an increase in cybersecurity challenges, requiring a more proactive and adaptable approach to security.

Vulnerabilities in Home Routers

The most traditional approaches to cybersecurity related to smart homes usually do not have the capacity to deal with today's cyberthreats. The motivation to improve conventional approaches arises from this, as routers act as entry points into the home network of IoT appliances [3].

Importance of User Protection

With the increasing number of cyberthreats in the smart home ecosystem, it is essential to protect users. As a result, there is a need to strengthen cybersecurity measures so as not to compromise their privacy and the integrity of their data.

Security Gaps

It is clear that conventional approaches need to be improved in order to effectively counter less common and more complex cyberthreats, such as DDoS attacks, identity theft attacks, or even malware and software vulnerabilities. In view of the weaknesses identified, the motivation to overcome all these flaws and evolve cybersecurity at a residential level has emerged.

1.4 Thesis objectives

This thesis aims to achieve a series of clearly defined objectives, each contributing to the overall goal of improving cybersecurity in smart homes.

1. Perform an extensive analysis of the most common cybersecurity threats in smart homes.
2. Investigate and identify vulnerabilities in IoT devices and home routers to understand potential entry points for cyber threats.
3. Investigate, propose and analyze new cybersecurity solutions aimed at strengthening smart home environments.
4. Investigate the evolution and impact of Artificial Intelligence (AI) and Machine Learning (ML) in Contemporary Applications.

These objectives collectively aim to comprehend the cybersecurity challenges in smart homes and contribute practical solutions to enhance security in these dynamic environments. The realization of these objectives is anticipated to lead to a significant improvement in the security of homes.

1.5 Document Structure

The document's introduction (1) describes the goals of the thesis, provides a brief explanation of the subjects covered, and sets up the document's structure.

The rise of the IoT in daily life and related security issues are addressed in more detail in the chapter 2, which also looks into the basic ideas of machine learning.

The machine learning methods chosen to create an anomaly detection pipeline are discussed in Chapter 3, with a focus on accurate data pre-processing.

The training of individual and ensemble models is addressed in Chapter 4, along with a review of performance metrics in binary and multiclass classification.

The use of Streamlit and FastAPI to implement APIs that allow for real-time visualization and the backend integration of machine learning models is addressed in Chapter 5.

The test bed created to automate cyberattacks is reviewed in Chapter 6 along with the relevant features that were extracted for the models' training.

Chapter 7 shows results from the performance evaluations and comparisons between the individual models and the ensemble, highlighting how well the ensemble detects anomalies.

The project's conclusions and contributions are finally summed up in Chapter 8 , which includes suggestions for future research directions.

Chapter 2

State of the Art

The integration of the IoT into daily life, along with cybersecurity challenges, requires the creation of proactive defence mechanisms. Machine learning, especially in intrusion detection systems, has become a key element in detecting threats in real time. The State of the Art explores the challenges of implementing ML in resource-constrained environments, emphasising the balance between model complexity and energy efficiency. It also addresses ML libraries in low-level languages and the interoperability of ML models.

2.1 IoT and Cybersecurity

The value of the home automation market increased from 56.15 billion United States Dollar (USD) in 2022 to an expected 207.23 billion USD by 2032, indicating the fast growth of IoT devices within smart homes [4].

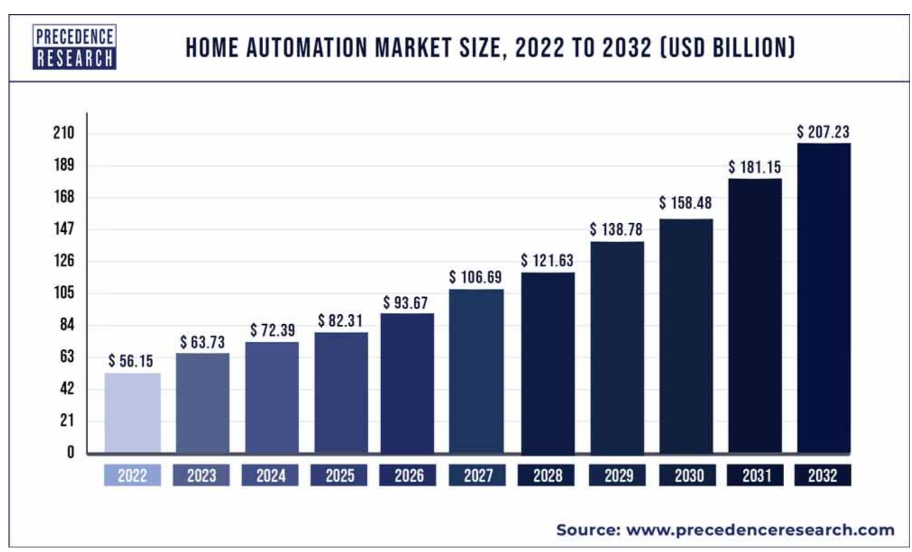


Figure 2.1: Home Automation Market Size, 2022 to 2032 Prediction (In USD).

The word **cyber** refers to everything having to do with computers, information technology, and the online world of the internet. It comes from the word **cybernetic**, consisting of methods, tools, and procedures designed to defend computers, programs, networks, and data against various types of attacks. It is frequently used as a prefix to refer to ideas,

procedures, and activities related to digital environments, networks, and virtual interactions in words like cyberspace, cybersecurity, cybercrime, and cybernetics [5].

With a basic understanding of the interaction between IoT and Cyber, it becomes important to be aware of not only the fundamental concepts of cybersecurity, but also the practical measures that should be taken to protect the environment. By analysing the types of cyber-threats and the impact that these attacks can have, it is easier to see how important cybersecurity is in the IoT environment and, above all, how it can protect users.

To demonstrate the impact of cybersecurity in the context of IoT, we can consider some examples of attacks, such as:

- **Smart Home Attacks** When someone maliciously gains unauthorised access to the smart home system, gaining full control over the ecosystem and IoT devices, such as controlling lights, temperature and even security cameras. As well as compromising homeowners' privacy, this puts physical security at risk.
- **Industrial Attacks** In an industrial context, a cyber-attack can steal sensitive data from IoT devices, or even disrupt vital company operations.
- **Healthcare Attacks** IoT devices that are in a healthcare context, such as pacemakers, can be targeted to access confidential patient information, risking serious privacy and health risks.

With these examples, it's possible to realise that a very solid cybersecurity strategy is needed to guarantee the safety of all.

2.1.1 Fundamental Concepts in Cybersecurity

Users use various IoT devices in their daily lives, from smart home appliances to health and sports devices. With this ongoing use, it is essential to guarantee the security of devices from potential threats. Therefore, I will go through the basic principles of cybersecurity in order to gain a better understanding of the challenges and goals that must be achieved in order to improve the security of the IoT device, but above all of the user.

IoT devices are developed with the objective of performing just one simple task, often prioritising energy efficiency.On the other hand, this approach creates a huge problem, because the main limitations of IoT devices comes from their development, often having very limited processing power and memory. Additionally, these devices have been designed to be used for a very long time, usually for dozens of year. These limitations influence their cybersecurity capabilities.

There have been several recent incidents of security cameras being hacked in smart homes. A passive vulnerability assessment was executed using tools such as Shodan and Nessus. This assessment found that thousands of IoT cameras had some kind of vulnerability, from weak authentication to weakly secured web interfaces. The study proved that even people with limited knowledge of offensive cybersecurity could exploit these vulnerabilities and compromise users' privacy and security [6].

On the other hand, limited processing power directly restricts the complexity of security algorithms that can be run. As a result, not having advanced encryption protocols or other types of systems will make the equipment more vulnerable to attacks.

The limited memory of IoT devices also creates a new problem, as it becomes quite complicated to manage the device's storage space knowing that there are regular security updates that need to be installed and many of these systems do not support updates.

In addition to the complexity of performing regular updates, with the limited memory space available, certain features that should be implemented on all devices are no longer implemented, or are implemented in a very simple way, such as authentication, making it possible for any user to access the IoT device.

The low processing capacity and limited memory present cybersecurity challenges that are different from the most typical challenges. Given these limitations of IoT devices, new mechanisms must be implemented so that user security is guaranteed and the functionality of IoT devices is not compromised [7].

Confidentiality, Integrity, Availability (CIA)

Three basic pillars are the foundation of cybersecurity: Confidentiality, Integrity, Availability (CIA). A scenario is considered to have a solid cybersecurity the basis if it complies with each of these three pillars.

The objective of the **confidentiality** principle is to prevent unauthorized access to sensitive information.Confidentiality in the context of smart homes is ensured by not exposing user preferences, personal information, or system settings.

The accuracy, consistency, and reliability of data can be referred to as **integrity**.As it applies to IoT devices, guaranteeing integrity means making sure that there is no external interference or tampering during communication.

On the other hand, **availability** defines the ability to access information and systems at any moment.In this environment, ongoing availability is critical to the IoT devices' ability to operate effectively [8].

Below is an illustration 2.2 of the CIA triad, highlighting its fundamental principles in cybersecurity [9].

Figure 2.2: CIA Triad

Encryption is an essential part of data protection, especially in connected situations like smart homes.Through this process, information that can be read is converted into formats that are indecipherable leaving it unreadable by anyone without decryption keys. Data must be encrypted for storage and for the security of data transfer between devices. This ensures

that data is safe even in the case of unauthorized access. We shall go into more depth on the role of encryption in smart home data security below.

Encryption

Once the three pillars of cybersecurity are well established, it becomes clear what practical measures are needed to guarantee the principles. A very important measure that guarantees the 3 pillars is encryption. Encryption serves as a vital tool to ensure that the CIA triad is not just a concept, but a practical implementation.

Encryption plays a fundamental role in improving smart home data security. It involves transforming easily recognisable data into totally unrecognisable data, which can only be deciphered by those who have access to decryption keys. In the context of smart homes, encryption has 2 main functions:

1. Security of data in transit, since communication between IoT devices and central systems is done over encrypted channels. By using data transmission protocols such as Secure Sockets Layer (SSL) and Transport Layer Security (TLS), unauthorised third parties are prevented from accessing sensitive information, guaranteeing confidentiality and integrity in data transmission [10].
2. Data already stored, because in the chance of a device being stolen, or unauthorised access to the storage, the data can only be decrypted by those who have the key, keeping the data safe at all times. The two most commonly used algorithms in this area are Advanced Encryption Standard (AES) and Rivest–Shamir–Adleman (cryptographic algorithm) (RSA) [11].

After a theoretical analysis of encryption and a perception of its practical importance in data security in smart homes, another fundamental aspect of cybersecurity is covered: **firewalls and Intrusion Detection System (IDS)**. These systems act as the first line of defence in multilayered security systems, providing not only fully authorised data traffic but also automatic detection of any suspicious/malicious activity.

Firewalls create a virtual barrier within the smart home network, segmenting different devices and services. Another function is to reinforce access policies, by limiting the devices that can communicate within the home network. These 2 measures prevent unauthorised access and data breaches.

Firewall technologies have evolved significantly since their creation in order to increase network security. Firewalls used to focus on static rule-sets to filter network traffic. Today they have a more dynamic security framework, which includes advanced features such as **deep packet inspection** and even **machine learning algorithms for threat detection**. Next Generation Firewall (NGFW)s fit the smart home system perfectly, where there is a wide range of interconnected devices, offering a layered defence mechanism that can identify and mitigate different types of threats, even unknown ones.

In addition, the appearance of **cloud-based firewall solutions** offers a more scalable and adaptable approach for smart homes. These firewalls take advantage of the cloud's computing resources to provide layered protection.

The combination of NGFWs and **cloud-based firewalls** offers a very strong security posture for smart homes, guaranteeing above all, the integrity of users [12].

On the other hand, IDS monitors network traffic and the behaviour of the various devices on the network, enabling the identification of suspicious activity patterns. Another feature is **Signature-Based Detection**, in which there is a kind of catalogue of the most common attack patterns. This way it's possible to mitigate such attacks [13].

The growth of the IoT, which connects a wide range of devices from household appliances to industrial sensors, has dramatically expanded the size and complexity of networks. Because of this expansion, more advanced IDS with the ability to adapt to cyberthreats, in order to target these networked environments, have had to be developed.

In the dynamic IoT environment, traditional intrusion detection systems—which frequently depend on predetermined rules or signatures to identify possible threats may not always be effective. It might be difficult for these systems to identify new or advanced threats that defy their current restrictions.

Advanced IDS models, such as those derived on the biological concept of the artificial immune system, have been created for IoT networks in order to tackle these issues. These systems imitate how the immune system of humans adapts to identify and combat infections. In a similar vein, an artificial immune system gathers new knowledge from the network environment it guards, evolving over time to identify and defend against new, yet undiscovered threats [14].

Algorithms that examine typical network behaviour and can identify anomalies suggestive of possible security risks enable this learning capability. By constant study of what defines "normal" for a certain IoT network, the artificial immune IDS is able to recognise anomalies that may indicate an attack—even if such an attack has never been detected before.

In addition, the effectiveness of the IDS's protection of IoT environments is improved when it combines this self-learning capability with conventional detection techniques. It provides a strong answer for protecting the IoT environment by enabling a more dynamic defence mechanism that changes in accordance with the threats it encounters.

In this way, various types of attacks such as Denial of Service (DoS) and Routing Protocol for Low-Power and Lossy Networks (RPL) topology attacks are effectively dealt using the same mechanism [15].

The combination of Encryption, Firewalls and IDS turns any common system into a defensively secure system, capable of mitigating the most common risks associated with unauthorised access, data interception and even malicious attacks on smart homes. This will ensure compliance with the 3 pillars of cybersecurity.

2.1.2 Existing Approaches to Residential Cybersecurity

As analysed so far, with the inclusion of IoT devices in smart homes, a number of problems have been identified. In addition, as the devices take an important role in our daily lives, it becomes essential to guarantee security. As such, various approaches and strategies for protecting devices and, consequently, the user, will be explored. By implementing all the strategies suggested, a system will be much better protected against cyberthreats, making smart homes not only safer, but also more efficient.

Network Segmentation

In the context of residential cybersecurity, where the connectivity of IoT devices comes with associated risks, the concept of network segmentation has become a critically important strategy. The smart home will be segmented into different zones, each with a different purpose, with the main objective being to provide extra security to different categories of devices located in different zones, containing possible threats and consequently limiting their impact.

The basic principles of network segmentation are the isolation of device types, as each type has unique security-related details, preventing lateral movement of threats across the smart home network. Another aspect is controlled access, as with restricted communication between segments of the home, attempts at unauthorised access or compromised devices in one segment have less risk of affecting other segments. [16]

There are also a number of challenges related to the implementation and use of segmented networks.

One of these challenges is finding the perfect balance between usability and security. With the enhanced security through segmentation, it can be quite challenging when it comes to user incovinience, and can even discourage users from taking recommended security measures. Another challenge comes with the addition of new IoT devices, which leads to the evolution of the smart home ecosystem. Therefore, network segmentation must be dynamic and flexible in order to adapt to changes in its environment without compromising security. The complexity of implementing and configuring network segmentation can also become a challenge, because effective implementation requires careful planning and configuration.

Network segmentation can evolve the home cybersecurity landscape, offering various strategies to mitigate the risks associated with IoT devices [17].

In addition to the previously mentioned aspects of network segmentation, it is crucial to address the security of wireless networks in smart home environments. In Wireless Fidelity (Wi-Fi) networks, for example, all data traffic is potentially open to interceptions if proper security measures are not implemented. This is especially true in home networks where IoT devices, often with security vulnerabilities, are constantly communicating information that is potentially sensitive.

Implementing network segmentation in wireless environments should therefore include strict security practices, such as using strong encryption,like Wi-Fi Protected Access 3 (WPA3), to protect data traffic. In addition, the creation of a separate Wi-Fi network exclusively for IoT devices should be explored. This dedicated IoT network can even limit exposure to attacks by isolating less secure devices from direct access to the main home network and the most critical devices, such as personal computers and storage devices [18].

This specific segmentation strategy for wireless networks not only protects against the interception of data traffic, but also mitigates the risk of wider network attacks. If an IoT device on a segmented wireless network is compromised, the potential damage is limited to that subnetwork, significantly reducing the risk to the main home network and other connected devices.

When considering network segmentation in smart home environments, it is essential to take wireless network security into consideration [18].

2.1.3 Types of Attacks

Given the diverse uses and ease of access of IoT devices, together with the security systems that are essential, the number of malicious attacks that threaten one or more of the fundamental pillars of cybersecurity has increased over the years.

With technological progress, it is only logical that there should be an increasing number of cyberattacks. An analysis of data covering more than a decade has revealed a significant increase in the number of cyberattacks, highlighting the importance of maintaining a solid defence and a preventive approach.

Over the last twelve years, the statistics show an increasing global trend, with noticeable peaks in certain years. From 2019 to 2021 there was a substantial increase due to the COVID-19 pandemic. The annual increase in cyberattacks reflects the adaptability of malicious attackers to exploit new vulnerabilities in various fields [19].

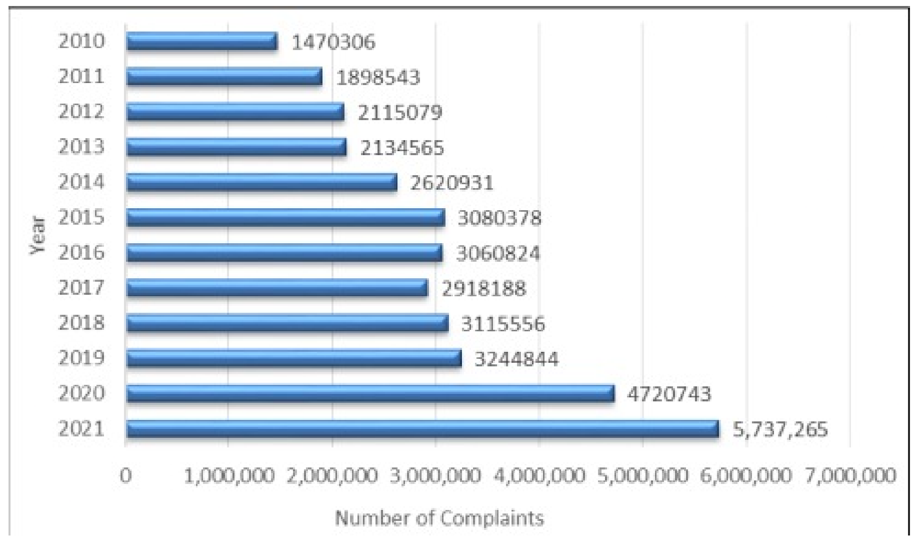


Figure 2.3: Cyber Attacks Over the years.

There are various types of attacks that can be performed on IoT devices, and they are categorised. A list showed by Geneva School of Economics and Management shows the various types of attacks, which are categorised as follows [20].:

Physical Attacks

- Object Replication Attacks: Creating replicas of devices in order to gain unauthorised access to IoT networks.
- Malicious Code Injection: Inserting malicious code into IoT devices in order to create vulnerabilities.
- Tampering with Hardware Components: Altering physical components of IoT devices in order to disturb the normal operation of the device.
- Side-Channel Attacks: Given the public information known about the IoT device, attackers exploit vectors such as power consumption, sound or even steal data.

Protocol-based Attacks

- Sniffing Attack: Intercept and record wireless network traffic, capturing information such as passwords or inter-device communication protocols.
- Transmission Control Protocol (TCP)-User Datagram Protocol (UDP) Port Scan: Scan the IoT device to identify Open port and available services for later attacks.
- Domain Name System (DNS) Spoofing: DNS settings are changed to redirect traffic to malicious services.
- Packet Injection: Injecting malicious packets into IoT network communication.

Data at Rest-based Attacks

- Data Exposure: Gaining unauthorised access to stored data on any IoT device.
- Brute-Force Attack: This method works on a trial/error logic, with the attacker trying combinations of usernames and passwords, gaining access to the IoT device.
- Unauthorised Data Access: The goal of this type of attack is to gain access to data without permission, either by exploiting vulnerabilities or through stolen credentials.
- Data Manipulation: The attacker changes the date stored on IoT devices in order to corrupt IoT operations that use that date.

IoT Software-based Attacks

- Distributed Denial of Service (DDoS): Involves overflowing the IoT network with traffic from multiple sources in order to cause a denial of service to real users.
- Phishing Attack: Deceiving IoT device users into revealing confidential information, such as passwords.
- Firmware Exploitation: Exploits IoT device firmware vulnerabilities.

With the increased attacks mentioned previously, an extraordinary financial loss is mirrored in the same way, requiring a change in proactive cybersecurity strategies.

In 2021, global financial losses due to cybersecurity incidents reached a total of 6 trillion USD. These losses are not only financial, but also affect the global reputation of the organisations involved in the incidents [19].

The prevention strategy must include stronger network security, advanced threat detection and a culture of population awareness. Statistics emphasise that the cost of prevention is insignificant when compared to the price paid following well-executed cyber-attacks.

So, the rising financial cost of cybercrime highlights the importance of giving priority to prevention.

Figure 2.4: Amount of Losses of Cyberattacks last twelve years (2010-2021), U.S.A.

2.2 Machine Learning (ML) and Artificial Intelligence (AI)

The evolution of Artificial Intelligence and Machine Learning is a journey back to the mid-20th century, with the appearance of AI as a formal subject with the purpose of building machines capable of emulating certain cognitive functions.

The birth of AI can be traced back to the Dartmouth Conference in 1956, where pioneers in the field met to set the foundations for this new area. With the initial hype, several ambitious projects appeared, such as the Global Positioning System (GPS), which aimed to create a universal machine with the ability to solve problems. However, with very limited computing power, a phase soon began known as the "AI winter", marked by a drop of interest, and subsequently of the resources allocated.

AI re-emerged in the 1980s, driven by advances in computing capabilities and a shift towards a new type of system, based on rules designed to emulate human behaviour. However, the limitations of such systems in dealing with uncertainty and adapting to new information became obvious[21].

As for ML, its origins go back to the 1940s and 1950s, with a game of checkers that demonstrated the potential of machines to learn from experience[22].

Figure 2.5: First AI Game of Checkers.

Towards the end of the 20th century, the integration of statistical approaches into ML, together with the availability of a larger dataset, led to significant advances. The appearance of neural networks, inspired by the structure of the human brain, gave birth to the domain of Deep Learning. With greater computing power, more sophisticated algorithms were created.

Today, with AI and ML incorporated into different contexts of our daily lives, from virtual assistants and voice recognition, to online content recommendation and autonomous vehicles, making the user experience more personalised and efficient than ever before, pushing forward a number of areas [23].

2.2.1 Recent Advances and Future Trends in AI and ML

Machine Learning is a subfield of Artificial Intelligence that focuses on developing algorithms and models capable of learning patterns from data. The main goal of ML is to allow computers to gradually become more proficient at a given task without having to be specifically programmed for it. This is accomplished by employing a number of strategies that enable algorithms to identify trends, anticipate outcomes, and modify their behaviour in response to the input data.

Recently, the areas of AI and ML have had a revival, as discussed above, mainly due to the appearance of Deep Learning.

Deep Learning has shown exceptional capabilities in various tasks, such as image recognition, Natural Language Processing (NLP) and speech recognition. Convolutional Neural Network (CNN) have featured heavily in image-related applications,as can be seen, for example,in object detection in autonomous vehicles, while Recurrent Neural Network (RNN) and Transformer models have made progress in the area of NLP. With all these developments, it has been possible to outperform humans in various complex tasks [24].

In addition, with the integration of reinforcement learning, it has been possible for machines to learn new decision-making strategies by interacting with dynamic environments. It is used in various areas, from robotics and autonomous systems to strategy games, proving the adaptability of algorithms.

With this analysis of history, it is expected that there will be an evolution in the path of AI and ML. Explainable AI (XAI) is growing in popularity as a critical consideration, highlighting the need for models to provide transparent information about the processes behind decision-making.

Another area that also focuses on the learning and adaptability of models is Continual Learning. With the addition of new data, scenarios and tasks, this field is becoming increasingly important and is already being used in real-world contexts, allowing systems to evolve over time while remaining effective and relevant [25].

2.2.2 Types of ML Algorithms:

ML algorithms can be classified into three main types, supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, the algorithms are trained by labelling, always providing input-output pairs. This allows the model to learn to map the inputs to the respective outputs. This kind of model is normally used for classification and regression tasks. An example of using this model is an email spam recognition system using supervised learning, where the algorithms are trained with labelled emails associated with

categories and can automatically identify and classify emails based on the features learned [26].

In contrast, unsupervised learning deals with unlabelled data, and the main objective of the algorithms is to discover patterns in the data. The data is grouped based on its similarities, which makes the data simpler while maintaining its original traits. An example of the use of this kind of model is a movie recommendation system, well known on the internet, in which algorithms analyse user behaviour, identifying patterns and similarities, and can suggest movies based on users' preferences[27].

Reinforcement learning involves algorithms interacting with an environment to make decisions. The algorithm has an associated feedback system, receiving either rewards or punishments depending on its actions, so it is guided towards making more optimal decisions over time. This type of learning is common in the world of gaming and robotics.

Each type of learning algorithm has its weaknesses and strengths, and the tasks to be performed must be studied in order to identify and select the perfect algorithm[28].

2.2.3 Machine Learning for Intrusion Detection System (IDS) in Embedded Systems

ML has become vital for improving the capabilities of intrusion systems in embedded systems and, with its integration, it represents a more proactive approach to dealing with growing cyberthreats.

In the world of embedded systems, where there are various resource constraints and real-time processing is crucial, traditional Intrusion Detection System (IDS) systems suffer from many limitations. With the integration of ML algorithms, IDSs can learn and adapt to new attack patterns, providing a more dynamic defence. Adaptability in this type of system is critical, as predefined rules may not properly cover the variety of today's cyberthreats.

One advantage of using supervised learning models in IDS systems is the ability to distinguish patterns and anomalies in the system's behaviour. By training the models with historical data, the system recognises and classifies both ordinary and malicious activity.

On the other hand, unsupervised learning models are relevant in detecting new attacks, as they can identify any deviations from the established behavioural database [29].

Another advantage of including ML models in IDS systems is the reduction in false positives. Traditional IDS can have multiple false positives, which results in unnecessary consumption of resources and, therefore, the failure to detect real threats. With the continuous learning of ML models in IDS systems, the ability to distinguish normal from dangerous activity is constantly being improved, contributing to better accuracy and efficiency

Some ML models, such as lightweight models or line learning, align perfectly with the real-time execution needs of embedded systems, allowing the IDS to function efficiently, despite resource constraints [30].

As cyberthreats continue on a path of constant evolution, adding ML to IDS for embedded systems becomes a proactive strategy, strengthening defensive cyber security. This interaction improves threat detection capabilities and enables a faster response to possible threats.

Challenges and Constraints of Embedded Systems

Implementing ML on devices with limited resources presents a number of challenges.

One of the major challenges is the trade-off between model complexity and resource constraints, since more complex models offer better performance but require more computing power and memory. The challenge is to find the perfect balance to develop models that are effective and suitable for implementation on devices with limited resources.

Another challenge is the lack of labelled data for training models in some fields. A reliable model requires more complex training with a larger data set. On the other hand, labelling the data requires a lot of resources, which means using strategies such as transfer learning or the use of artificial data[31].

Another challenge is meeting the latency requirements needed for real time, because with heavier and more robust algorithms, these requirements will not be satisfied. So there needs to be a balance between the quantity and quality of the dataset[32].

Energy efficiency is also a challenge, since excessive energy consumption affects the longevity of the device. With the computational power demands of ML algorithms, sustainable implementation in embedded systems is crucial[33].

The security and strength of learning models is also a challenge, since devices can be more susceptible to attacks and the lack of resources makes it difficult to implement security measures.

Implementing ML on devices with limited resources requires a multidimensional approach that addresses the many challenges presented.

ML Libraries in Low Level Languages

Machine Learning libraries in low-level languages have become fundamental for increasing the range of ML models on resource-poor devices. Low-level languages such as C or C++ provide direct control over the device's hardware resources, which makes them ideal for embedded systems like Arduino and Raspberry Pi.

A library often used in low-level languages is TensorFlow, which has a C++ Application Programming Interface (API) that allows ML models to be directly integrated into applications.

Another well-known library is **Caffe**, implemented in C++ and built for high-performance computing. It is mostly used for computer vision tasks, as it is highly focussed on Convolutional Neural Network.

LibSVM, a C library for support vector machines, focuses on processing large-scale classification tasks and is considered a very viable option given its high efficiency with limited resources.

Regarding work on embedded systems, **Cortex Microcontroller Software Interface Standard Neural Network (CMSIS-NN)**, written in C, stands out. It has been adapted for ARM processors, providing optimised routines for neural network inference

The use of low-level languages in ML libraries aligns with the requirements of applications that demand efficiency, real-time processing, and optimal resource usage, commonly found in IoT devices[34].

2.3 Data Preparation and Preprocessing

2.3.1 The Importance of Preprocessing

Any ML pipeline must include data pre-processing, but it's especially important for complicated situations like anomaly detection in Internet of Things networks.The performance of the models is strongly impacted by the consistency and quality of the data, and proper pre-processing can significantly improve the outcomes of cyber-attack detection [35].

Data Cleaning:

During the data collection stage, it is common to deal with problems such as duplicate values, missing data, and outliers that can compromise the quality of the data set. According to a survey conducted by Forbes, up to 60% of data scientists' time is spent on cleaning, standardizing, and organizing data [36]. Meanwhile, knowledge workers spend up to 50% of their time dealing with obfuscated and incorrect data. Data cleaning aims to identify and solve these anomalies to prevent them from negatively influencing the model's results. Various techniques are used to do this, such as: [37].

Duplicate removal: Duplicate entries have the potential to distort the model and induce it to learn irrelevant patterns. These duplicates are removed to guarantee that every sample contributes in a unique way [38].

Handling missing values: A lot of datasets contain rows or columns with missing values, either because records weren't collected or because of errors occurring during data collection. These issues are fixed via row/column deletion techniques or data imputation, which ensures the dataset's integrity.

Removing invalid values: On some occasions, values for some features (such as negative values for features that should only contain positive values) may fall outside of the expected range. To maintain data consistency, these entries are identified and corrected.

Combining Datasets: Detecting cyberattacks in IoT environments is an extensive subject, and it is usual to use different datasets to address the variety of attacks and scenarios. By extending the range of normal and malicious traffic included in the training set, combining datasets can improve the models' stability and ability to make predictions. However, there are some issues with this approach that need to be taken into account. These issues include attribute standardization, where different datasets may have different names for the same feature, and scenario balancing, which is required to make sure the dataset is balanced and addresses the various attack types [39].

2.3.2 Data Normalization

A fundamental pre-processing step is data normalization, especially if the data comes from different levels and sizes. The goal is to bring different attribute values into alignment so that MLalgorithms—which are susceptible to these variations—can function properly. Differences in data scaling can have an important impact on algorithms like neural networks and distance-based techniques like *K-Nearest Neighbors (KNN)* because, in the lack of normalization, features with larger magnitudes can prevail when calculating distances or updating weights in neural networks.

The outcomes of different strategies to data normalization and the two commonly used normalization algorithms (Min-Max Scaling and Z-Score, or Standardisation) on model performance will be covered.

Min-Max Scaling: Min-Max attribute's values are transformed via scaling into a specified range, often between 0 and 1. For algorithms that are sensitive to the size of the data, including neural networks and gradient-based approaches, this approach maintains the original proportions between the variables while maintaining the links between the values [40].

The formula for the Min-Max transformation is given by:

$$X_{\text{scaled}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

Where:

- X where X is the original value of the feature;
- X_{min} is the minimum value of the feature;
- X_{max} is the maximum value of the feature.

Z-Score (Standardization) Z-Score-based normalization, sometimes referred to as standardization, modifies the data to ensure each feature has a mean of zero and a standard deviation of one.This method works well with algorithms like logistic regression and Support Vector Machine (SVM)s that require a normal (Gaussian) distribution of the data [40].

The formula for Z-Score normalization is as follows:

$$Z = \frac{X - \mu}{\sigma}$$

Where:

- X is the original value of the feature;
- μ is the mean of the variable;
- σ is the variable's standard deviation.

Other Normalization Strategies

Depending on the context of the issue and the machine learning algorithm being used, additional strategies may be used in addition to Min-Max Scaling and Z-Score normalization.

Decimal Scaling Normalization (DSN): Depending on the magnitude of the values, this method shifts the decimal point to a particular number of places in order to normalize the data values. To normalise a feature, divide each value by its maximum absolute value and multiply the result by a power of 10. The following is the formula for normalizing a value x_i:

$$x_i' = \frac{x_i}{10^j}$$

where the smallest integer $\max(|x_i'|) < 1$ is represented by j.When dealing with data that shows high scale variability and it needs to keep the proportionate relationships between the initial values, this method can be useful [40].

Median and Median Absolute Deviation Normalization (MMADN): This technique is a reliable way to handle outliers since it normalizes the data using the median and Median Absolute Deviation (MAD)median absolute deviation. The following is the formula for normalizing a value x_i:

$$x_i' = \frac{x_i - \text{Median}(X)}{\text{MAD}(X)}$$

where Median(X) is the median of the set of data and MAD(X) is the absolute median deviation, calculated as the median of the absolute values of the differences between the data and the median:

$$\text{MAD}(X) = \text{Median}(|x_i - \text{Median}(X)|)$$

This method is efficient in situations with many outliers, as it is not sensitive to extreme values [40].

Comparing Min-Max and Z-Score

Both normalization approaches have advantages and disadvantages, and the choice between them depends on the context of the problem

Min-Max Scaling works best for neural networks that use activation functions like *sigmoid* or *tanh*, which benefit from restricted data in a fixed range, and is good when the data needs to be within a certain limit.

Z-Second Normalization is the best approach when there is a need to remove scale bias from the data, especially in algorithms like logistic regression or SVM that imply certain assumptions about the data's distribution [40].

2.4 Variable Encoding and Feature Selection

Variable encoding and feature selection are to guarantee that only the variables that are most relevant are included in the model training and that the data is in the right format for the algorithms to use. It is possible to convert qualitative data into numerical values by encoding categorical variables, and reducing dimensionality via feature selection can improve the model's interpretability and performance.

2.4.1 One-Hot Encoding

Binary variables are created by converting category variables using the One-Hot Encoding method.With this method, categorical attributes can be appropriately interpreted by Machine Learning algorithms, which can only handle numerical data, by converting each category into a new binary column (0 or 1) [41].

One-Hot Encoding has the benefit of maintaining category distinction without creating an unnatural numerical relationship, which could cause issues if the values were encoded as numbers directly. But when working with categorical variables that have numerous categories, this method can lead to a noticeable increase in the number of columns, which can affect the size of the dataset and increase the time it takes to train the model. The term

"dimensional explosion problem" refers to this event, in which the number of newly formed columns can significantly exceed the number of initial variables [41].

Also, techniques like Binary Encoding and Label Encoding can be applied to handle categorical values. However, Label Encoding gives each category an integer, which could point to an unrealistic order between the categories [42].

2.4.2 Feature Selection

Finding the features that are most relevant to the problem in question is the process of feature selection. By lowering the dimensionality of the data set and removing irrelevant features, this step has significance in improving the accuracy and computational efficiency of models [43].

Correlation Analysis: The linear connection between two variables can be measured using correlation analysis. It is common to compute the correlation between each predictor variable and the target variable (label) while choosing features. Low or negative correlation features are eliminated because they probably don't improve the performance of the model; features having a high correlation with the class label are kept. A Pearson correlation coefficient is used to find these links in binary classification issues [44].

2.4.3 Significance of Dimensionality Reduction

Dimensionality reduction's main objective is to remove duplicate and unnecessary features, which can enhance a model's interpretability while also speeding the training process. The presence of many variables can cause overfitting issues, in which the model learns particular patterns from the training data but is unable to generalize for new data, in problems involving high-dimensional datasets, such as network-based intrusion detection systems.

Principal Component Analysis (PCA)Principal component analysis () and Linear Discriminant Analysis (LDA)linear discriminant analysis () are two common methods for reducing dimensionality. However, they should be used properly since they may modify the features of the data, making it harder to interpret. In the context of Internet of Things networks, where anomalies are many and dynamic, incorrect feature removal may cause problems in the detection of some attacks [45].

2.5 Binary vs. Multi-Class Classification

In Machine Learning, classification is a key step, especially for IDS, where the objective is to distinguish between suspicious and benign traffic, or even between several cyberattack types. The complex nature of the issue and the features of the data used to train the models determine whether to use binary or multiclass classification.

2.5.1 Differences between Binary Classification and Multiclass

Binary Classification

In binary classification, the goal is to split the data into two distinct classes. In intrusion detection systems, this approach is often used to classify network traffic as normal (benign) or abnormal (malicious).

For instance, a binary classification model for anomaly detection can identify if the traffic on a network is normal or suspicious of an attack (0 or 1).The binary approach is used in security systems because it requires less complexity, resulting in simpler training and application of models. However, binary classification might not be enough in more complicated situations where many attack types need to be identified.

Multi-class classification

With multi-class classification, the binary concept is expanded to handle more categories. Identifying several cyberattack types is the goal of multiclass classification, as opposed to splitting data into two classes (normal vs. abnormal). This method can be used in the context of cybersecurity to categorize network traffic into several groups, including Denial of Service (DoS), Scanning, and even Bennign Traffic.

When attempting to gain additional accuracy in intrusion detection, multi-class classification is important because it enables the system to find the precise type of attack that is happening alongside detecting any signs of malicious traffic.This has particular importance for modern security systems, since fast and accurate identification of various threats can significantly impact damage mitigation.

When it comes to simplicity versus granularity, binary and multiclass classification contrast significantly. Although binary classification provides a more straightforward, computationally efficient, and frequently easier to use method, it eliminates the ability to offer an in-depth analysis of threat types. On the other hand, multi-class classification offers more implementation issues including class balance and the requirement for bigger volumes of data, but it also offers an additional level of detail and better adaptability for complex scenarios.

The decision between binary and multiclass depends on particular conditions and performance.Multiclass classification might be a better option in large-scale networks with diversified traffic and multiple threats in order to ensure a targeted response. However, a binary model can be more suitable in situations when the main objective is to quickly recognize any kind of anomaly.

2.6 ML Algorithms for Intrusion Detection

2.6.1 Algorithms Used in Project

Now, the algorithms for anomaly detection will be reviewed together with their advantages and disadvantages, with a focus on how they relate to smart home cyber security.

1. Logistic Regression

Logistic regression can be used to classify the question of whether or not a sample is an anomaly in binary task classification.Its benefits include strong efficiency in linear applications and simplicity in implementation and interpretation.However, it has disadvantages, including decreased efficiency in unbalanced data sets and a restriction in capturing non-linear relationships without extra adaptations [46].

2. K-Nearest Neighbors (KNN)

KNN is an instance-based method that uses the closest known instances to classify new scenarios.Among its benefits are its simplicity and ease of comprehension, as well as the fact that it does not assume anything regarding the data's distribution. However, it has

disadvantages including requiring a lot of computing power on big data sets and being dependent on the scale and K value selection [46].

3. Random Forest

An ensemble approach that boosts reliability and precision by using several decision trees.Its benefits include the capacity to handle imbalanced and non-linear data, decreased overfitting, and improved accuracy.Nevertheless, it has drawbacks such requiring more computer power and being harder to interpret than straightforward models [46].

4. Decision Tree

Regression and classification using decision trees create subgroups from the data. Their benefits include being simple to comprehend and interpret, as well as having the ability to handle numerical and category features. But when little changes in the data occur, they might become unreliable and susceptible to overfitting [46].

5. Multilayer Perceptron (MLP) Classifier

A type of neural network called a MLP is able to model complex links between features. Its flexibility in handling various data types and its ability to capture complex non-linear relationships are among its benefits.Its drawbacks include the requirement for a larger data set and a strong fit of the hyperparameters in order to prevent overfitting [46].

6. Stochastic Gradient Descent (SGD) Classifier

The SGD Classifier can handle big datasets, is effective for classification issues, and is used for large-scale optimization.Its benefits include versatility in permitting the selection of various loss functions and great efficiency on huge datasets.However, it can converge to less-than-ideal solutions because it is dependent on the volume of data provided and the learning rate [47].

2.6.2 Model Comparison

It is necessary to compare anomaly detection models in order to evaluate the dependability and efficiency of the methods that have been used. Among the most important performance indicators are F1-Score, Precision, and Recall.

Performance Metrics

Performance metrics provide an important part in measuring the efficiency of models. Among the main metrics are:

- **Precision**: It measures the ratio of True Positives (TP) to the total number of expected positives (TP + False Positives (FP)) :

$$\text{Precision} = \frac{TP}{TP + FP}$$

  Precision is crucial in scenarios where the false positive rate is high, such as in cyber security systems. [[48]].

- **Recall**: It measures the proportion of true positives in comparison to the total of real positives (TP + FN):

$$\text{Recall} = \frac{TP}{TP + FN}$$

Recall is important in situations where the detection of all anomalies is critical, even if this leads to more false positives. [48].

- **F1-Score**: It is the balanced mean between precision and recall, offering a single metric that balances both:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

This metric is useful when there is an imbalance between classes. [48].

- **Accuracy:** It is the most popular and frequently used method for assessing how well an algorithm performs in classification situations. Its definition is the proportion of correctly identified data items to all observations:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Although accuracy is often used, there are situations in which it is not the best performance metric, particularly when the target variable classes in the dataset are unbalanced. [48].

2.7 Model Combination and Ensemble Learning

Several models are combined in ensemble learning to increase prediction robustness and accuracy. This part discusses the theory behind ensemble learning and evaluates several methods.

2.7.1 Ensemble Learning

The core idea of ensemble learning is that an ensemble of multiple models may provide better performance than a single model.The two ensemble methods that are most frequently used are **boosting** and **bagging**.

- **Bagging (Bootstrap Aggregating)**: Using sampling and replacement, this strategy creates multiple subsets of data from the original set. The predictions are aggregated, typically by voting (for classification) or averaging (for regression), after each model is trained on a distinct subset. Bagging is an useful technique for minimizing variation, particularly in unstable models like decision trees. [49].
- **Boosting**: In opposition to bagging, boosting trains models in order, with each new model focusing on the cases that previous models classified incorrectly.All of the models are combined and weighted, with the strongest models having the greatest influence.Boosting is often used with algorithms like AdaBoost and Gradient Boosting, which can increase accuracy in unbalanced datasets. [49].

In the context of this project, the choice of bagging as an ensemble technique is explained by several points:

- When variation is a concern, like in cyber security datasets where data can be noisy and unstable, bagging works well.
- By learning from different subsets of the data, the technique enables better generalization and creates a more reliable anomaly detection model.

- Bagging is simple to implement and provides a straightforward way to reduce variance.

2.8 APIs for Integrating ML Models

Integrating Application Programming Interface (API) into the real-time anomaly detection system is a crucial step in ensuring efficient scalability, visualisation and interaction with the data and trained models. For this project, it was decided to use two main APIs: **FastAPI** and **StreamlitAPI**. Both were chosen due to their simplicity, performance and compatibility with the development environment. This section details the implementation of these APIs, as well as a comparison with other similar alternatives, justifying the choices made for the project.

2.8.1 FastAPI: Backend Interaction and Integration

FastAPI was the main choice for creating the backend API. The main function of this API is to communicate between the anomaly detection model and other system components, facilitating the sending and receiving of data and obtaining real-time forecasts. FastAPI is known for its high performance, using *asynchronous programming* and taking advantage of the speed of `Starlette` and `Pydantic` for data validation.

FastAPI's advantages

FastAPI was chosen for various reasons:

- **Performance**: FastAPI is one of the fastest APIs available for Python, rivalling more reliable frameworks such as `Node.js` or `Go` in terms of performance. For real-time anomaly detection systems, this feature is fundamental, as it allows them to handle large volumes of network traffic without introducing excessive latency.
- **Ease of Use**: The API offers a simple and intuitive syntax, with native support for Python's *type hints*, which makes the development process more efficient and less prone to errors.
- **Async/await support**: The ability to handle asynchronous calls allows the API to process multiple requests efficiently, optimising response times and the use of resources [[50]].

FastAPI alternatives

Some of the popular alternatives to FastAPI include:

- **Flask:** One of the best-known options in the Python ecosystem for creating APIs. However, Flask is less performant than FastAPI, especially in scenarios of high demand and real-time processing, as is the case with this project. Although it is more flexible in some respects, its lack of native support for data validation and asynchronous programming was a decisive factor in choosing FastAPI.
- **Django Rest Framework (DRF):** DRF is a reliable and complete solution based on `Django`.Although it offers a wide range of functionalities, it is heavier and more complex compared to FastAPI, which would make development less agile for this specific project, where performance is a priority [50].

FastAPI was therefore the ideal choice for the system's backend, offering an optimum balance between development simplicity, performance and scalability.

2.8.2 Streamlit: Visualisation and Interaction with the Model

For the visualisation layer and interaction with the end user, the **Streamlit** was chosen. Streamlit enables the creation of interactive dashboards, making it easier to visualise model results in real time, as well as to analyse performance metrics and interpret anomalies detected. The integration of Streamlit with FastAPI allows users to interact directly with the model, sending new network traffic data to be analysed and receiving immediate feedback on predictions.

Streamlit's advantages

Streamlit was chosen for the following reasons:

- **Easy to use and implement**: The Streamlit API has been designed so that any developer with a basic knowledge of Python can quickly create graphical interfaces. No in-depth knowledge of `HTML`, `CSS` or `JavaScript` is required [51].
- **Dynamic visualisation**: Streamlit allows graphs, tables and other visual elements to be updated in real time, without the need to reload the page.This is essential for this project, where continuous visualisation of detected anomalies is a constant necessity [51].
- **Integration with machine learning libraries**: Streamlit has native integration with popular machine learning libraries, such as `scikit-learn`, `TensorFlow`, and `Pandas`, which makes it easy to display model results and data in tabular or graphical format.
- **Focus on rapid prototyping**: Streamlit is ideal for prototyping dashboards quickly and iteratively, allowing the visualisation system to be adjusted as the project's needs evolve [51].

Streamlit Alternatives

Some alternatives to Streamlit for data visualisation include:

- **Dash (Plotly)**: Dash is a popular alternative for creating dashboards in Python. However, Dash is more complex to set up and requires greater familiarity with web interface development. Although it has greater flexibility in terms of design and customisation, the simplicity offered by Streamlit was a decisive point for the project [51].
- **Bokeh**: Bokeh is an interactive visualisation library that lets you create complex graphics.However, similar to Dash, it requires a higher level of configuration and is more suitable for advanced developments where fine control over visualisation is required.

Choosing Streamlit enables agile development and effective visualization, ensuring users can easily interact with model results and track anomaly detection in real time.

2.9 State of the Art Conclusion

The main concepts and techniques for anomaly detection in cyber security systems have been addressed in this chapter. It was also mentioned how important it is to evaluate

measures like recall, precision, false positives, and false negatives since these are essential for assessing how well models work. The choice of the best model should be driven by a thorough understanding of these measurements.

A comprehensive study took place on the ensemble learning tactics, with particular emphasis on methods like bagging and boosting.Because bagging can lower variance and boost prediction reliability in situations where data can be unpredictable and noisy, it was chosen for our research.The decision of using Streamlit for visualisation and FastAPI for API integration is a result of the need for a solution that not only meets performance requirements but also provides users an enjoyable experience.

These studies form the basis that supports the practical pipeline, ensuring that the decisions made during development are both informed and successful.

Chapter 3

ML-Based Algorithm

This chapter provides a presentation and discussion of the proposed system based on ML algorithms for anomaly detection.

3.1 Workflow of the Anomaly Detection Models

Cyberattacks of every type, from basic reconnaissance to more complex and damaging attacks like Denial of Service (DoS) can target smart homes. When training anomaly detection models, the variety of connected devices and potential threats represents an important issue. Managing the different volumes of data related to various attack types is one of the main challenges.Due to the need for the model to be versatile enough to accurately detect anomalies in both high- and low-volume data environments, this imbalance makes modeling challenging.

To effectively train models, the data is generally categorized into two main traffic types:

- **Malicious Traffic**: This involves data generated by a variety of cyberattacks, including DoS and scanning. Each type of attack introduces different patterns and anomalies that the detection model should monitor. Building a generalized model is made harder by variability in the frequency and intensity of malicious traffic.
- **Benign Traffic**: This includes regular, day-to-day communication between the smart home's devices. In order to train the model to distinguish between normal patterns and anomalies, benign traffic is essential. On the other hand, benign traffic can vary greatly depending on the type of device.

Regardless of the complexity or quantity of the attack traffic, anomaly detection models can be better adjusted to detect anomalies from common patterns by understanding and classifying traffic into these two main groups.

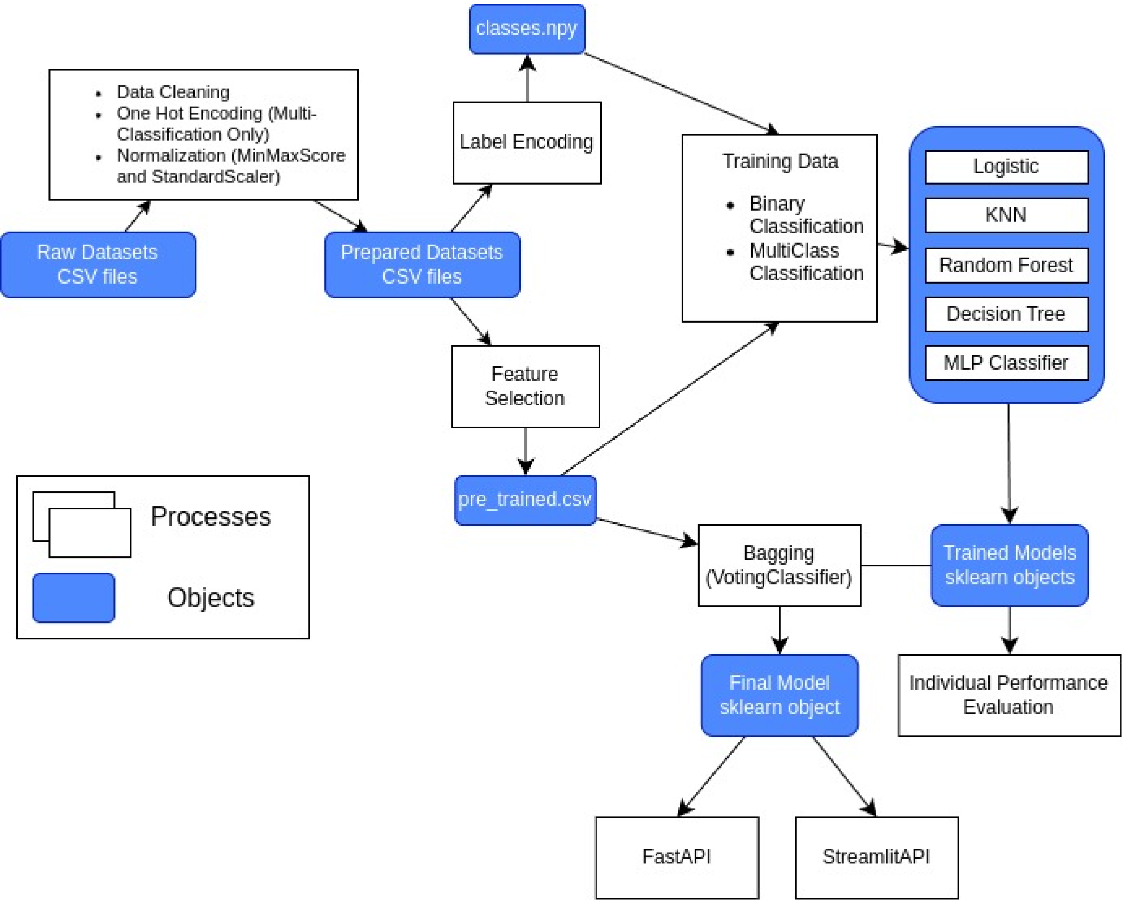


Figure 3.1: Workflow of the Anomaly Detection System

As shown in Figure 3.1, the model training process was carried out through many steps. Initially, network datasets were used, which were cleaned using one-hot coding for multiple classifications and normalization using either *StandardScaler* or *MinMaxScaler*. To make them easier to use later, the datasets were cleaned, labeled, and saved in *.npy* files.

The created datasets then underwent to feature selection in order to reduce dimensionality and improve efficiency.The final data was saved in a file named *pre_trained.csv*, which was used to train the models.

For training, a number of classification methods were chosen, including binary and multi-class models like MLP, KNN, Random Forest, Decision Tree, and Logistic Regression. After individual training, an ensemble model built on Bagging—VotingClassifier, in this case—was used to combine the predictions made by the classifiers that were previously trained.

After evaluating the final model's individual performance, a sklearn object was created and included in StreamlitAPI and FastAPI to enable inference and result visualisation. This ensures that the model can be used in real-world scenarios through RESTful and visualization interfaces.

Now, for a detailed explanation of each process in the pipeline, the stages from data preparation to integration with the APIs will be covered.

3.1.1 Data Collection

For any Machine Learning system to be successful, collecting data is an important step, especially when dealing with challenging tasks like identifying anomalies in network traffic. To ensure the relevance of both benign and malicious traffic, it was needed to fully investigate the datasets that were available, focusing on the types of attacks that each one covered, in the context of this research, which is intended to identify cyber-attacks in IoT environments in smart homes.

Explored Datasets

A diversified dataset that accurately reflects both benign and malicious traffic is essential for training an anomaly detection system. A number of cybersecurity industry-recognized datasets containing various attacks and network scenarios were explored. The primary datasets used for the research are shown below, along with a brief overview of each one and the range of attacks they address.

- **UNSW-NB15** is a dataset that has been used to mimic different kinds of attacks in a real network environment. This dataset covers both harmful and benign traffic, including attacks like **DoS, Exploits, Fuzzers, Analysis, Reconnaissance**, and others. It is helpful for analyzing cyberattacks on complex networks due to the range of traffic and integration of current attacks [52].
- **CTU-13** consists of a collection of network traffic recordings showing several attack scenarios. The main attack type that this dataset addresses is **botnets**, in which compromised computers execute malicious commands in a coordinated way. CTU-13 is valuable for studying malicious communications and unusual network behaviour in compromised systems [53].
- **CICIDS 2017 and CICIDS 2018** were created to show network traffic scenarios in corporate environments. They include various types of attack, such as **DoS, DDoS, Brute Force, Infiltration and Port Scanning**. The main advantage of these datasets is their scope, as they capture both targeted attacks and benign traffic generated in real scenarios [54].
- **IoT-23** contains network traffic captured from IoT (Internet of Things) devices and includes attack scenarios against devices such as security cameras and smart thermostats. The attacks included include everything from **network scanning and DoS to Man-in-the-Middle (MiTM)**, with a special focus in IoT Wearable devices [55].

Selection of Datasets

It became clear to start off the project's development with simpler datasets, including **UNSW-NB15** and **CTU-13**. These datasets were selected for a number of reasons. First of all, the variety of scenarios they both cover is important since it offers a solid basis for training models of anomaly detection in a range of conditions. In addition to having various attack types, UNSW-NB15 and CTU-13 provide a range of both benign and malicious traffic patterns.

Also, both datasets are easily manipulated due to their simplicity, which makes it possible to clean and process the data fast without compromising the accuracy of the project.This simplifies the data preparation and analysis.

Finally, using the CTU-13 dataset supports the goal of having a sufficient quantity of events to train the model effectively, without requiring managing a variety of diverse cyberattacks. Because CTU-13 has a large number of events and a label that clearly indicates whether the traffic is benign (normal traffic = 0) or malicious (attack = 1), making it possible to analyze the data in binary form with ease, which allows for easier training model while keeping the level of quality required for the project's initial development.

Creating a Combined Dataset

A new method based on combining the CTU-13 and UNSW-NB15 datasets to create a third dataset was developed as part of the project plan. The new dataset adds more reliability to the detection algorithm, presenting a wider range of benign and malicious patterns, by combining the variety of attacks in UNSW-NB15 with the quantity of events found in CTU-13.

In addition, combining datasets allows machine learning models to be far more generalized.Combining multiple scenarios introduces diversity that helps train models to detect deviations in a variety of situations, which is essential for adapting to dynamic network environments like smart homes [56].

A solid dataset for identifying cyberattacks on Internet of Things networks is provided by the combined CTU-13 and UNSW-NB15 datasets, which allows for both binary classification (attack or no attack) and multi-class classification (identifying the kind of attack).

Initial Data Cleaning

Before initiating the cleaning process, an in-depth statistical analysis of the features found in the three datasets—UNSW-NB15, CTU-13, and CTU-UNSW—was performed. The goal of this investigation was to make sure that pre-processing was consistent while also spotting general patterns and any outliers.

The following properties were evaluated for each dataset:

- **Distribution of Values**: Understanding each feature's variability and data types by identifying its value distribution can help to spot anomalies such as out-of-range values.
- **Missing Values**: Analysing missing values, removing completely missing features and identifying processing strategies for partially missing columns.
- **Invalid values**: Incorrect values are replaced. For instance, to ease the cleaning process, "-" value was substituted for the NaN that was included in the service column of the UNSW-NB15 and CTU-UNSW datasets.
- **Duplicates**: Finding and removing redundant inputs that might affect the behavior of the models.

In addition, the cleaning process included the following steps specific to each dataset:

- **Filter Attack Categories**: The UNSW-NB15 dataset contains many different kinds of cyberattacks, making it necessary to remove events considered irrelevant to the project's objective. There were just three categories remaining: **'DoS', 'Reconnaissance' (Scanning), and 'Normal'**. The CTU-UNSW dataset went through under an identical set of criteria, and only the categories stated were retained after filtering the `attack_cat` column.

- **Conversion of Data Types**: According to the features auxiliar files, the columns in the three datasets were converted to the correct types (nominal, integer, binary, or float). This conversion was needed to provide accurate data representation, particularly for the numerical and category columns.
- **Auxiliary Feature Files Creation**: Auxiliary feature files were created during the pre-processing stage with the goal to convert each feature to the proper data type (nominal, integer, binary, or float). This allowed future transformations and analysis easier, making sure the data types were converted across the datasets.

The project's initial objective of detecting Denial of Service and scanning attacks had been achieved by the three datasets due to the filtering of attack types and the proper handling of missing or invalid values.

The analysis didn't include the **Man-in-the-Middle attack** category since there was insufficient data in the datasets that were selected. It was found after evaluating the datasets that there weren't enough samples available for MiTM to offer comprehensive evaluation or successful model training.

Following these cleaning procedures, the datasets were finally kept for further analysis and model training. Visualizations, such as pie charts, were created to show the distribution of binary labels (normal/abnormal) and multiclasses (attack categories) for each dataset in order to more effectively represent the overall state of the data.

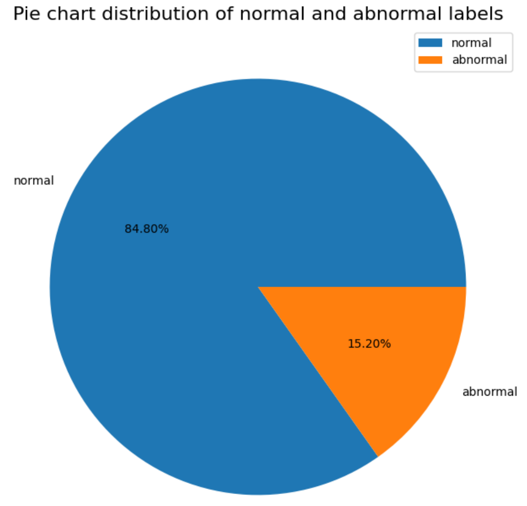


Figure 3.2: Distribution of Normal and Abnormal Labels in the UNSW-NB15 Dataset

The pie chart above shows the distribution of binary labels in the UNSW-NB15 dataset:

- Normal: 84.80%
- Abnormal: 15.20%

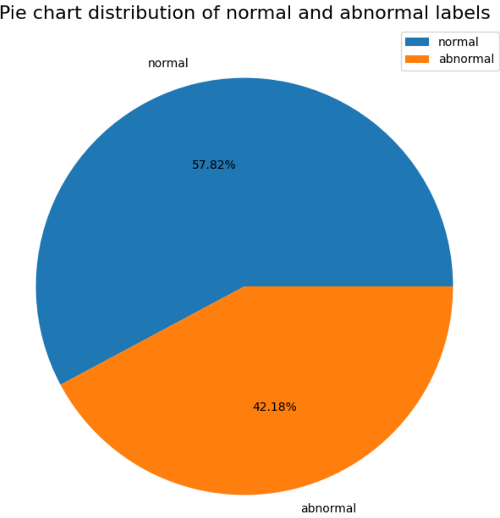


Figure 3.3: Distribution of Normal and Abnormal Labels in the CTU-13 Dataset

The pie chart above shows the distribution of binary labels in the CTU-13 dataset:

- Normal: 57.82%
- Abnormal: 42.18%

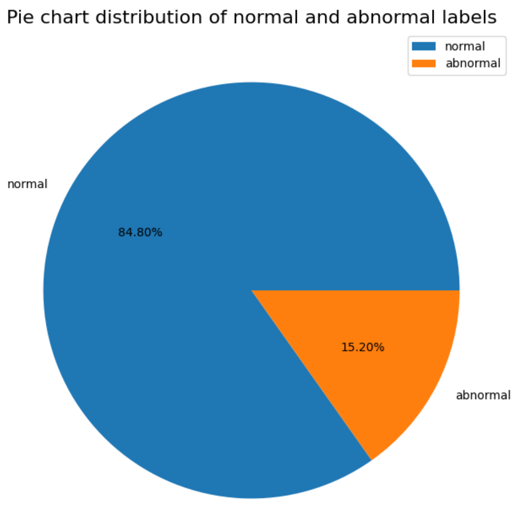


Figure 3.4: Distribution of Normal and Abnormal Labels in the CTU-UNSW Dataset

The pie chart above shows the distribution of binary labels in the CTU-UNSW dataset:

- Normal: 84.80%
- Abnormal: 15.20%

The following pie charts show the distribution of the different attack categories (multiclass) for the `UNSW-NB15` and `CTU-UNSW` datasets containing the `attack_cat` column.

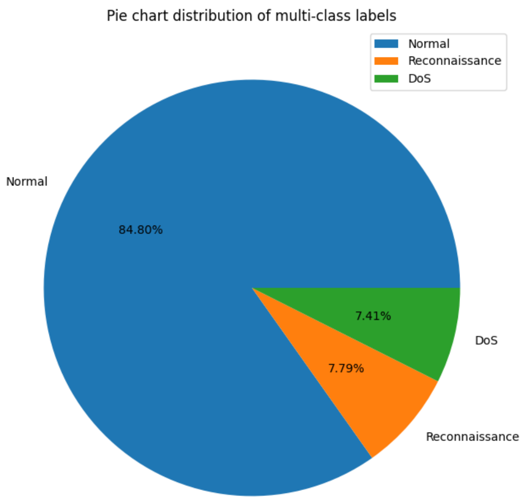


Figure 3.5: Distribution of Attack Categories in the UNSW-NB15 Dataset

The pie chart above represents the attack category distribution in the UNSW-NB15 dataset:

- Normal:84.80%
- Reconnaissance:7.79%
- DoS:7.41%

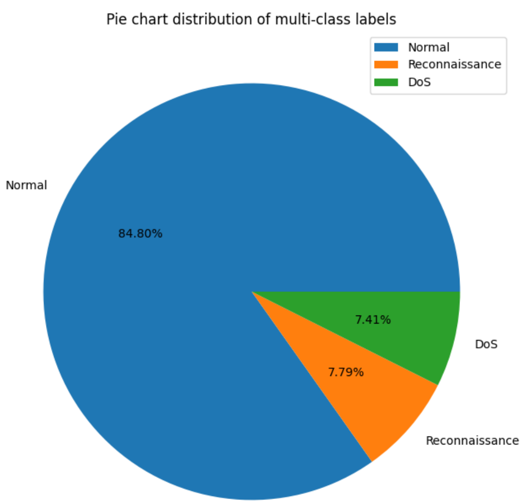


Figure 3.6: Distribution of Attack Categories in the CTU-UNSW Dataset

The pie chart above represents the attack category distribution in the CTU-UNSW dataset:

- Normal:84.80%
- Reconnaissance:7.79%
- DoS:7.41%

A number of challenges occurred when merging the two datasets (UNSW-NB15 and CTU-UNSW). The presence of features that represented the same information under different names was one of the main challenges. In order to fix this, an extensive study was done to identify and merge these features, thereby standardizing the names and ensuring integrity in the data.

Differences in data types between datasets, such as numerical columns interpreted as texts in one dataset and as integers in another, were another common challenge.Additional conversions were required to fix these problems and ensure feature compatibility.

Data Cleaning Summary

A summary of the datasets' initial statistics both before and after cleaning is displayed in the table 3.1 below. The number of original lines and features can be seen in this chart along with the effect of cleaning in terms of lines and features that have been changed or eliminated.

Table 3.1: Summary of Dataset Statistics Before and After Cleaning

Dataset	Original Lines	Lines after Cleaning
UNSW-NB15	175341	22982
CTU-13	92212	92212
CTU-UNSW	267553	22982

As a way to ensure the quality of the data used for the modeling process, prior data cleaning was required. The methods used generated cleaner datasets ready for further analysis. CTU-13 dataset was already precleaned, so the number of original lines before and after cleaning are the same.

3.1.2 One-Hot Encoding

A technique that is often used in the context of preparing data for machine learning algorithms is *One-Hot* encoding (*One-Hot Encoding*), particularly when working with categorical attributes. As most machine learning algorithms struggle with raw categorical data, this technique turns categorical variables into binary variables, enabling models based on linear math to handle such qualities properly. The *UNSW-NB15* and *CTU-UNSW* datasets are the two instances where the approach can be especially helpful.

Every individual value in a categorical column is converted into a new binary column using the *One-Hot* encoding.The original variable is represented by each column, and the rows are filled with *0* or *1* values based on whether that category is present or absent for the corresponding dataset observation.

Application of One-Hot Encoding

It is required to first determine which columns in the dataset are categorical before applying *One-Hot* encoding. This can be achieved by choosing non-numerical variables. The code used for this task joins the output with the numerical columns in the original dataset after choosing categorical attributes and applying the encoding. Finally, the initial category columns are removed, leaving only numerical columns in the dataset—including those produced by the *One-Hot* coding.

3.1.3 Data Normalization

Data normalization is an essential step in the data preparation process that comes after the categorical variable encoding stage. Many *machine learning* algorithms, specially those that are sensitive to data sizes, like neural networks and distance-based algorithms like *k-nearest neighbours*, can perform better when normalization is applied to ensure that all features are on the same scale. Two normalization techniques were used in this work: Z-Score (Standardization) and Min-Max Scaling, both of which are explained below.

Min-Max Scaling

The approach known as Min-Max normalization or *Min-Max Scaling*, requires modifying variables so that their values fall inside a predetermined range, typically ranging from 0 to 1.

The formula used to perform the Min-Max transformation is given by:

$$X_{\text{scaled}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}} \tag{3.1}$$

Where:

- X is the variable's original value,

- X_{min} e X_{max} are the minimum and maximum values of the variable, respectively.

The main advantage of Min-Max normalization is the preservation of proportional relationships between values when they are changed to the same scale. However, because unusual values may distort the data range and reduce the transformation's efficiency for the majority of observations, this method can be vulnerable to outliers.

This normalization was done on all the numerical columns in the datasets, with the exception of the identification and label fields (*id* and *label*), using the *MinMaxScaler* from the *scikit-learn* library.After normalization, the dataset was saved for use in further phases of model training.

Z-Score Normalization (Standardization)

Normalization based on *Z-Score*, also referred to as *standardization*, was the second technique used.Using this method, the data is transformed so that each variable's mean is equal to zero and its standard deviation is one. When the data has a Gaussian distribution, the *Z-Score* normalization is especially helpful since it attempts to eliminate scale differences between the variables while preserving the original distribution.

The formula for Z-Score normalization is as follows 3.2:

$$Z = \frac{X - \mu}{\sigma} \tag{3.2}$$

Where:

- X is the variable's original value,
- μ is the variable's mean,
- σ is the variable's standard deviation.

Since *Z-Score* doesn't rely on extreme values like *Min-Max Scaling* does, it is less susceptible to outliers. It doesn't, however, restrict the data to a predetermined range, which, in the event of outliers, may lead to normalized values that are far above 1 or below 0.

Additionally, this method was applied to the same numerical columns as the previously encoded dataset using the *StandardScaler* method and the *scikit-learn* library [57]. After normalizing the dataset using *Z-Score*, it was saved for a following Min-Max normalization comparative analysis and model training.

Comparing Techniques

The pros and cons of both normalization strategies change based on the machine learning algorithm and the context. Z-Score normalization is appropriate for models that assume Gaussian distributions or are less sensitive to outliers, such logistic regression and SVM, while Min-Max normalization is best for techniques that require data inside particular ranges, like neural networks. Both approaches are applied with the goal of evaluating which performs better in the context of anomaly detection.

3.1.4 Label Encoding

The qualitative category variables are converted into integer values by label encoding. In order for machine learning algorithms to understand and interpret these variables, this conversion is needed. When categories can be represented by numbers without sacrificing essential information, the technique is especially helpful.

Binary Encoding

An approach for variables with two categories is binary encoding. The method is converting two different numerical classes from categorical labels to numeric. This is how the procedure is explained:

- **Definition of Categories:** Categories are converted into two main classes, often represented as 0 and 1. In an anomaly detection scenario, labels can be mapped as 'normal' and 'abnormal'.
- **Transformation:** Each categorical value is replaced by a corresponding binary value. This is done to simplify the analysis and application of machine learning algorithms that require numerical variables.
- **Class Storage:** To be able to achieve consistency in data interpretation across the subsequent stages of the process, the encoding-resulting classes are preserved.

This approach is a crucial step in getting the data ready for models that use binary variables, as it is valuable for them.

Multi-class encoding

Multi-class encoding is used when a categorical variable has more than two distinct categories. The process for multi-class coding includes:

- **Identifying Categories:** Every category has a distinct integer value assigned to it. For instance, the values 0, 1, and 2 can be used to represent the categories "Normal", "Scan", and "DoS" respectively.
- **Transformation:** To assign a corresponding numerical value to each category, encoding is used. This makes it possible for categorical variables to be processed by machine learning algorithms into a format that is suitable for training and analysis.
- **Class Storage:** The resulting numerical classes are saved, just like with binary encoding, so that the transformation can be performed and verified in later stages.

This method is essential for multi-category classification problems because it enables the model to manage and process categorical information correctly.

3.1.5 Feature Selection

One of the basics in pre-processing data is feature selection, which finds the variables that are most important for the machine learning model. This process can improve the interpretability of the results and the efficiency of the model in addition to reducing the complexity of the dataset.

Feature Selection for Binary Classification

Feature selection for problems with binary classification is often based on examining the correlation between the features and the binary label. The steps are as follows:

- **Correlation Analysis:** Between each of the numerical features and the binary label, a correlation is calculated. The linear link between two variables' strength and direction are indicated by this metric.
- **Correlation Filtering:** Features are only chosen if their correlation with the binary label is greater than a predetermined cutoff (0.3). This rule guarantees the retention of only the most significant features.
- **Creation of the New Dataset:** Only the chosen features remain in the smaller dataset.The machine learning model is subsequently trained using this new dataset.

Finding variables that have a strong correlation with the binary label using correlation-based selection allows a boost in the model's efficiency.

Feature Selection for Multi-class Classification

The feature selection process for multi-class problems with classification can be performed in a similar way but taking into account more than one category:

- **Correlation Analysis:** The multi-class labels and each numerical feature are correlated using correlation analysis. Since there can be more than one class that needs to be taken into account, the correlation in this instance is assessed using the absolute value.
- **Correlation Filtering:** Features are chosen if their correlation with the multi-class label is greater than the established threshold (0.1).
- **Labelling:** The multi-class label is sure to be included in the features that have been selected.This is done to make sure that the important output variable stays in the final dataset.
- **Creation of the New Dataset:** In order to guarantee that the final dataset keeps the important output variable, the multi-class label is guaranteed to be included among the chosen features.

The approach used for feature selection in multi-class situations aims to optimize the relevance of variables for class prediction while considering the additional complexity of multiple categories.

Chapter 4

Model Training

4.1 Individual Model Training

Two main datasets were used for the individual model training process: one for binary classification and the other for multi-class classification. The purpose of this process was to test several algorithms and assess each one's performance regarding anomaly detection. The steps involved in the training pipeline—data preparation, algorithm selection, and evaluation metrics—are outlined below.

4.1.1 Data Preparation

It required a lot of preparing for the data before starting the model training. For a binary classification task, datasets with binary labels indicating whether a sample is benign or malign were used. For the multiclass classification task, datasets that classify each sample into either benign traffic or one of the attack types were also used.

Using a split of 80% for training and 20% for testing in binary classification and 70% for training and 30% for testing in multiclass classification, the data was divided into training and test sets. The objective for this separation is to keep a sufficient amount of data for testing so that the models can be consistently evaluated. In addition, 95% of the original data's variance was preserved by the use of Principal Component Analysis (PCA) in the dimensionality reduction technique, which increases computing efficiency without significantly sacrificing information.

4.1.2 Selected Algorithms

For individual training, a variety of classification algorithms were selected. The entire list of algorithms includes:

- **Logistic Regression**: A solid algorithm for classification, especially in binary problems, but also applied here for multiclass classification with adjustment for multinomial logistic regression.
- **K-Nearest Neighbors (KNN)**: An instance-based learning algorithm, where classifications are made based on the nearest samples.
- **Random Forest**: An ensemble method that builds multiple decision trees and combines their predictions, ideal for dealing with complex, non-linear data.
- **Decision Tree**: An easy-to-understand algorithm that splits data based on simple rules, often used for classification problems.

- **Multilayer Perceptron (MLP)**: A type of feedforward neural network with hidden layers, capable of capturing complex links in the data.
- **Stochastic Gradient Descent (SGD)**: An efficient algorithm for large datasets, using a stochastic gradient for optimisation. In this context, the *hinge* type loss was used for classification.

4.1.3 Evaluation Metrics

A set of common measures for categorization issues were used to evaluate the models' performance, including:

- **Mean Absolute Error (MAE)**: Determines the average size of the errors between the values that were predicted and the actual values.
- **Mean Squared Error (MSE)**: Calculates the mean square of the errors, punishing larger errors.
- **Root Mean Squared Error (RMSE)**: The square root of the MSE, keeping the original data unit.
- **Coefficient of Determination (R^2 Score)**: Measures the proportion of variance explained by the model in relation to the data test.
- **Accuracy**: The proportion of correct predictions in relation to the total number of samples.
- **Precision**: The ratio of true positive predictions to the sum of true positive and false positive predictions. It quantifies the accuracy of positive predictions.
- **Recall**: The ratio of true positive predictions to the sum of true positives and false negatives. It measures the ability of the model to identify all positive instances.
- **F1-Score**: The harmonic mean of precision and recall, providing a balanced measure when there is an uneven class distribution.
- **Support**: The number of actual occurrences of each class in the dataset. It provides context for the other metrics.
- **Classification Report**: It shows the precision, recall, and F1-score metrics for each class in the test set.

These measures allowed to compare the algorithms in-depth and provide an analysis of each model's performance for both binary and multiclass classification.

4.1.4 Results and Metrics

Individual Model Binary Evaluation

The performance evaluation of several machine learning models used with binary classification datasets is shown in the next part. Two normalization strategies—Min-Max and Z-Score—were used for the evaluation. These strategies are essential to improving the efficiency and accuracy of models, particularly in situations involving unbalanced data.

Model	Accurary MM (%)	R^2Score MM	Accuracy Z (%)	R^2Score Z
LogisticRegression	92.41	40.69	92.41	40.69
KNN	99.11	93.02	99.11	93.02
RandomForest	99.09	92.85	99.09	92.85
DecisionTree	99.09	92.85	99.09	92.85
MLPClassifier	93.78	54.32	93.15	46.37
SGDClassifier	84.38	-3.19	84.38	-3.19

Table 4.1: Model Performance using Min-Max and Z-Score Normalization on UNSW Binary Classification Dataset

The KNN, RandomForest, and DecisionTree models performed best when Min-Max normalization was applied. They constantly had high accuracy levels above 99%, demonstrating their excellent generalisation and anomaly detection abilities. While still showing good performance, the MLPClassifier model outperformed the Decision Tree and KNN models. With an accuracy of just 84.38% and a negative R^2, the SGDClassifier performed the lowest, making it inappropriate to use in an ensemble.

The same three models—KNN, RandomForest, and DecisionTree—continued to perform well in the Z-Score normalization scenario, with accuracy levels approaching 99% and extremely low error metrics.SGDClassifier also performed worse, exhibiting a negative R^2. The best models outperformed LogisticRegression and MLPClassifier, which displayed respectable performances. For the ensemble, only SGDClassifier will be eliminated.

Model	Accuracy MM (%)	R^2Score MM	Accuracy Z (%)	R^2Score Z
LogisticRegression	70.98	-14.55	73.48	-6.64
KNN	82.63	29.09	87.77	49.87
RandomForest	74.56	-2.84	80.50	20.09
DecisionTree	74.56	-2.84	79.44	15.74
MLPClassifier	70.98	-14.55	74.37	-3.59
SGDClassifier	70.98	-14.55	73.44	-6.75

Table 4.2: Model Performance using Min-Max and Z-Score Normalization on CTU-13 Binary Classification Dataset

After the Z-Score normalization process, the KNN, Random Forest, and Decision Tree models stood out, with KNN obtaining an accuracy of almost 83%. Because of their reliable performance, these three models will be used in the ensemble model training.

On the other hand, the Logistic Regression, MLP Classifier and SGD Classifier models showed bad results, especially after Min-Max normalization, with low accuracy and very negative R^2, suggesting convergence problems. As a result, these models were excluded from the final ensemble training.

Model	Accuracy MM (%)	R^2Score MM	Accuracy Z (%)	R^2Score Z
Logistic Regression	92.41	40.69	92.41	40.69
KNN	99.11	93.02	99.11	93.02
Random Forest	99.09	92.85	99.09	92.85
Decision Tree	99.09	92.85	99.09	92.85
MLP Classifier	93.78	54.32	93.15	46.37
SGD Classifier	84.38	-3.19	84.38	-3.19

Table 4.3: Model Performance using Min-Max and Z-Score Normalization on CTU-UNSW Binary Classification Dataset

The fact that the outcomes in the table below match those from the UNSW dataset suggests that there was a major problem with the data cleaning or merging procedure. The results' integrity becomes compromised by this issue, and the CTU-UNSW dataset cannot be integrated with the anomaly detection API. The use of this dataset will be discarded and will not be taken into consideration in any further phases of this project because of these issues.

Individual Model Multi-Class Evaluation

The CTU-13 dataset has a few limitations when it comes to multi-class categorization. This dataset is not adequate for training and evaluation in multi-class settings since it lacks any feature that may identify or categorize the specific type of attack, only a binary feature that shows the presence or absence of malicious activity. As such, it will be discarded for this approach.

Model	Accuracy MM (%)	R^2Score MM	Accuracy Z (%)	R^2Score Z
Logistic Regression	90.54	2.2613	90.65	4.76
KNN	97.10	65.63	96.91	63.17
Random Forest	97.19	68.01	97.16	67.21
Decision Tree	96.88	65.62	96.82	64.64
MLP Classifier	95.52	52.84	95.52	52.84
SGD Classifier	90.28	-4.0228	90.28	-4.0228

Table 4.4: EModel Performance using Min-Max and Z-Score Normalization on UNSW Dataset for Multi-Class Classification

The models of SGD Classifier and Logistic Regression were eliminated from the ensemble because of their poor performance, which was demonstrated by low accuracy rates. Due to their outstanding performance, the remaining models—KNN, Random Forest, Decision Tree and MLP Classifier—were taken into into account during the last ensemble phase. The accuracy of anomaly detection will be improved by the combination of these models.

4.2 Model Combination with Bagging

The strength and performance of the anomaly detection system was improved by the use of a model combination technique based on bagging. The goal of bagging (*Bootstrap Aggregating*) is to lower the variance of learning models by the pooled combination of

several classifiers. For cases where individual models have a tendency to overfit the training set, this method is especially useful. Bagging's primary goal is to improve generalization by merging the predictions of various classifiers.

The reasoning behind using *Voting Classifier* alongside with *hard voting* was its ease of use and efficiency in situations where the performance of each classifier is strong. By combining the models, the specific weaknesses of each classifier can be offset, creating a final model that is more accurate and reliable.

4.2.1 Binary Bagging

The first script was created to handle binary classification problems, such as *attack* and *non-attack*, in which the goal is to distinguish between the two categories. The *.csv* file which had the features and the label column (*label*), which represented the binary classes, were used to process the data in this case.

The features and labels were separated once the data was read. Afterwards, a random seed set was used to guarantee repeatability before the data was divided into training and test sets at a ratio of 80% for training and 20% for testing.

Previously trained models were loaded from `.pkl` files, each containing one algorithm. A *Voting Classifier*, a combination technique that enables the merging of many classification algorithms, was used to combine these models. Each model was given a vote in the final result. The *hard voting* method was applied in this instance, where the majority of votes decides the class that will be used.

The model was evaluated using the test data after the ensemble was successfully trained using the training set. The *classification report* metric was used to analyze the various classes that were predicted, and accuracy was calculated for both the training and test sets. In the end, the combined model was saved for later use in a `.pkl` file.

Ensembled Binary Model Metrics and Results

This subsection focuses into how ensemble models perform when used on datasets, with a special focus on accuracy metrics under different normalization scenarios.

```
Train Accuracy:  99.89665488169703
Test Accuracy:  99.41266043071568
Classification Report (Test):
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       691
           1       1.00      0.99      1.00      3906

    accuracy                           0.99      4597
   macro avg       0.98      0.99      0.99      4597
weighted avg       0.99      0.99      0.99      4597
```

Figure 4.1: Accuracy results for Min-Max Normalization for Binary Classification

```
Train Accuracy:  99.9564862659777
Test Accuracy:  99.65194692190559
Classification Report (Test):
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       691
           1       1.00      1.00      1.00      3906

    accuracy                           1.00      4597
   macro avg       0.99      0.99      0.99      4597
weighted avg       1.00      1.00      1.00      4597
```

Figure 4.2: Accuracy results for Z-Score Normalization for Binary Classification

After evaluating all of the datasets, the ensemble model with Z-Score normalization on the UNSW dataset came as the most reliable option for API integration. With a 99.65% accuracy rate on the test data, this model showed amazing performance metrics. Additionally, the modest variation between the test accuracy (99.65%) and training accuracy (99.96%) suggests that the model has an effective ability for generalization, reducing the chance of overfitting—an essential concern for production models.

Z-Score normalization offered an extra benefit, with slightly better accuracy metrics, making it a safer option for real-life scenarios where consistency in classification is important, even though the model with Min-Max normalization provided similarly impressive results.

4.2.2 Multi-Class Bagging

The second script was developed to deal with classification issues involving multiple categories. Since the system must be able to distinguish between various attack types and benign traffic, the scope of the challenge increases thanks to the diverse classes included in the datasets.

The script's structure is identical to the binary problem's, with the primary distinction being the kind of input data. Every model that was employed had been trained to solve a multi-class classification issue. Here, the *Voting Classifier* was also used, and the final classification was decided using the *hard voting* method.

The data was divided into training and test sets, and the ensemble was evaluated and trained using the same performance standards as in the prior script. After that, the generated model was saved for further usage.

Ensembled Multi-Class Model Metrics and Results

This part analyzes the efficiency of ensemble models using the UNSW dataset, with an eye on measuring accuracy in scenarios involving multi-class classification.

```
Train Accuracy:  98.2920859396247
Test Accuracy:  96.99804220143572
Classification Report (Test):
              precision    recall  f1-score   support

           0       0.84      0.81      0.83       352
           1       0.98      0.99      0.99      3906
           2       0.92      0.94      0.93       339

    accuracy                           0.97      4597
   macro avg       0.92      0.91      0.91      4597
weighted avg       0.97      0.97      0.97      4597

Modelo combinado salvo com sucesso!
```

Figure 4.3: Accuracy results for Min-Max Normalization for Multi-Class Classification

```
Train Accuracy:  98.29752515637749
Test Accuracy:  97.04154883619752
Classification Report (Test):
              precision    recall  f1-score   support

           0       0.84      0.81      0.83       352
           1       0.99      0.99      0.99      3906
           2       0.93      0.94      0.93       339

    accuracy                           0.97      4597
   macro avg       0.92      0.91      0.92      4597
weighted avg       0.97      0.97      0.97      4597

Modelo combinado salvo com sucesso!
```

Figure 4.4: Accuracy results for Z-Score Normalization for Multi-Class Classification

Z-Score normalization was the highest standard choice for API integration after an examination of the bagging models on the UNSW dataset using Min-Max (MM) and Z-Score (Z) normalizations. This model performs more efficiently overall, with a slightly better test accuracy of 97.04% compared to 96.49% for the MM model.

The Z-Score is especially remarkable for being able to handle unbalanced data, evidenced by the precision and recall (both at 0.99) with which the normal class (1) was found. While the recall for the DoS class (0) was also 0.81, indicating a good recognition rate, there is still room for improvement even with the precision of 0.84. The Z-Score model's weighted averages and F1-Score metrics (weighted F1-Score of 0.97), when applied to scenarios with uneven classes, indicate that this model can deliver a more balanced performance.

The individual and ensemble models' training steps generated good anomaly detection results.Models such as KNN, Random Forest and Decision Tree stood out in terms of both binary and multi-class classification performance, while Logistic Regression and SGDClassifier performed worse and were discarded. In most cases, Z-Score normalization performed better than Min-Max, achieving slightly better results. High levels of accuracy were achieved, particularly in the UNSW dataset, by using bagging through the Voting Classifier, which greatly increased the models' accuracy of the ensemble models in binary and multi-class classification.

These results give a solid basis for the anomaly detection system's implementation in production, ensuring both accuracy and ability for generalization.

Chapter 5

Implementation of APIs for Visualisation and Interaction

This section describes the actual implementation of the two distinct APIs that were used to enable communication and the user interface of the anomaly detection system. The frontend, which gives the user visual feedback in real time, and the backend, which processes data and makes predictions, may be easily integrated thanks to these APIs. Specifically, **FastAPI** was used to build an API for backend communication, while **Streamlit** was used to handle user interaction and visualization.

5.1 Data Format

It is essential to explain the structure and acquisition of the data used by machine learning models before getting into the integration of APIs.

Before individual model training began, the features for each binary and multiclass classification model were printed using the following code on the terminal:

```
# Print all features
print("\n Features in the dataset:")
print(data.columns)
```

An additional file, *.csv*, in the dataset, including details on every feature, was also studied. By identifying the type of variable connected to each feature, this file helped to guarantee that the attributes used to train the models were useful for identifying anomalies.

As a result, it was possible to identify the features and types of variables matching each model, as shown in the table 5.1 below:

Binary		Multi-Class	
Feature	Type	Feature	Type
ct_srv_dst	int	smean	int
trans_depth	int	dmean	int
synack	float	sttl	int
tcprtt	float		
ackdat	float		
sttl	int		
dttl	int		
ct_state_ttl	int		

Table 5.1: Features and types of variables for binary and multiclass classification models.

5.2 FastAPI: Backend Interaction and Model Integration

Building a FastAPI-based backend, which manages requests, validates data, and makes predictions based on trained anomaly detection models, was the initial step in the implementation process.

The API was created to take in network traffic data, process it with the help of machine learning models that have been trained, and then provide predictions in real time. The backend allows communication between the machine learning models that generate the predictions and the frontend, which collects input from users.

5.2.1 API Workflow

1. Data Reception: JSON-formatted data is received by the FastAPI backend from the frontend. The features that were used to train the anomaly detection models show up in the structure of the input data.
2. Model Inference
    - Binary Classification: This model is used to identify if an anomaly has been detected or if the traffic is benign.
    - Multiclass classification: The accurate type of anomaly or attack, such as DoS or reconnaissance, or if the traffic is benign is identified using the second model.
3. Predictions Provided: The JSON-formatted prediction results are then sent back to the frontend for processing and real-time user presentation.

Ensuring the backend could handle many requests at once without delaying predicted responses was a major implementation issue. This problem was solved by using FastAPI's built-in async/await capability for asynchronous operations, which let the system effectively handle numerous requests. Careful error handling was also used to handle possible problems like timeouts and incorrect data inputs.

5.3 Streamlit: Frontend and Real-time Dashboard

A dashboard using Streamlit was integrated for the front end to provide users with instant visual input about the anomaly identification method. Streamlit was selected because of its ease of use and fast process for creating eye-catching and interactive dashboards without requiring an in-depth knowledge of web development.

5.3.1 Dashboard Features

In order to provide clients with an easy-to-use interface for interacting with the system and monitoring real-time predictions, the Streamlit dashboard was built.

1. The results of anomaly detection are updated in real time on the "Live Detection" tab. The dashboard uses a dynamically created line chart to show the severity of traffic anomalies over time. Real-time updates are made to the graphic to reflect fresh information and forecasts.

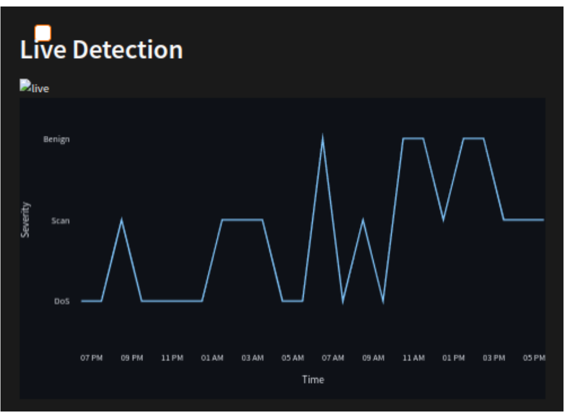


Figure 5.1: Live Detection StreamlitAPI Screenshot

2. History Detection Tab: This area lets users see previous anomalies that the system has found. The visualization of data from prior detection sessions makes it simple to monitor patterns and anomalies over time.

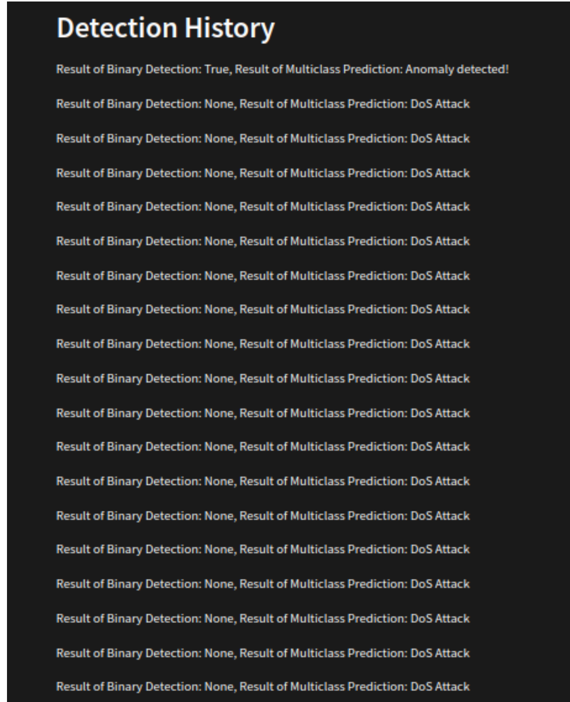


Figure 5.2: History Detection StreamlitAPI Screenshot

3. User Input: Users can enter their own network traffic statistics straight into the dashboard with Streamlit, allowing for instant analysis. To make it simpler for users to complete and submit the data required for prediction, the input fields are divided into sections(dataset features)

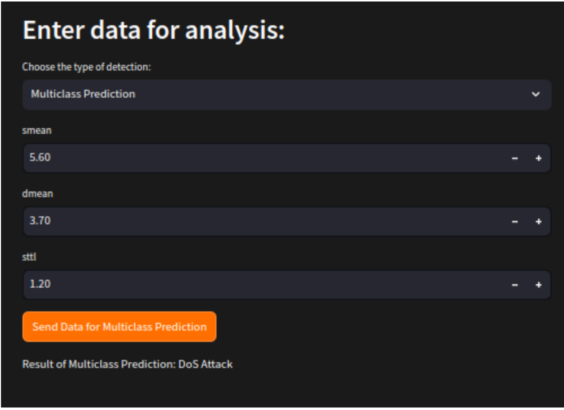


Figure 5.3: Data Analysis StreamlitAPI Screenshot

5.3.2 Communication with FastAPI

The Streamlit frontend sends HTTP POST requests to the FastAPI backend in order to receive predictions. The API sends data to the FastAPI backend, which processes it and generates a prediction, each time the user enters new data or the live detection is updated. The dashboard then shows this prediction.

5.3.3 Final Layout and Usability Enhancements

The dashboard's final design prioritized user experience. The real-time line chart and other important data visualization components were positioned front and center so that users were able to quickly understand the state of the system. To make the prediction results more understandable, input fields were logically grouped together and backend feedback was shown clearly.

5.4 Conclusion

Streamlit was used for front-end visualization and FastAPI for back-end processing to create a secure and efficient system for anomaly identification in real-time. The asynchronous features of FastAPI guaranteed prompt and effective backend prediction, and Streamlit offered clients a user-friendly interface with speed to interact with the system and view detection results. Combining these two tools worked well in achieving a balance between usability and performance.

Chapter 6

Feature Engineering and Data Preparation

6.1 Testbed for Cyberattack Automation and Feature Extraction

The creation of a cyberattack-safe environment and automated feature extraction are essential steps in the continuous adjusting of anomaly detection models, ensuring a high identification and response rate to attack patterns. The process of implementing *testbed* will be described in this section, along with the reasons behind the decisions taken and the fixes for the primary engineering issues.

6.1.1 General Objectives

This *testbed*'s main goal is to automatically generate attacks that can be used to continuously test the created API and retrain machine learning models, enabling real-time visualization of the results. In this process, network packets are captured, various attacks (such *scanning* and *DoS*) will be carried and detected, and the key features related to these events are extracted. In addition, the system should autonomously produce log files including comprehensive details about every attack executed, allowing further examination and retraining of detection models.

6.1.2 Structure of Testbed

The design of *testbed* makes it possible to execute several attack scenarios in a cyclical and controlled way.To simulate attacks over an extended length of time, the user defines the entire duration of the experiment and selects the situations at random.

Each scenario consists of three main steps:

- **Environmental Preparation:** The environment is cleared of any traces of earlier scenarios, such as running processes from earlier attacks, before every attack. Interference is avoided and data integrity is ensured.
- **Packet Capture:** The network traffic created during the attack is collected using the `tcpdump` tool. Depending on the situation, a predetermined random time frame is used for this capture.

- **Attack execution:** For each specific type of attack, such as a Denial of Service or network scan (*scanning*), is executed for every situation. To improve the variety of the generated data, the attack type and parameters are chosen at random.

6.1.3 Technical details

Directory Structure

The data has been split into different subdirectories for each sort of attack, with names that match to the attack scenario and type. As a result, accessing and analyzing the data is made simpler. In addition, every attack execution generates two different kinds of files:

- **PCAP File:** Contains packets captured during the attack.
- **CSV File:** It has detailed information about the attack, such as the parameters used, the duration, and any errors found.

Packet Capture Process

The `tcpdump` command is used to capture packets and enables selective recording of network traffic on the selected interface. In order to allow for the simultaneous execution of other commands, like executing the attack, the process is launched asynchronously. To ensure data diversity, each capture is limited to a random period of 60-150 seconds [58].

Attack Execution

The attacks are performed based on the selected scenario, each with specific tools and configurations to simulate real-world threat behaviors. Below are the two main types of attacks and how they are executed:

- **Scanning:** In a network, scanning is used to identify open ports and vulnerable devices. `Nmap`, a common network scanning tool, is used for this attack. Many scanning techniques are randomly chosen to simulate multiple types of scanning:
  - `-sn`: Ping scan to discover live hosts without port scanning.
  - `-F`: Fast scan to quickly identify open ports by scanning only the most common ports.
  - `-A`: Aggressive scan, which includes OS detection, version detection, script scanning, and traceroute.
  - `-p 1-100`: Scanning the first 100 ports to target specific services.
  - `-T4`: High-speed scan to reduce execution time, increasing the chances of detection.
  - `-sS`: Stealthy SYN scan, often used in real-world attacks due to its ability to bypass certain firewalls.
  - `-O`: Operating system detection based on TCP/IP fingerprinting.
  - `-sU`: UDP scan to find services running over the User Datagram Protocol.

  These parameters are randomly combined to generate diverse scan traffic and different scanning intensities, making it more difficult to detect and prevent [59].

- **Denial of Service (DoS):** Denial of Service attacks aim to overwhelm the target with a flood of traffic, rendering it unable to respond to legitimate requests. Several tools and techniques are used:
  - `hping3`: A packet crafting tool that can generate TCP, UDP, ICMP, or RAW-IP packets. It is used to create a high volume of traffic, simulating SYN floods, ICMP floods, or fragmented packet attacks [[60]].
  - `ping`: A basic network utility that sends ICMP Echo requests. It can be used in flood mode to overwhelm a target with continuous requests, leading to a denial of service [61].
  - `nping`: Other flexible packet generator capable of crafting packets for various protocols. It can simulate both DoS and Distributed Denial of Service attacks by sending large volumes of requests with specific intervals and payloads [59].

  By varying the packet types, payload sizes, and traffic volumes, different DoS attack patterns are simulated to stress the target in multiple ways, making it difficult to distinguish legitimate from malicious traffic.

Detailed Event Log

A complete logging system that captures every important detail from each attack, including the length, the parameters used, and any mistakes that happened, has been established to make it easier to analyze the data that is created later.

6.1.4 Conclusion of the Testbed

In the end, this testbed's implementation for automating cyberattacks and obtaining features is an essential step for the ongoing development of anomaly detection models. It is possible to produce useful data that will enable the testing of the developed API and the efficient retraining of machine learning models by simulating different attack scenarios.

In-depth event logging and packet collecting ensure every attack is recorded and studied, which improves the comprehension of attack dynamics and system security.By using this method, it has the goal to improve the pace that attack patterns are identified as well as the ability to react to these threats promptly, increasing the anomaly detection system's efficiency in real-life situations.

Chapter 7

Results and Discussion

7.1 Introduction

The efficiency of machine learning models for classifying network traffic will be addressed in this section in both binary and multi-class classification scenarios.In the beginning, it will be shown the outcomes of the individual models, highlighting their individual efficencies and the identification of the most suitable candidates for building ensemble models. The benefits of combining several models to detect anomalies will be explored in a detailed comparison between the individual and ensemble models. Finally, the system's development and implementation constraints will be addressed, with a focus on how these limitations may affect the application of the results and the potential for additional research in this area of study.

7.2 Evaluation of Model Performance

Finding the best approaches for binary and multiclass classification involved evaluating the performance of each individual model. Based on what was learned from *Chapter 4*, the following section shows the results of the various models and provides support for the selection of the best ones for the ensemble.

7.2.1 Individual Models

Using the UNSW-NB15 dataset, this subsection explores the individual models used in binary and multi-class classification.

Binary Classification

The **dataset UNSW-NB15** was selected as the base dataset for the API's implementation in binary classification since it has been shown to be reliable in identifying anomalies in network traffic.Not every model that was tested, however, was suitable for developing an ensemble.

The **Logistic Regression** and **SGD Classifier** models were eliminated because of their poor performance, as stated in *Chapter 4*. For example, the SGD Classifier model showed a negative value for the R2 and a *accuracy* of only 84.38%, indicating issues with convergence and a high error rate.

On the other hand, with *accuracy* over 99% for both **Min-Max normalization** and **Z-Score normalization**, the **KNN**, **Random Forest**, **Decision Tree** and **MLP Classifier** models

showed better performances.These four models were ideal choices for the ensemble stage since they showed strong generalization and anomaly detection skills.

Plots showing the efficiency of each of the models appear below.

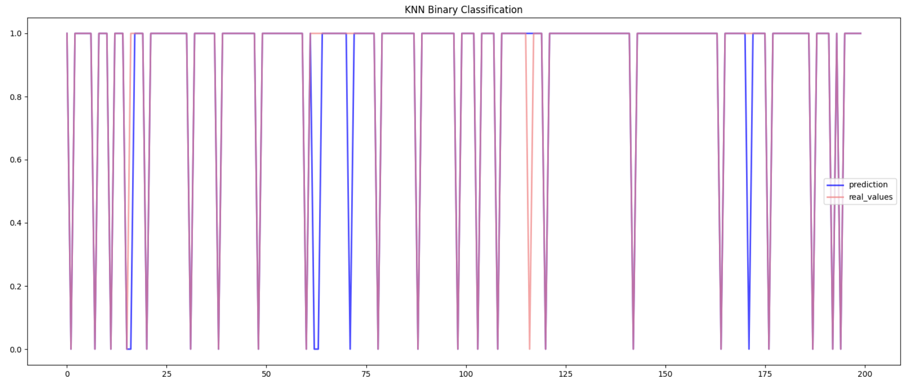


Figure 7.1: Performance of the KNN model on the UNSW-NB15 dataset for Binary Classification

With an *accuracy* of 99.11%, the **KNN** (Figure 7.1) showed the ability to identify anomalies. For most of the samples, there is nearly a perfect match between the predictions and the actual values, as seen by the graph's closely spaced lines representing the actual and predicted values. It also means that the majority of abnormal and normal traffic cases can be accurately classified by the model.

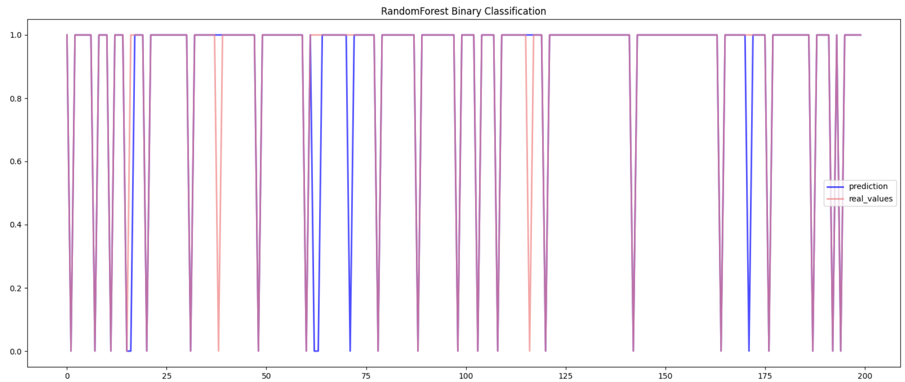


Figure 7.2: Performance of the Random Forest model on the UNSW-NB15 dataset for Binary Classification

The Random Forest model (Figure 7.2) achieved an accuracy of 99.09% and a R^2 value of more than 92%, showing outstanding results.As we can see from the graph, there is a high success rate in the binary classification because most of the predictions (blue lines) are superimposed on the real values (red lines).

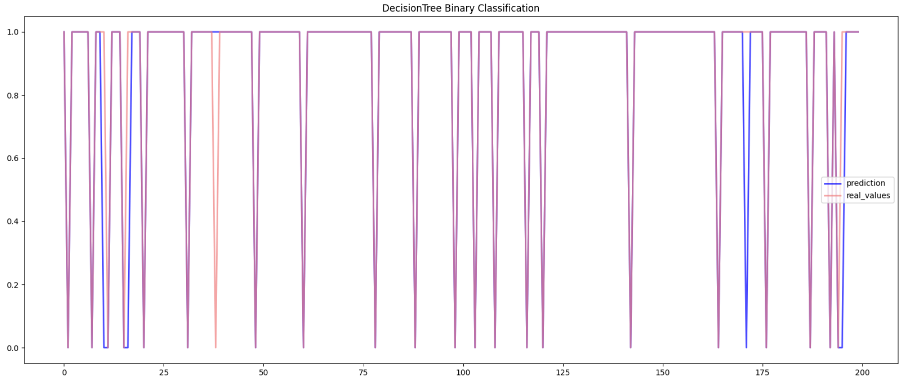


Figure 7.3: Performance of the Decision Tree model on the UNSW-NB15 dataset for Binary Classification

With an accuracy of 99.09%, the Decision Tree (Figure 7.3) produced successful outcomes.The graph shows that there is strong alignment between the model and the data because the predictions (blue lines) are mostly superimposed on the actual values (red lines).The fact that the differences between expected and actual values are small and limited, however, supports the ability of the model to identify simple patterns.Even though this model is not as advanced as Random Forest, its almost identical performance confirms its reliability and justifies its inclusion in the ensemble stage since it can produce predictions quickly and accurately with minimal computing complexity.

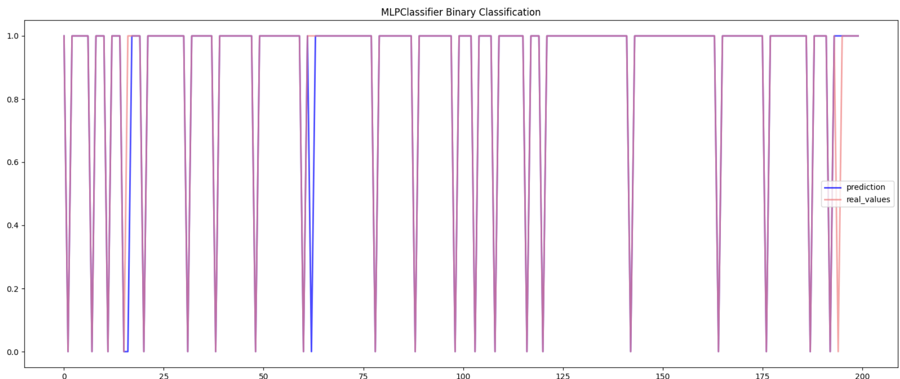


Figure 7.4: Performance of the MLP Classifier Model on the UNSW-NB15 dataset for Binary Classification

With a high accuracy of 93.78%, the MLP Classifier (Figure 7.4)produced acceptable results.On the other hand, in contrast with Decision Tree and Random Forest, the model's predictions on the graph showed less perfect alignment with the real values (red lines).The model has problems correctly identifying multiple trends in the data, as seen by the areas where the predictions (blue lines) vary greatly from the actual values, especially between indices 50 and 150.This is explained by the fact that MLP is more complicated than other neural networks because it uses several layers of neurons and requires more hyperparameter

tweaking. However, the MLP's overall performance is reliable and acceptable, and its ability to capture non-linear patterns is very remarkable.

Multi-Class Classification

The **dataset UNSW-NB15** was once again chosen as the dataset for model training in the multi-class classification for its detailed labeling of different attack types. Similar to the binary classification, not every model was considered viable for the final ensemble.

As stated in *Chapter 4*, the models **SGD Classifier** and **Logistic Regression** have been removed based on their weak results. More specifically, the SGD Classifier performed badly, showing an accuracy of 90.28% and a negative R2 value. The logistic regression model's accuracy was about 90%, but it was still lower than the other models', therefore it wasn't suitable for the ensemble phase, although it was slightly better.

Yet, significantly better results have been achieved by the models **KNN**, **Random Forest**, **Decision Tree**, and **MLP Classifier**. Using Min-Max normalization, the **KNN** model achieved an accuracy of 97.10%. The **Random Forest** and **Decision Tree** models followed with accuracies of 97.19% and 96.88%, respectively. In addition, the accuracy of the **MLP Classifier** was 95.52%, indicating good performance. For this reason, these four models were selected for the ensemble step due to to their high degrees of generalization in the context of multiple classes.

The **plots** showing the individual performance of each of the selected models are shown in the following parts.

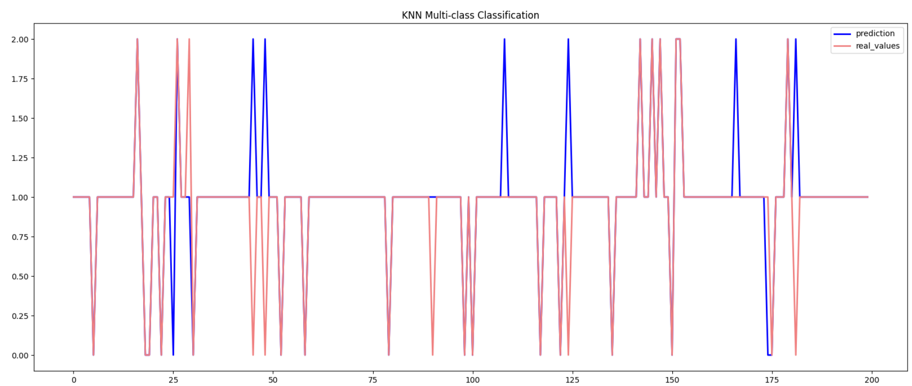


Figure 7.5: Performance of the KNN model on the UNSW-NB15 multi-class dataset.

With an accuracy of 97.10%, the KNN model (Figure 7.5) showed good performance in identifying differences between multiple attack classes. Upon evaluating the plot, we notice some instances where the actual class labels *(in red)* differ significantly from the predicted label *(in blue)*. There are visible instances of misclassification, especially where the true values vary between classes, yet the model typically correctly corresponds with real values.

These variations show that although KNN is good at multi-class classification, some classes are difficult to handle, which leads to prediction gaps. This may be explained by the dataset's basic features or by KNN's sensitivity to noise or class imbalances. The high accuracy

suggests the model's overall generalization ability is still strong, but further improvements might improve performance in the areas where the differences are the most visible.

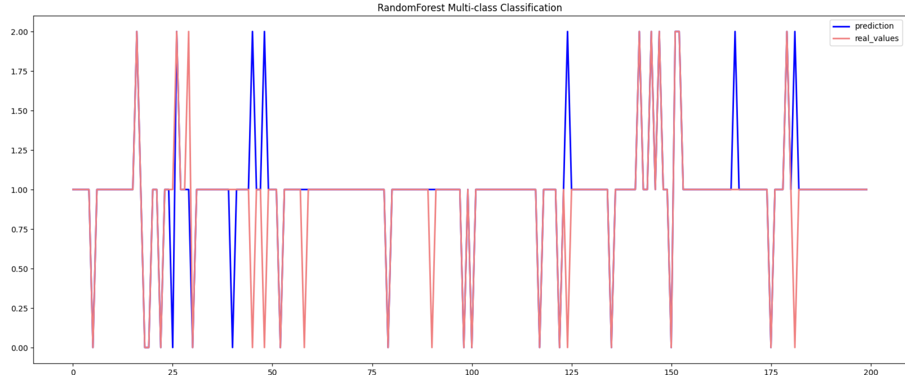


Figure 7.6: Performance of the Random Forest model on the UNSW-NB15 multi-class dataset.

Similar to the KNN model, the Random Forest model (Figure 7.6) achieved an accuracy of 97.19%. But looking at the graph, we can see that the real values *(in red)* and the predictions *(in blue)* often differ, especially in some areas.

The Random Forest model uses multiple decision trees to capture complex structures of data, which shows reliability regardless of these inconsistencies.

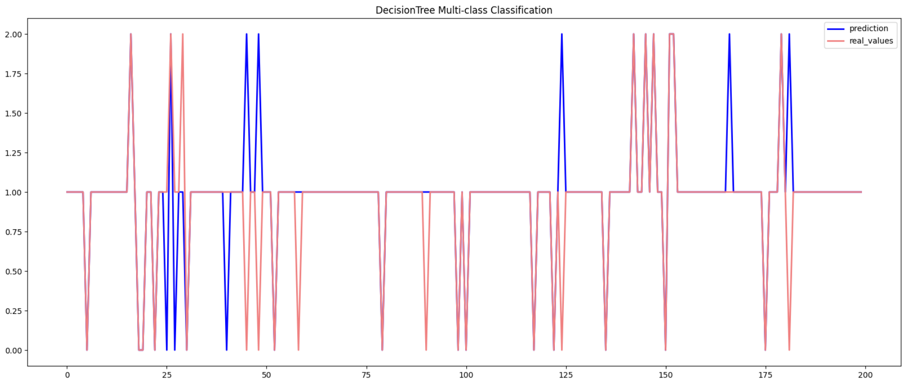


Figure 7.7: Performance of the Decision Tree model on the UNSW-NB15 multi-class dataset.

With a high accuracy of 96.88%, the Decision Tree model (Figure 7.7) demonstrated its efficiency in multi-class classification. The graph shows that although the model is able to accurately represent many of the variations across classes, there are a number of areas, especially near class transitions, where the predictions *(in blue)* differ from the actual values *(in red)* .

Despite these limitations, the Decision Tree performs well because of its simple structure, allowing it to identify significant patterns in the dataset.

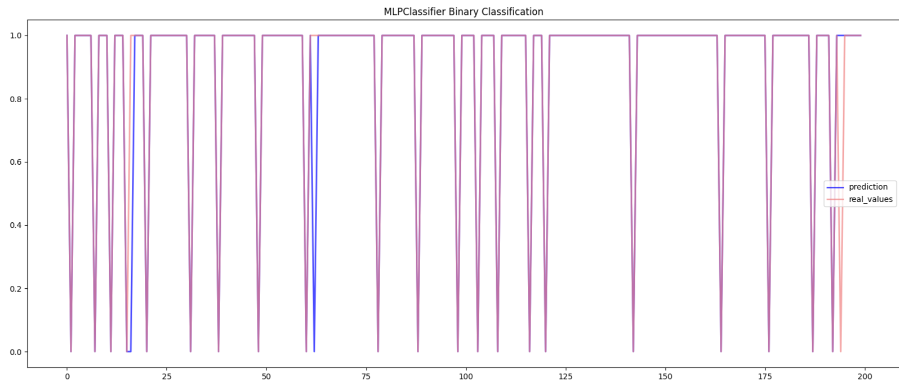


Figure 7.8: Performance of the MLP Classifier on the UNSW-NB15 multi-class dataset.

With an accuracy of 95.52%, the **MLP Classifier** (Figure 7.8) showed a high overall performance. The model performed well in a multi-class scenario mainly because of its ability to capture non-linear links between features. While the model's performance is strong, an in-depth analysis of its behavior shows that there are occasionally differences between expected and actual values, particularly in areas with sharp peaks. These findings indicate that the model might not be able to adapt to sudden changes in the data.

7.2.2 Ensembled Models

After reviewing the best individual models, we evaluated the performance of the ensemble models.

The **KNN**, **Random Forest**, **Decision Tree**, and **MLP Classifier** models were used to create the ensemble model for **binary classification**.The ensemble model's confusion matrix for binary classification is presented below, showing the accuracy with which it recognizes anomalies.

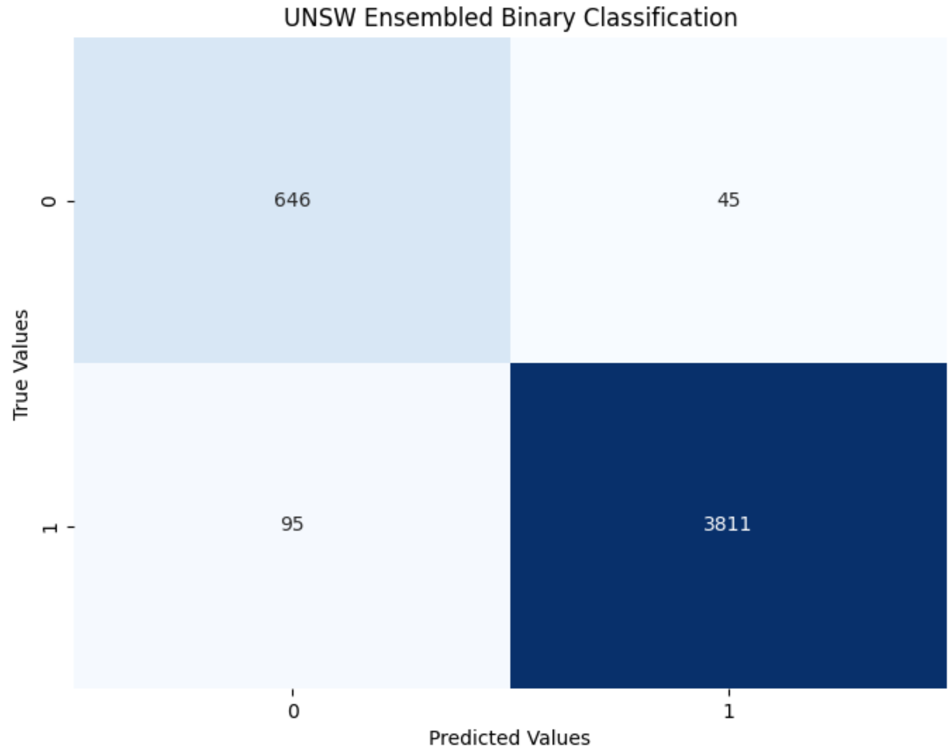


Figure 7.9: Confusion Matrix of the Ensembled Model for Binary Classification

The ensemble model has outstanding results, shown in the confusion matrix (Figure 7.9). With just 2.43% of false negatives (95 instances), the model accurately identified 97.56% of the positive class incidents (3811 out of 3906). Comparably, there was minimum false positive rate (45 instances) and high accuracy of 93.48% in labeling negative instances (646 out of 691). Compared to individual methods, the outcomes indicate that the ensemble provides a large reduction in classification errors and superior performance for anomaly identification.

The ensemble model with the best performances for **multi-class classification** has also been created, including **KNN**, **Random Forest**, **Decision Tree**, and **MLP Classifier**.The confusion matrix generated shows how effectively the model can distinguish between different types of attacks, and it can be seen below.

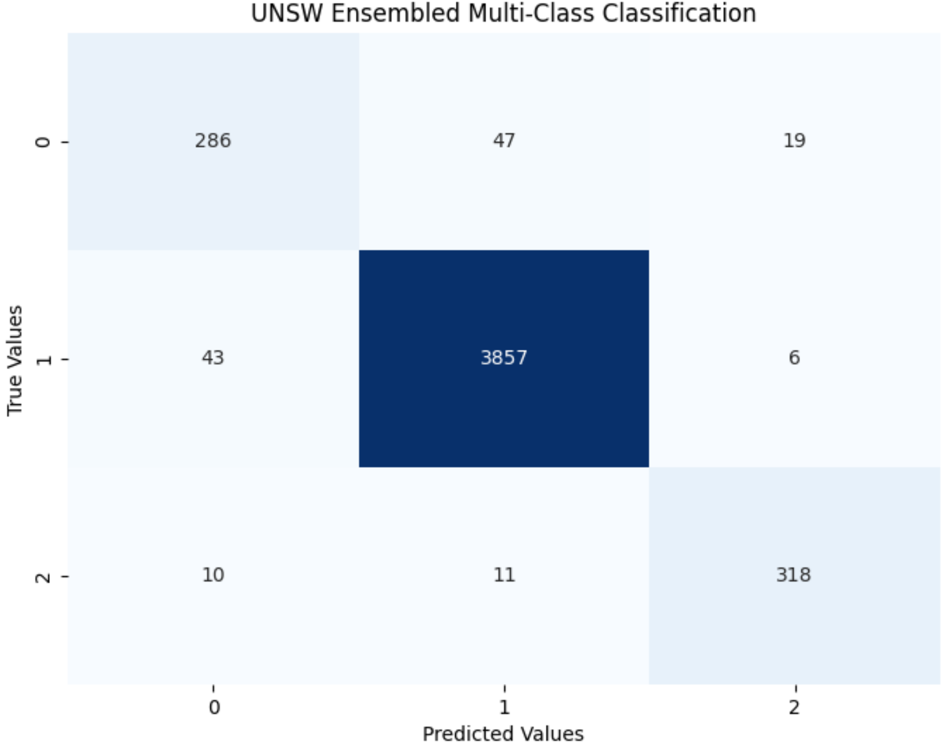


Figure 7.10: Confusion Matrix of the Ensembled Model for Multiclass Classification

The ensemble model performs well in a multi-class scenario, as shown by the confusion matrix (Figure 7.10). With 99.1% accuracy for the majority class (class 1-Benign Traffic) (3857 out of 3906), the model showed astounding precision. The minority class detection (82.3% accuracy for class 2(Scanning) and 77.7% accuracy for class 0(DoS) was identically good. The results show an important increase in the detection of the least represented classes compared to individual approaches, even though the model's difficulties with differentiating between classes 0 and 1. This suggests that the ensemble technique has improved the reliability of multiclass classification.

The final results and charts in this part highlight the importance of integrating models, showing that the ensemble method significantly increased accuracy and adaptability.

7.3 Testbed Evaluation

The objective of implementing the testbed was to measure the effiency of the system by creating an environment that mirrored actual cyberattack scenarios. The testbed's main objective was to create a controlled environment where benign traffic could be generated randomly alongside several attack types, including *Denial of Service* and *Port Scanning*.

However, it was not possible to properly evaluate the *ensembled* models' performance in real-time using this testbed due to difficulties in preparing the data required for the models. Consequently, it was not possible to carry out the originally intended evaluation of the

system's precision and effiency in identifying anomalies based on network traffic patterns and attack data. This restriction made it more difficult to confirm how well the models performed in actual attack scenarios.

In spite of this, the testbed's ability to replicate a nearly realistic environment made it an important starting point for future tests. The testbed will be a useful tool for evaluating how well the integrated models perform in detecting anomalies and classifying various types of traffic once the preprocessing problems are fixed.

7.4 Application Input

This section will go over the frontend and backend script-based handling of the input data (vectors) by the application to identify anomalies in security networks.The data which the interface provides, how it is processed, and how detection results are obtained will be the primary topics of concern.

7.4.1 Input Data Format

Users can evaluate anomalies by entering feature vectors produced from network traffic into the application interface.Multiclass prediction and binary detection are the two types of analysis. Each one has a number of features and/or vectors, and binary detection includes the following:

- **ct_srv_dst:** The total amount of different service destinations seen in the current flow is represented by this feature.
- **trans_depth:** This feature, which indicates the number of layers or packet structure segments, shows the depth of the transmission.
- **synack:** The SYN-ACK response time during the TCP handshake is measured by this feature.
- **tcprtt:** The time it takes for a packet to go from the source to its destination and back is known as the TCP Round Trip Time.
- **ackdat:** The acknowledgment of received data packets is captured by this feature.
- **sttl:** The value given in the IP header named Source TTL determines how long a packet can stay on the network before being deleted.
- **dttl:** The maximum amount of time a packet can stay in the destination network is shown by the destination TTL.
- **ct_state_ttl:** This feature gives details about the connection's status and the amount of time left until the packet is eliminated. It is a combination of connection state and TTL.

On the other hand, a multiclass prediction only has three features, which are as follows:

- **smean:** This feature shows the average size of packets that the source has sent over a given amount of time.
- **dmean:** This feature shows the average size of the packets that the destination received in the same amount of time.

- **sttl:** A field named Source Time to Live in the IP header indicates how long a packet can remain in the network at its longest.

Following the feature selection step, these features are extracted from the *.csv* files; the features with the highest correlation will be included in the *.csv* file that will be used for splitting the training and test data of the model.These vectors are manually entered using an interface panel.

7.4.2 Data Analysis Process

FastAPI is used to implement the application's backend. It processes the input data received by a REST API, returns the anomaly predictions to the graphical interface developed with Streamlit, and then stores and processes the data.

Users can choose between Binary Detection and Multi-class Prediction once the data is gathered in the frontend.Upon selection, the input data will be stored in JSON and sent to the correct endpoint: either the Multiclass Prediction API *http://localhost:8000/predict_multiclass/* or the Binary Detection API *endpoint:http://localhost:8000/detect_binary/*.

With the input data given in the request body, the frontend sends an HTTP POST request to the matching URL. Pre-trained models, saved in.pkl files, are used to process the input on the backend and return a classification of the traffic as normal or anomalous.

The system provides Benign(0) or Malign(1) as the basic classification for Binary Detection.Benign, Scanning and DoS are the classes that can be selected in Multiclass Prediction. After converting the numerical labels(0, 1, 2) back into their appropriate classes, the backend replies to the frontend.

7.4.3 User Interface

Streamlit's graphical user interface is made to display predicted results in real time, allowing data visualization and ongoing anomaly detection monitoring. It is set up with three primary tabs to provide an easy and objective user experience.

Figure 7.11: Streamlit's Frontend Tabs

The impact of the anomalies discovered is shown in real-time on a graph in the 'Live Detection' tab, and it can change between the 'Benign,' 'Scan,' and 'DoS' categories. This means that users will be able keep up to date on unusual activity as it happens, though the exact results could change depending on how accurate the model is.

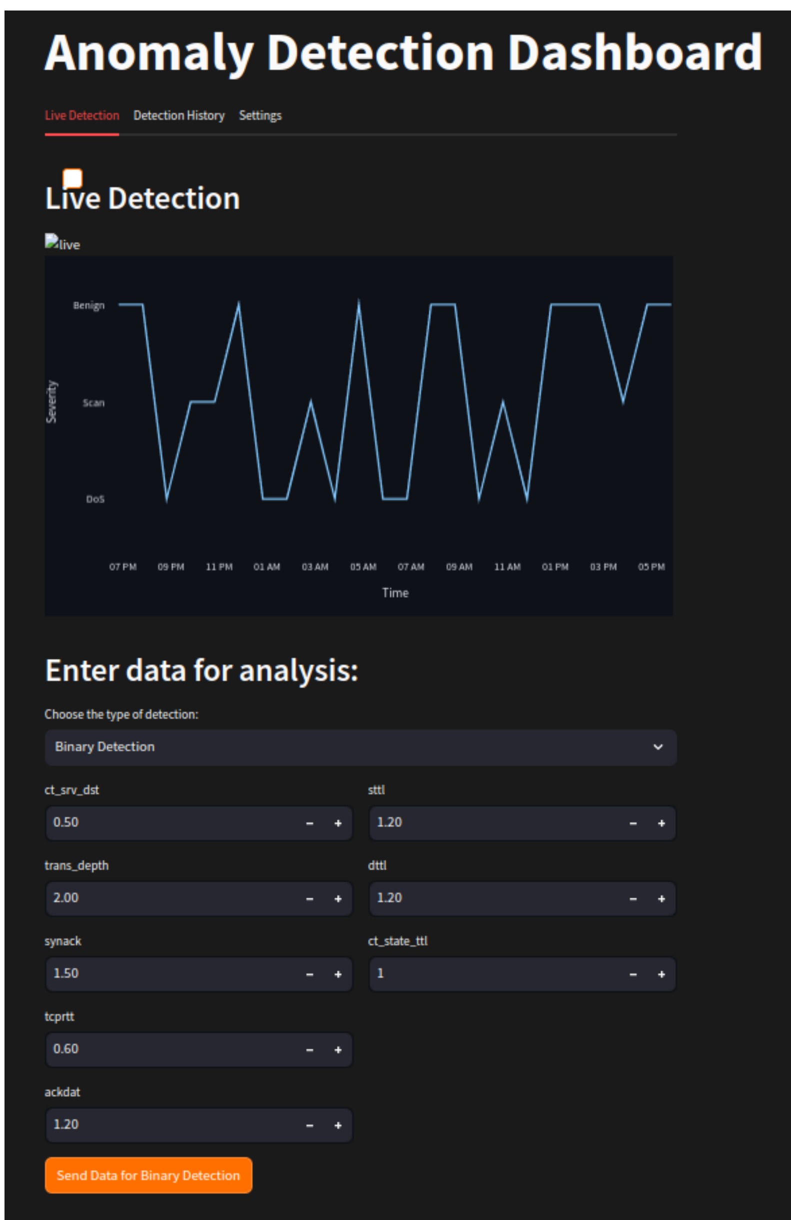


Figure 7.12: Caption

The user can monitor past events and obtain an expanded view of anomaly patterns over time by navigating to the 'Detection History' tab, which displays previous records of detections.

The code creates interactive graphs using the Altair library, which can clearly visualize patterns and occurrences in network traffic. While exact results can still be confirmed, this combination of features tries to make it simpler to monitor and analyze anomalies found in the system.

7.4.4 Correlated Feature Analysis for Binary and Multi-Class Classification

A closer look of every feature with a high correlation for both binary and multi-classification will be performed in this subsection. The analysis will be separated into two parts: a multi-class analysis that focuses on all of the attack types(DoS, scanning, and normal traffic)

and an analysis of the features for binary classification (anomaly and normal traffic).

Binary Classification

This section's main objective is to examine the most relevant features for identifying between the anomaly traffic *(label_1)* and normal traffic *(label_0)*. Graphics like histograms and box plots were used in this analysis, in addition to a table that included statistical information for every feature.

Normal Traffic (label_0):

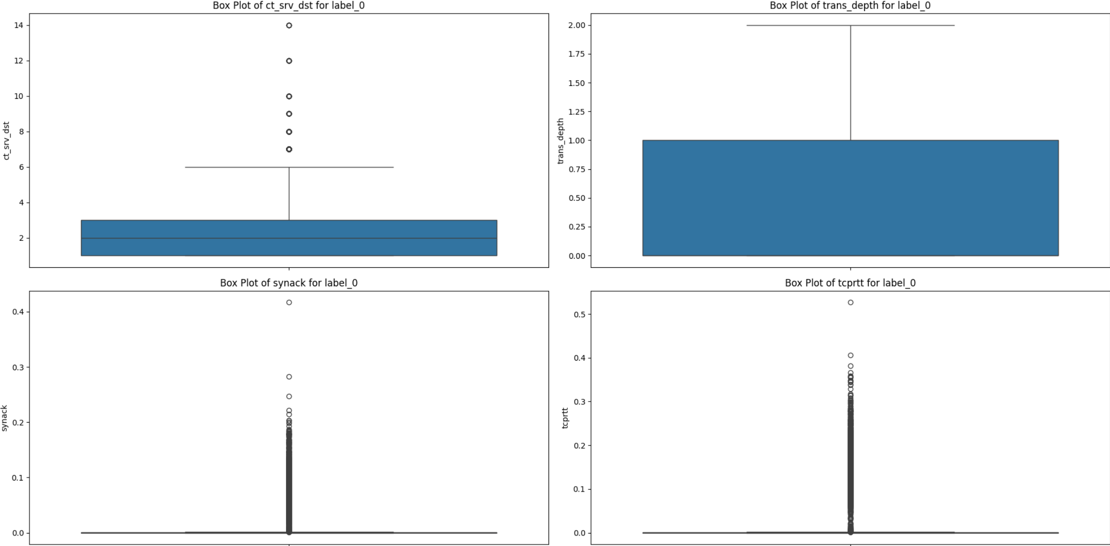


Figure 7.13: Boxplots of variables for Label 0 - Part 1

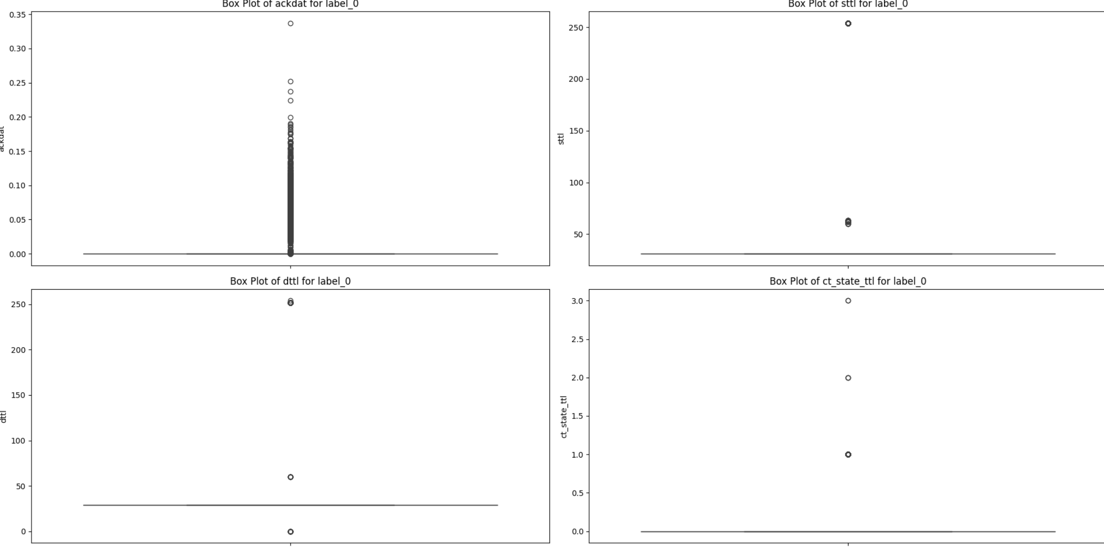


Figure 7.14: Boxplots of variables for Label 0 - Part 2

Boxplots and frequency distributions are used in the analysis of the graphs for *label_0*. These plots are important for locating outliers, asymmetry in the distributions, and potential value ranges that can be utilized to split or label the data.

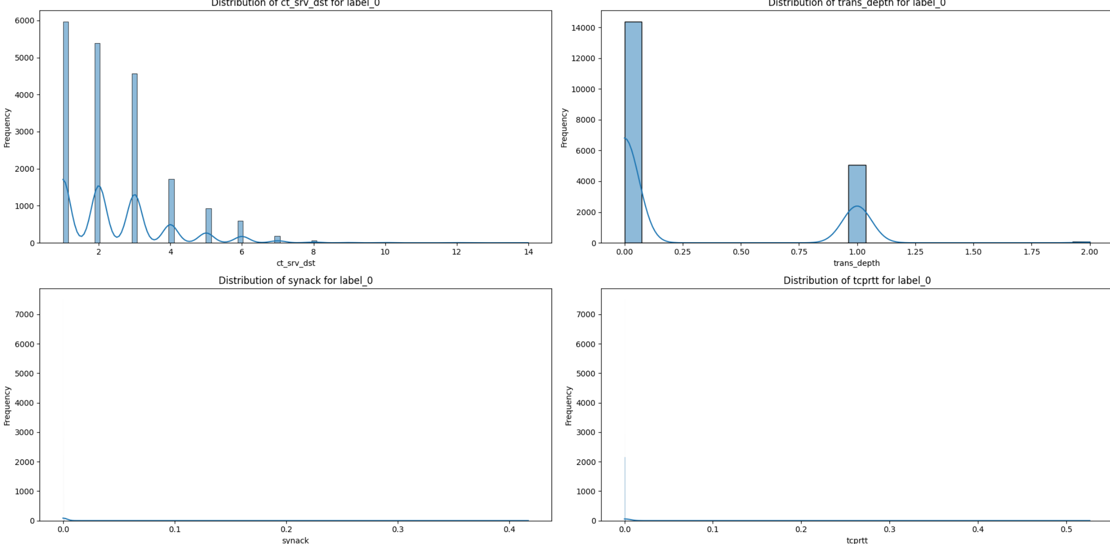


Figure 7.15: Variable Distributions for Label 0 - Part 1

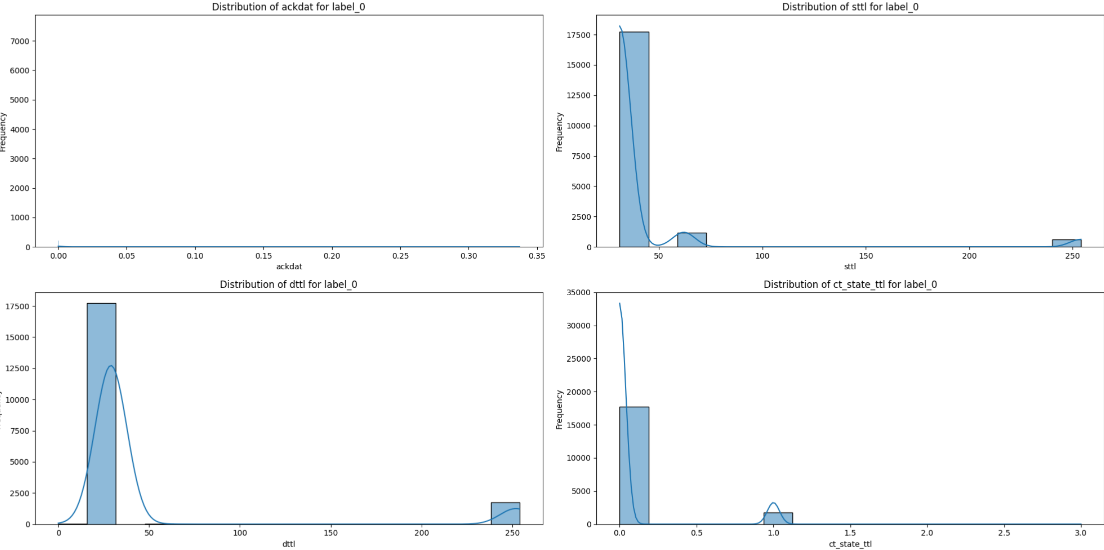


Figure 7.16: Variable Distributions for Label 0 - Part 2

Most of the data is distributed between 1 and 3, as seen in the boxplot displayed by the *ct_srv_dst* variable, which has a median around 2. There are only a few outliers with a maximum of 14 above 6. This concentration of low values is consistent with the pattern seen in the *label_0* events and suggests that most connections to the same service are reduced in normal situations. Higher values, particularly those surpassing 6, may suggest unusual or possibly malign behavior.

The *trans_depth* boxplot shows a median of about 1. No obvious outliers can be seen, and the majority of the data is concentrated between 0 and 1.25. The distribution shows asymmetry, showing a peak at approximately 1, indicating that the average depth of HTTP transactions in *label_0* events is in proximity to 1.

The *synack* variable has a large concentration of values close to 0, with a small number of outliers in the 0.1–0.4 range. The distribution shows that a lot of events have essentially zero reaction times.According to this pattern, *label_0* events typically involve quick connections, and values greater than 0.1 might point to anomalies that need additional study.

Many of the values are near to 0, with a few outliers reaching 0.5, as the *tcprtt* boxplot shows. This suggests that *label_0* events usually have very low RTTs, indicating connections with low latency or local connectivity. Values greater than 0.1 are uncommon and may indicate unusual events.

A big concentration of values near 0 also exists in the *ackdat* variable, with a few outliers falling between 0.2 and 0.35. A pattern of agile communication is indicated by the distribution, which reveals that low delays define most events in *label_0*.

The median for *sttl* is extremely low, with outliers found at 50 and 250.With a peak at 0, most of events are concentrated between 0 and 75, suggesting that packets with short TTL are more common in *label_0*. Anomalies with values close to 250 are rare and indicate suspicious traffic.

The *dttl* variable shows a higher level of values close to 0 with a few large outliers at 250 and a few around 50. This distribution indicates that the TTL fluctuates little during *label_0* events.Unexpected actions may be indicated by outliers between 50 and 250.

In the end, the *ct_state_ttl* boxplot displays a median around 0 and a few outliers as high as 3. This variable's value is typically extremely low, showing that there aren't many state transitions during *label_0* connections. Anomalies with values greater than 1 deserve to be tracked in order to spot any of the potentially suspicious traffic patterns.

Attack Traffic (label_1):

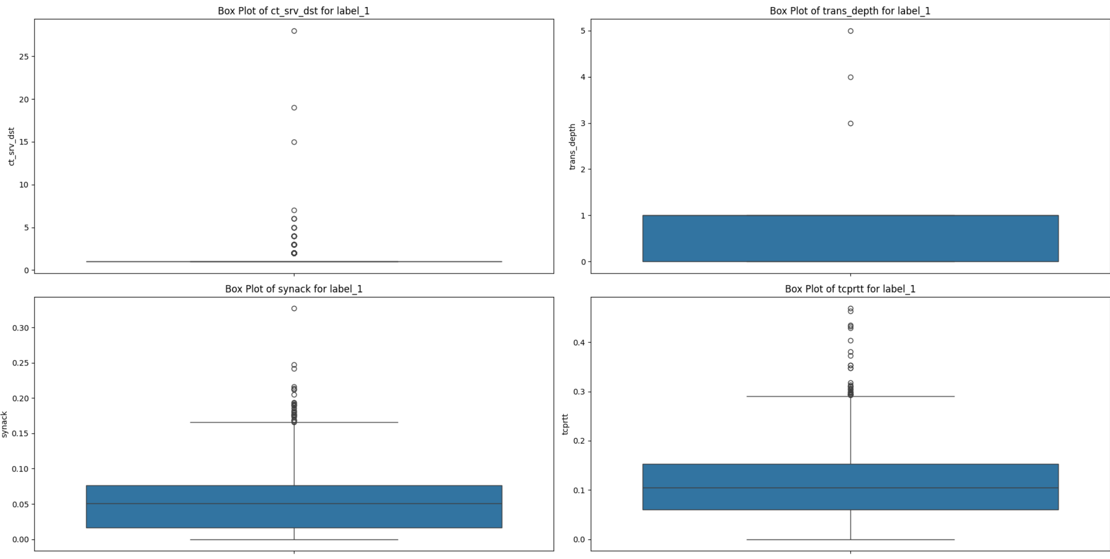


Figure 7.17: Boxplots of features for Label 1 - Part 1

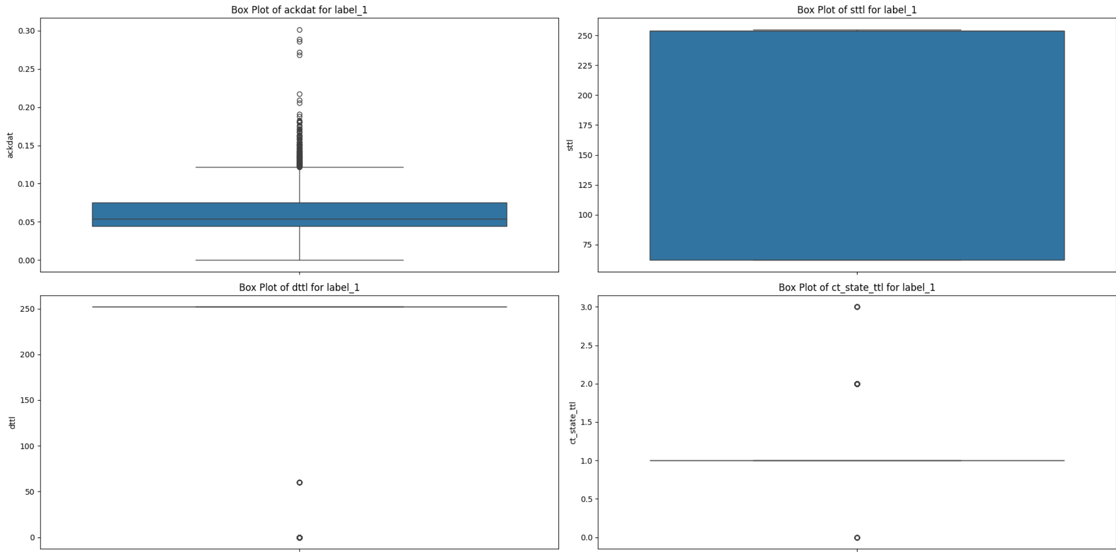


Figure 7.18: Boxplots of features for Label 1 - Part 2

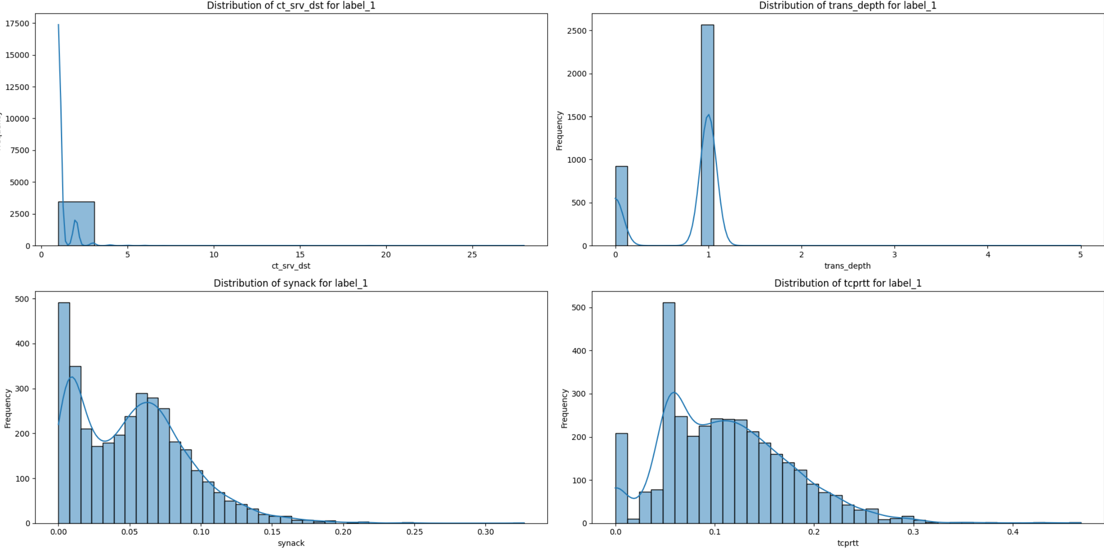


Figure 7.19: Features Distributions for Label 1 - Part 1

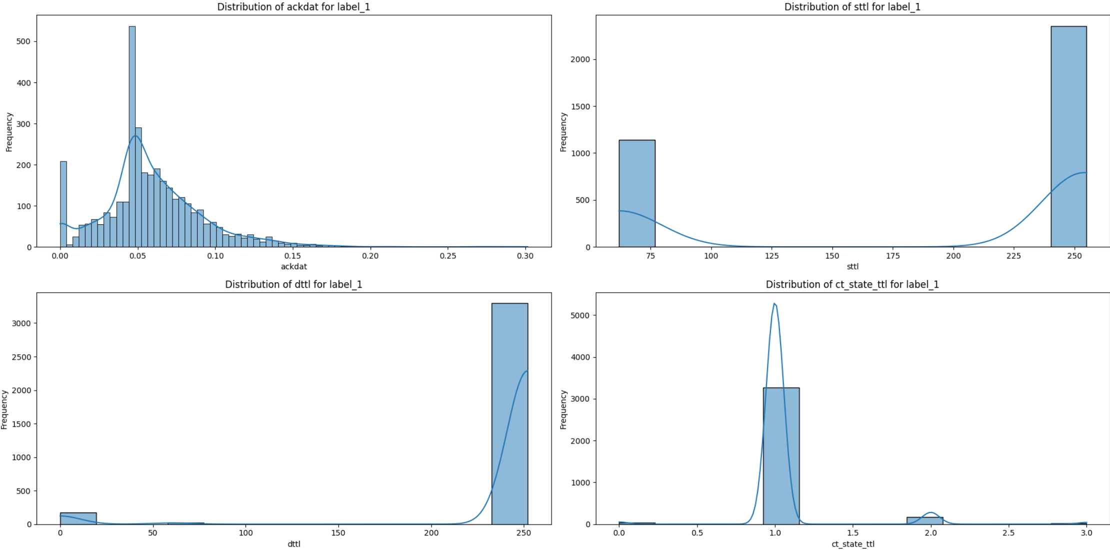


Figure 7.20: Features Distributions for Label 1 - Part 2

A highly dispersed distribution is shown in the boxplot of *ct_srv_dst*, with most of values concentrated close to zero and a few outliers reaching 25. This suggests that while there can be anomalous behavior with clearly greater values, most of attack cases had a low number of target services. This pattern is backed up by the frequency distribution, which has a long tail that goes up to 25 and a high concentration of values close to zero, indicating that these outliers could indicate rare but effective attacks.

The *trans_depth* boxplot shows a low transaction depth during most attacks, with a concentration of values close to 0 and a few outliers above 1. This analysis is confirmed by the frequency distribution, which shows that deeper transactions are rare. Transaction depths of 0 or 1 are usual for attacks; higher values are exceptions that might point to unusual events.

The boxplot for *synack* shows that most of the values fit between 0.05 and 0.10, with outliers as high as 0.3.It also means that, with a few higher exceptions, attacks falling under this category typically have low TCP recognition times. The frequency distribution reflects this observation, with a significant density in the 0.05 to 0.1 range, confirming that these times are common in attacks, while values above 0.1 are rarer.

A concentration of values around 0.1 is seen in the boxplot of *tcprtt*, with some outliers reaching as high as 0.4.It also means that the round-trip times of most attacks against *label_1* are relatively short (TCP). This is verified by the frequency distribution, which shows a high density of values at 0.1 and a tail that reaches 0.4, suggesting that values above 0.1 might be uncommon.

Similar to *synack*, the boxplot of *ackdat* shows a distribution with most values falling between 0.05 and 0.10 and outliers reaching up to 0.3. This pattern is backed by the frequency distribution, where most of attacks show slow response times, while higher values point to a few exceptions that might be outliers.

With no significant outliers and most of values gathered around 250, the boxplot of *sttl* is very stable.It means that this TTL value remains constant in almost all of the attacks. This conclusion is confirmed by the frequency distribution, which shows a very high density of

values around 250, indicating that this is an usual and consistent behavior across all types of attacks.

The *dttl* boxplot shows that the majority of values center around 0, with outliers reaching as high as 250. This suggests that while there are rare cases of larger target TTLs, most attacks have low target TTLs. With a long tail that reaches up to 250 and a concentration of values close to zero in the frequency distribution, it is possible that high TTL values are an indication of strange events.

At last, a similar pattern can be seen in the boxplot of *ct_state_ttl*, where most of the values are grouped around 1. This shows that, under most attacks, the state transition TTL is minimal. This conclusion is supported by the frequency distribution, which shows a strong concentration of values around 1 and very few exceptions that go above it. Values higher than 1 may indicate unusual patterns.

Table 7.1: Feature Value Ranges by Traffic Type - Binary Classification

Feature	Traffic Type	Value Range
ct_srv_dst	Normal Traffic (label_0)	[0, 14]
	Anomaly Traffic (label_1)	[0, 25]
trans_depth	Normal Traffic (label_0)	[0, 1.25]
	Anomaly Traffic (label_1)	[0, 1]
synack	Normal Traffic (label_0)	[0, 0.4]
	Anomaly Traffic (label_1)	[0.05, 0.3]
tcprtt	Normal Traffic (label_0)	[0, 0.5]
	Anomaly Traffic (label_1)	[0.1, 0.4]
ackdat	Normal Traffic (label_0)	[0, 0.35]
	Anomaly Traffic (label_1)	[0.05, 0.3]
sttl	Normal Traffic (label_0)	[0, 250]
	Anomaly Traffic (label_1)	[250, 250]
dttl	Normal Traffic (label_0)	[0, 250]
	Anomaly Traffic (label_1)	[0, 250]
ct_state_ttl	Normal Traffic (label_0)	[0, 3]
	Anomaly Traffic (label_1)	[1, 1]

Binary Detection Tests

Multiple tests were done to test the model's ability to distinguish between normal and anomalous traffic. The tests all returned the message "Anomaly Detected," so the results were not what was expected. The fact that the predictions were consistent suggests that there may have been issues, such as inconsistent input data or a potential problem with the model's ability to make generalizations.

Multi-Class Classification

The second part of this analysis, aimed at normal traffic, DoS attacks, and scanning, examines multi-class classification. The multiple attack classes appear in an identical analysis structure as in binary classification.

Normal Traffic:

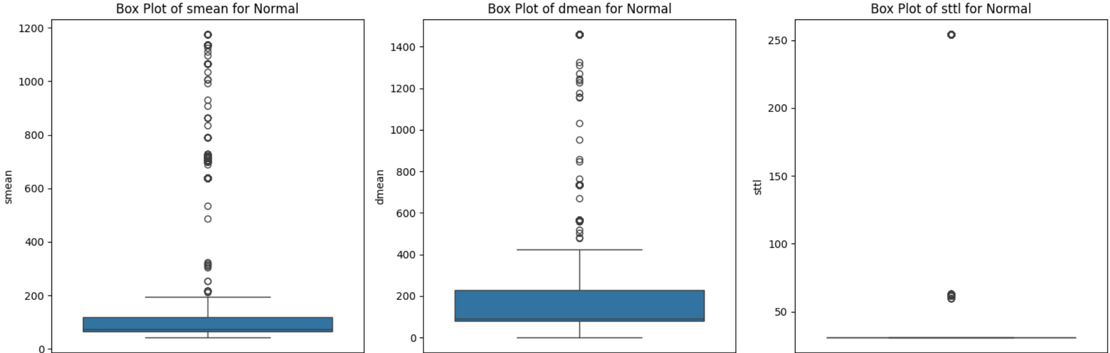


Figure 7.21: Boxplot Normal Traffic-Binary Classification

When the boxplot for the *smean* feature in normal traffic is examined, it can be seen that the median value is slightly below 100 and most of the values are concentrated between 0 and 200. Around 250 is where the upper whisker ends, suggesting that values above this cutoff may be regarded as anomalies in normal traffic. Even though there are a lot of outliers—values as high as 1200—these instances are obviously rarer, suggesting that the focus of anomaly detection should be on values greater than 200.

The boxplot shows a pattern similar to the one found in *smean* for the variable *dmean* (Destination Mean).In addition, most of the values fall between 0 and 200, with a median just above 100.The upper whisker reaches approximately 250, and many outliers up to 1400 are found. It means that high values in *dmean*, particularly those that surpass 200, should be viewed as possible indicators of anomaly traffic, similar to how *smean* is monitored.

In the end, there are two sets of outliers in the variable *sttl* (Source TTL): one group of values around 50, and the other group of values near 255.Most of the values are below 50, which means that most packets have a shorter TTL.There is a peak in the distribution at 50, then a steep decline and basically no values between 100 and 200.Since very high values are rare, it makes sense to look into these cases as potential anomalies.

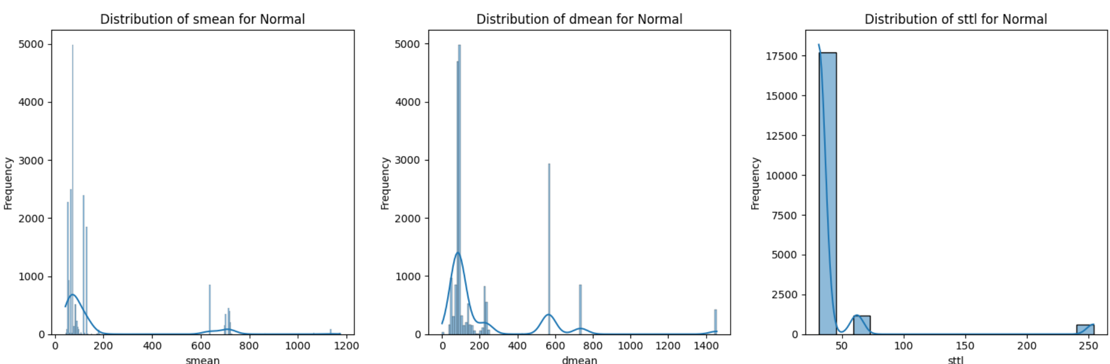


Figure 7.22: Distribution of features for Normal Traffic

The density distributions for the *smean* and *dmean* variables, which quickly decreases in frequency of data above 200, complement the analysis by displaying interesting peaks between 0 and 100.A threshold of 200 is needed to identify anomalies because most of the data is

concentrated in the lower range, with only a few small, sporadic peaks around 600 and 800 representing anomaly traffic values.

On the other hand, the *sttl* distribution shows a strong peak at 50 and a rapid fall shortly after.Values near 255 are far less common, pointing out how rare high TTL packets are in normal traffic. Based on this distribution, it is suggested to keeping a closer look for potentially anomaly traffic by monitoring packets with very low or high TTL.

Lastly, we studied each variable's normal ranges in order to spot anomalies.The range from 0 to 200 is predominant for *smean* and *dmean*, and values above 250 are more likely to show anomalous behavior.A normal range of 40 to 60 is recommended for *sttl*; values that are very high or low should be examined as they may indicate unusual traffic patterns.

DoS Attack:

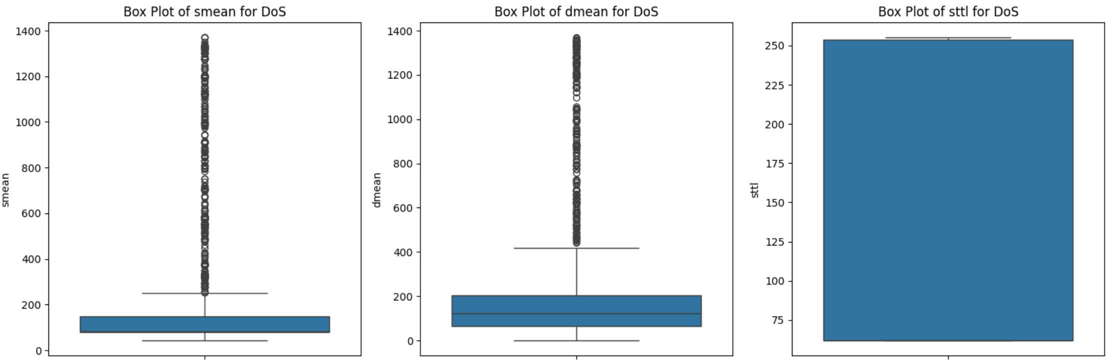


Figure 7.23: Boxplot DoS Attack

Boxplot analysis for the variables *smean*, *dmean*, and *sttl* delivers important traffic and DoS detection information.Most of the values for the *smean* variable are concentrated between 100 and 200, with the median being close to 150.But there are a lot of outliers—up to 1400—which implies that values higher than 250 might be signs of abnormal behavior.

The distribution of the *dmean* variable resembles to that of *smean*, with most of values falling between 100 and 200 again.There are many identified outliers, with a maximum of 1400, and the median is very near to 175.It means that high values in *dmean*, especially those above 250, may indicate anomalous traffic, similar to *smean*.

In the end, the distribution of the *sttl* variable is quite different, with no apparent outliers and most of values falling between 75 and 255.This shows that packets usually have a high TTL, suggesting they are not dropped off the network instantly.Low TTL packets are rare, as shown by the median's closeness to the upper limit.

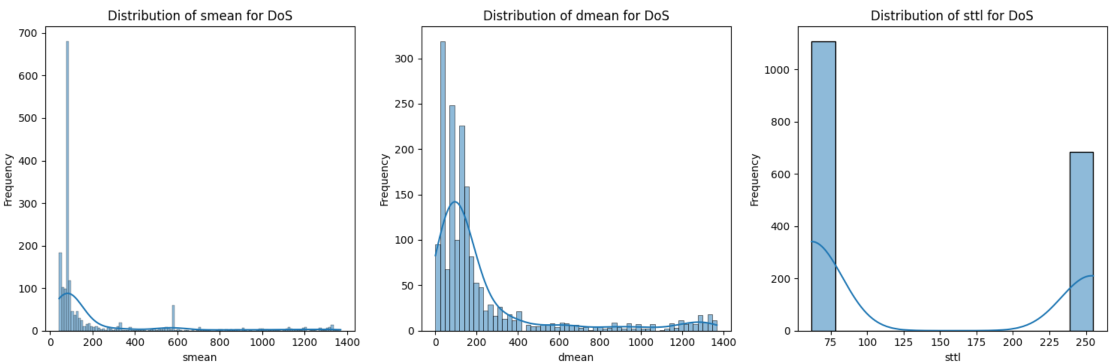


Figure 7.24: Distribution of features for DoS Attack

The density distributions for the variables offer useful data as well to this analysis.The distribution for *smean* shows a peak just below 50, with most of values falling between 0 and 200.This supports the idea that, because values above 200 are rare and may be signs of attacks, anomaly detection should focus on them.

A similar pattern can be seen in the distribution of *dmean*, with most of values falling between 0 and 200, and a peak near 100.This suggests that most of the traffic is light, and values above 200 should be observed as potential anomalies.

Lastly, a strong and symmetrical distribution between 200 and 255 is shown by the *sttl* variable, suggesting that packets usually have high TTLs.Since low values are rare, it is possible to keep an eye on such events to spot rare or potentially anomaly traffic.

Scanning:

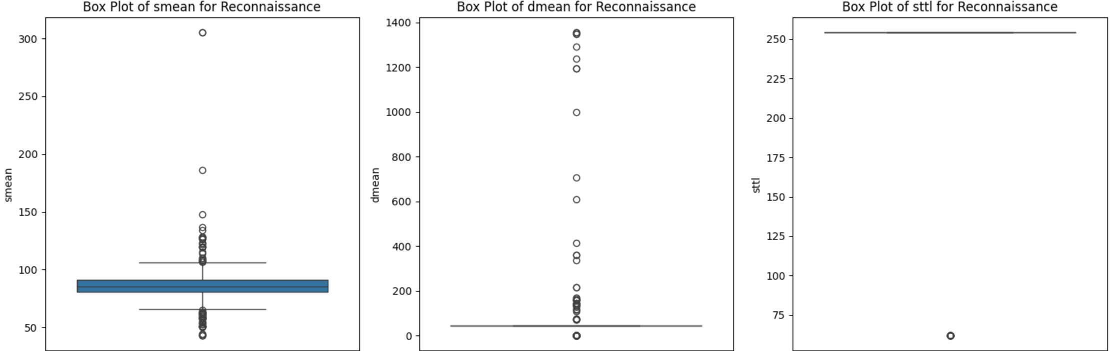


Figure 7.25: Boxplot Scanning Attack

The review of boxplots related to the variables *smean*, *dmean*, and *sttl* produces valuable information about the behavioral trends associated with scanning attacks.With a median of roughly 95–100 bytes, we found that the values for the *smean* variable tend to be concentrated between 75 and 100 bytes. Though some values exceed 300 bytes, there are notable outliers above 150 bytes, indicating that higher values might indicate unusual scans.

The box and whiskers in the *dmean* variable are quite flat, indicating that most of the data is at low values, even though the median in that variable is also between 50 and 100

bytes.Outliers up to 1400 bytes in size can indicate uncommon responses, suggesting that larger packets should be kept an eye out for feasible anomalies.

Lastly, the *sttl* variable's boxplot is especially curious; it shows an extreme concentration around 250 and minimal value deviation overall, with one instance of an obvious outlier below 100.This suggests that most of packets have a high TTL.

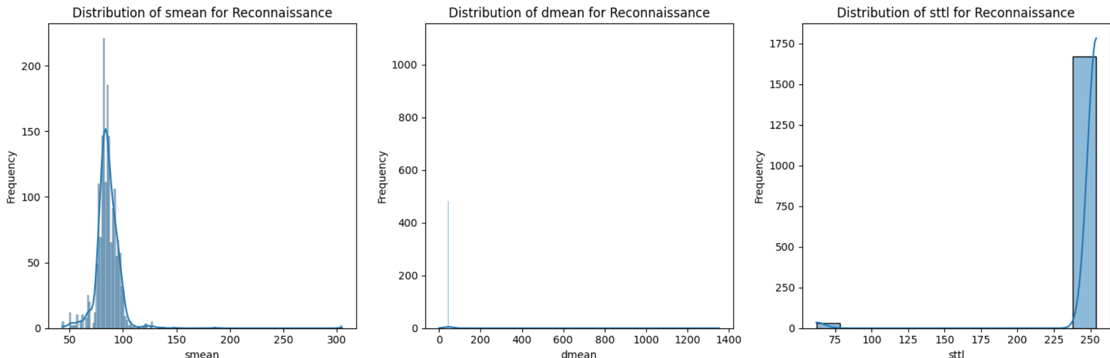


Figure 7.26: Distribution of features for Scanning Traffic

It is also helpful to analyze the variable density distributions.The distribution of *smean* is positively asymmetrical, with a tail to the right that extends to values above 200 bytes and the highest concentration between 75 and 100 bytes. According to this, most of normal activity is believed to happen below 125 bytes, while values above 150 bytes tend to be abnormal and should be investigated.

dmean has a very asymmetric distribution, with a long tail that reaches up to 1400 bytes and a peak that is quite close to zero.Most of responses are small sized—less than 100 bytes—and values greater than 200 bytes can indicate packets that diverge from the standard behaviour.

Finally, a strong concentration in the highest values, near 250, can be seen in the *sttl* distribution, suggesting that most of packets have a high TTL.Occasionally, values less than 100 could point to anomalies like packets with longer routes or network changes.

By the definition of proper intervals for each variable, these analyses help the identification of abnormal behavior in reconnaissance operations.The normal range for *smean* is 75–125 bytes, with outliers exceeding 150 bytes; the normal range for *dmean* is 0–100 bytes, with outliers over 200 bytes; and the normal range for *sttl* is 240–255 bytes, taking into account that values below 100 bytes may be suspicious.

The table below 7.2shows the ranges observed in the features for the 3 different scenarios for multi-class classification:

Table 7.2: Feature Value Ranges by Attack Type

Feature	Attack Type	Value Range
smean	Normal Traffic	[0, 200]
	DoS	[100, 200]
	Scanning	[75, 125]
dmean	Normal Traffic	[0, 200]
	DoS	[100, 200]
	Scanning	[0, 100]
sttl	Normal Traffic	[0, 50], [200, 255]
	DoS	[75, 255]
	Scanning	[240, 255]

MultiClass Prediction Tests

To evaluate the accuracy of the model in identifying all the different attack types, a variety of tests were executed. However, the results were unexpected, similar to the case with the binary detection tests, as every test generated the string "DoS Attack."The model may not be generalizing correctly or there may be inconsistencies in the input data that affect the classifications' accuracy, as suggested by this consistency in the predictions.

7.5 Limitations

7.5.1 Dataset Selection

The first stage of the work process, which included linking the feature values with the type of attack, required the use of files that already had defined labels, which conditioned the choice of dataset.Because of this limitation, the available datasets had to be limited, and datasets that would assist initial data categorization had to be chosen.

7.5.2 Attack Type Options

The list of datasets with important incidents decreased in scope depending on the attack types selected. The simplicity of some datasets limited the complexity of the scenarios that could be addressed, even though these datasets were appropriate for the chosen attacks.

7.5.3 Problems with Merging Datasets

There were many issues when merging datasets like CTU-13 and UNSW-NB15, such as the presence of several names for the same attributes, which required further work in standardizing and cleaning the data. Because of these variations, it was difficult to properly include the CTU-UNSW dataset into the anomaly detection API, which caused fusion process delays and errors.

7.5.4 Data Cleaning

It took a lot of time to clean up the data, which included fixing duplicate, invalid, and missing entries.For instance, it was required to remove or replace features with missing or irrelevant data, which would have resulted in the loss of essential data and possibly influenced results.

7.5.5 Data Visualization

The size of the.csv files prevented extensive examination of the dataset data as it wasn't possible to open them completely for detailed display.In order to bypass this limitation, an additional script had to be written, splitting the.csv files into smaller parts so that it could be exported to Excel and partially viewed.

7.5.6 No integration with the Cloud

One of the main requirements of the project failed because it wasn't possible to integrate the project into a cloud platform due to the costs associated with cloud services. Instead, the data was handled and modified locally, which might have limited the system's capacity to develop and become easier to access.

7.5.7 Live Detection Limitations

The system wasn't able to fully implement the live detection functionality, even though it was intended to detect attacks in real time. Pre-processing incoming network traffic data posed a number of challenges, especially in terms of organizing the data so that the machine learning models could evaluate it right away.

7.5.8 Overfitting and Incorrect Prediction in the API

The model's prediction and detection process in the API failed. This failure is probably due to model overfitting, which may have caused the model to be unable to correctly generalise the input data in real time. When a model is overfit to the training dataset, it loses performance on untrained data, which has a direct impact on the accuracy of the system.

7.5.9 Impossibility of Model Retraining on the Testbed

It was not possible to go through the same steps as the initial model (data pre-processing, normalization, training, and ensemble) because of the Testbed's poor implementation. As a result, the model's ability to adapt to new scenarios and system conditions was limited because it could not be retrained using data from the Testbed. This feature restricted the model's progress.

7.5.10 Time limit for in-depth Research

Due to the limited time available to develop the thesis, some areas of the work may not have been conducted in as much depth. Therefore, it wasn't possible to look at areas like a comprehensive review of different methods, changes of features, and validation in different data scenarios. A more comprehensive examination that would have increased the practical value of the results and improved the conclusions wasn't possible given the time limitation.

7.5.11 Using Python instead of C

Although C would be a more effective choice for embedded systems like routers, Python was the language of choice for the project's development. Performance and optimization issues might occur from the application of Python, particularly if the project is implemented on hardware with limited resources.

7.5.12 Functions of Parallelism and Complexity

When integrating with lower-performance hardware, like routers with only two processing cores, the implementation of parallelization functions and more complex algorithms is a limitation. The reliability and efficiency of the finished system could be compromised by these functions, which can result in serious performance issues or even crashes during execution.

Chapter 8

Discussion and Conclusion

8.1 Summary of Findings

8.1.1 Key Insights

The results of the research suggest that Z-Score normalization is better than Min-Max normalization, especially when used in the UNSW dataset. The accuracy of the test using the Z-Score model was 97.04%, which was higher than the Min-Max normalization model's 96.49%.

Z-Score demonstrated nearly perfect accuracy in identifying normal traffic with a precision of 0.99 for the normal class. The precision for the DoS class, on the other hand, was 0.81, indicating that while the model works well for recognizing regular traffic, its performance for detecting attack traffic is slightly lower, especially for identifying all cases of DoS attacks. These results show that Z-Score normalization keeps majority class performance without greatly affecting minority class performance, enabling better processing of unbalanced data.

Addressing the individual models, algorithms like Random Forest, Decision Tree, KNN, and MLP Classifier showed over 99% accuracy in binary classification.With accuracies of 84.38% and 93.82%, respectively, models like the SGD Classifier and Linear Regression were seen as worse and were eliminated from the analysis.

When it came to multiclass classification, the models performed similarly. KNN got a 97.10% accuracy rate, while Random Forest and Decision Tree got 97.19% and 96.88%. Somewhat lower, the MLP Classifier has an accuracy of 95.52%.

In binary classification, the ensemble model performed very well, with only 0.28% false negatives and 99.3% accuracy in labeling negative cases.The ensemble model demonstrated exceptional performance in multiclass classification, reaching an accuracy rate of 99.1% for the majority class and rates of 82.3% and 77.7% for minority classes. This indicates a significant increase in the identification of minority classes.The ensemble model increases accuracy in recognizing anomalies by reducing classification errors, as shown by the confusion matrices.

Meaningful Insights

These findings emphasize how important the choice of normalization is, since it can affect the performance of the model. In cases where there were unbalanced class distributions, Z-Score normalization in particular proved to be more reliable. In difficult scenarios, the accuracy of ensemble models may improve anomaly identification, particularly in multi-class classifications.

8.1.2 Contributions to the Field

The research presented here contributes greatly to the domains of anomaly detection and cyber security, especially when it comes to the application of machine learning models and normalization strategies. First, it highlights the need to select the right normalization, showing that in situations with unbalanced data, Z-Score normalization works better than Min-Max normalization.

In addition, the results show the high accuracy rates that machine learning models like KNN, Random Forest, and MLP Classifier are capable of in anomaly detection, indicating that traditional methods are still relevant in a cybersecurity environment that is constantly evolving.The application of ensemble techniques reveals how combining many models can greatly improve detection, especially in complex situations with a variety of attack types.

Finally, by providing a solid base for future investigation and real-world application, this research helps in the development of stronger and accurate anomaly detection systems.The suggested approach can be modified to be implemented into cyber security systems across a range of sectors, assisting in risk reduction while improving defense against cyberattacks.

8.2 Implications

8.2.1 Practical Implications

The practical implications of the research findings are important, especially in the context of smart home environments, where cyber security is an increasing relevant concern. Protecting Internet of Things devices in homes can directly benefit from the efficiency of machine learning models, particularly when ensemble approaches are used. By creating an anomaly detection system which is based on the trained models, it becomes possible to identify suspicious activity immediately and enhance system integrity and personal data protection.

An important step in this project involves creating a testbed for automating cyberattacks and capturing features for training. It makes it possible to retrain models using new data, guaranteeing that they can adjust to changing cyberattacks. In addition, the testbed offers a controlled environment for testing of the integrated API and the models, confirming their efficency before implementing in real-world scenarios. This iterative process ensures that the models maintain high performance and stay secure against attacks.

The use of APIs for the visualization and prediction of cyberattacks is another innovative element that makes it possible to offer the results of analysis in an engaging and simple way. This approach not only accelerates quick decisions but also decentralizes access to security data, allowing new as well as experienced users to gain a better understanding of the threats available in their environments.

8.2.2 Theoretical Implications

The results of this research have important theoretical implications for machine learning models used in cyber security. First of all, the findings highlight the importance of proper normalization for model performance, indicating that data pre-processing is an essential factor that can influence anomaly detection efficiency. This contributes to the body of research by showing the ongoing adoption of conventional normalization techniques in data modeling.

Also the research confirms the idea that combining multiple models might improve performance on difficult classification tasks like anomaly detection. This could lead to new recommendations and best practices for developing more reliable and efficient anomaly detection systems, as well as promote greater investigation into ensemble techniques and their applications in other cybersecurity fields.

8.3 Future Work

8.3.1 Short-Term Improvements

Retraining the model with data from the Testbed is one of the most important things that needs to be done. This allows the model to continually evolve to adapt to new threats and attack patterns. Regular retraining with new data ensures that the system maintains its ability to identify abnormalities, lowering the chance of false negatives and raising prediction precision. Additionally, it is possible to create an automated pipeline that will effectively incorporate fresh data, minimizing the need for human interaction and speeding the model update process.

8.3.2 Long-Term Research Directions

For an understanding over the stated limitations, a number of study routes could be investigated in the future. Firstly, it's essential to widen the dataset selection, including more complex and varied datasets that don't just concentrate on specific types of attacks can help us understand the variety of malicious activity that may take place in smart home environments.

Enhancing anomaly detection with advanced machine learning methods, including deep neural network-based methods or reinforcement learning, can help the system learn from its errors and become better at defending against new attacks.

In addition, the use of more solid and user-friendly data visualisation technologies that can manage large quantities of data without losing efficiency will help in the processing and comprehension of data. Reviewing the use of cloud services is also essential because of the ability to grow and offer remote access to data, which can greatly increase system efficiency and flexibility.

Finally, research on the portability of code, considering the use of languages like C to optimize performance in embedded systems, needs to be done to guarantee that the application works effectively on hardware with limited resources, such as home routers.

Bibliography

[1] Ryan Daws. *Kaspersky: Attacks on IoT devices double in a year*. 7th September 2021. 2021. url: https://iottechnews.com/news/kaspersky-attacks-on-iot-devices-double-in-a-year/.

[2] Fortune Business Insights. *Smart Home Market Size, Share & Industry Analysis, By Device Type (Safety & Security Access Control, Home Appliances, HVAC, Lighting Control, Smart Entertainment Devices, Smart Kitchen Appliances, and Others), By Application (Retrofit and New Construction), By Protocol (Wired and Wireless), and Regional Forecast, 2024-2032*. Last Updated: September 16, 2024. 2024. url: https://www.fortunebusinessinsights.com/industry-reports/smart-home-market-101900.

[3] Chiran Chhetri and Vijay G. Motti. "Identifying Vulnerabilities in Security and Privacy of Smart Home Devices". In: *Sep. 2020* (). doi: 10.1007/978-3-030-58703-1-13.

[4] Precedence Research. *Home Automation Market Size, Share, Growth Analysis Report 2022-2032*. 2023. url: https://www.precedenceresearch.com/home-automation-market.

[5] Joseph Migga Kizza. *Guide to Computer Network Security*. 6th. Texts in Computer Science. Springer Cham, 2024, XX–YY. isbn: 978-3-031-47548-1. doi: 10.1007/978-3-031-47549-8.

[6] J. Bugeja, D. Jonsson, and A. Jacobsson. "An Investigation of Vulnerabilities in Smart Connected Cameras". In: (2018). doi: 10.1109/percomw.2018.8480184.

[7] L. Farhan et al. "A Concise Review on Internet of Things (IoT) - Problems, Challenges and Opportunities". In: (2018). doi: 10.1109/csndsp.2018.8471762.

[8] Spyridon Samonas and David Coss. "The CIA strikes back: Redefining confidentiality, integrity and availability in security." In: *Journal of Information System Security* 10.3 (2014).

[9] Fortinet. *CIA Triad*. unknown. url: https://www.fortinet.com/resources/cyberglossary/cia-triad.

[10] S. K. Mousavi et al. "Security of Internet of Things Based on Cryptographic Algorithms: A Survey". In: (2021). doi: 10.1007/s11276-020-02535-5.

[11] M. S. Mehmood et al. "A Comprehensive Literature Review of Data Encryption Techniques in Cloud Computing and IoT Environment". In: (2019). doi: 10.1109/icict47744.2019.9001945.

[12] P. P. Mukkamala and S. Rajendran. "A Survey On The Different Firewall Technologies". In: 5.1 (2020), pp. 363–365.

[13] Somayye Hajiheidari et al. "Intrusion detection systems in the Internet of things: A comprehensive investigation". In: *Computer Networks* 165 (2019). Received: 26 December 2017; Revised: 16 May 2019; Accepted: 16 May 2019, p. 106979. doi: 10.1016/j.comnet.2019.05.014. url: https://doi.org/10.1016/j.comnet.2019.05.014.

[14] Alka Mishra and Pradeep Yadav. "Anomaly-based IDS to Detect Attack Using Various Artificial Intelligence & Machine Learning Algorithms: A Review". In: *Proceedings*

of the 2nd International Conference on Data, Engineering and Applications (IDEA). Bhopal, India, Feb. 2020, pp. 1–7. doi: 10.1109/IDEA49133.2020.917.

[15] M. F. Elrawy, A. I. Awad, and H. F. A. Hamed. “Intrusion Detection Systems for IoT-based Smart Environments: A Survey”. In: (2018). doi: 10.1186/s13677-018-0123-6.

[16] Neerja Mhaskar, Mohammed Alabbad, and Ridha Khedri. “A Formal Approach to Network Segmentation”. In: *Computers & Security* (2021). doi: 10.1016/j.cose.2020.102162.

[17] N. Mhaskar, M. Alabbad, and R. Khédri. “A Formal Approach to Network Segmentation”. In: (2021). doi: 10.1016/j.cose.2020.102162.

[18] Shailendra and K. Joseph. “Analysis on IoT Networks Security: Threats, Risks, ESP8266 based Penetration Testing Device and Defense Framework for IoT Infrastructure”. In: (2023). doi: 10.1109/conit59222.2023.10205679.

[19] M. H. U. Sharif and M. A. Mohammed. “A literature review of financial losses statistics for cyber security and future trend”. In: (2022). doi: 10.30574/wjarr.2022.15.1.0573.

[20] H. A. Abdul-Ghani, D. Konstantas, and M. Mahyoub. “A Comprehensive IoT Attacks Survey based on a Building-blocked Reference Model”. In: ().

[21] E. L. Rissland. “Artificial Intelligence and Law: Stepping Stones to a Model of Legal Reasoning”. In: *The Yale Law Journal* 99.8 (June 1990), pp. 1957–1981. doi: http://www.jstor.org/stable/796679.

[22] L. Samuel. “Some Studies in Machine Learning Using the Game of Checkers”. In: (July 1959). doi: 10.1147/rd.33.0210.

[23] K. M. Jablonka et al. “Big-Data Science in Porous Materials: Materials Genomics and Machine Learning”. In: (June 2020). doi: 10.1021/acs.chemrev.0c00004.

[24] Z. K. Maseer et al. “Benchmarking of Machine Learning for Anomaly Based Intrusion Detection Systems in the CICIDS2017 Dataset”. In: *IEEE Access* (Jan. 2021). doi: 10.1109/access.2021.3056614.

[25] S. S. Gill et al. “AI for next generation computing: Emerging trends and future directions”. In: *Internet of Things* (Aug. 2022). doi: 10.1016/j.iot.2022.100514.

[26] F. Jáñez-Martino et al. “Classification of Spam Emails through Hierarchical Clustering and Supervised Learning”. In: *arXiv preprint arXiv:2005.08773* (May 2020). doi: 10.48550/arxiv.2005.08773.

[27] Ajay Kaushik, Shubham Gupta, and Manan Bhatia. “A Movie Recommendation System using Neural Networks”. In: *International Journal of Advance Research, Ideas and Innovations in Technology* 4.2 (2018). Impact factor: 4.295, p. 425. issn: 2454-132X. url: http://www.ijariit.com.

[28] I. H. Sarker. “Machine Learning: Algorithms, Real-World Applications and Research Directions”. In: *Journal of Ambient Intelligence and Humanized Computing* (Mar. 2021). doi: 10.1007/s42979-021-00592-x.

[29] M. Zhong, Y. Zhou, and G. Chen. “Sequential Model Based Intrusion Detection System for IoT Servers Using Deep Learning Methods”. In: *Sensors* 21.4 (Feb. 2021), p. 1113. doi: 10.3390/s21041113.

[30] M. Al-Ambusaidi et al. “ML-IDS: An Efficient ML-Enabled Intrusion Detection System for Securing IoT Networks and Applications”. In: *Soft Computing* (Dec. 2023). doi: 10.1007/s00500-023-09452-7.

[31] F. Hussain et al. “Machine Learning in IoT Security: Current Solutions and Future Challenges”. In: *IEEE Communications Surveys & Tutorials* (Jan. 2020). doi: 10.1109/comst.2020.2986444.

[32] S. Branco, A. G. Ferreira, and J. Cabral. "Machine Learning in Resource-Scarce Embedded Systems, FPGAs, and End-Devices: A Survey". In: *Electronics* (Nov. 2019). doi: 10.3390/electronics8111289.

[33] E. Batzolis, . Vrochidou, and G. A. Papakostas. "Machine Learning in Embedded Systems: Limitations, Solutions and Future Challenges". In: *2023 13th International Conference on Computing, Communication and Networking Technologies (ICCCNT)* (Mar. 2023). doi: 10.1109/ccwc57344.2023.10099348.

[34] M. N. Gevorkyan et al. "Review and Comparative Analysis of Machine Learning Libraries for Machine Learning". In: *Journal Name* 27.4 (Dec. 2019), pp. 305–315. doi: 10.22363/2658-4670-2019-27-4-305-315.

[35] Kiran Maharana, Surajit Mondal, and Bhushankumar Nemade. "A review: Data preprocessing and data augmentation techniques". In: *Global Transitions Proceedings* 3.1 (2022), pp. 91–99. issn: 2666-285X. doi: 10.1016/j.gltp.2022.04.020. url: https://www.sciencedirect.com/science/article/pii/S2666285X22000565.

[36] Gil Press. "Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says". In: *Forbes* (2016). Accessed: 2024. url: https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task/.

[37] Ga Young Lee et al. "A Survey on Data Cleaning Methods for Improved Machine Learning Model Performance". In: *arXiv* (2021). Submitted on 15 Sep 2021. url: https://doi.org/10.48550/arXiv.2109.07127.

[38] Formplus. *Data Cleaning: 7 Techniques + Steps to Cleanse Data*. Accessed: 2024. n.d. url: https://www.formpl.us/blog/data-cleaning.

[39] Susan Seba and Amitesh Kumar. "The balancing trick: Optimized sampling of imbalanced datasets—A brief survey of the recent State of the Art". In: *Engineering Reports* 2.10 (Oct. 2020), e12298. doi: 10.1002/eng2.12298.

[40] Dalwinder Singh and Birmohan Singh. "Investigating the impact of data normalization on classification performance". In: *Applied Soft Computing* (2019). doi: 10.1016/j.asoc.2019.105524.

[41] Chris Seger. "An investigation of categorical variable encoding techniques in machine learning: binary versus one-hot and feature hashing". PhD thesis. Dissertation, 2018.

[42] Harith Al-Sahaf et al. "A survey on evolutionary machine learning". In: *Journal of the Royal Society of New Zealand* (2019). doi: 10.1080/03036758.2019.1609052.

[43] Pritam Dhal and Chandan Azad. "A comprehensive survey on feature selection in the various fields of machine learning". In: *Applied Intelligence* (2021). doi: 10.1007/s10489-021-02550-9.

[44] N. Gopika and A. Meena Kowshalaya M.E. "Correlation Based Feature Selection Algorithm for Machine Learning". In: *2018 3rd International Conference on Communication and Electronics Systems (ICCES)*. 2018. doi: 10.1109/CESYS.2018.8723980.

[45] Wenwen Jia, Minglei Sun, Jiancheng Lian, et al. "Feature dimensionality reduction: a review". In: *Complex & Intelligent Systems* 8.4 (2022), pp. 2663–2693. doi: 10.1007/s40747-021-00637-x.

[46] Lubna Ali Hassan Ahmed and Yahia Abdalla Mohamed Hamad. "Machine Learning Techniques for Network-based Intrusion Detection System: A Survey Paper". In: *National Computing Colleges Conference (NCCC)* (2021), pp. 1–7. doi: 10.1109/NCCC49330.2021.9428827.

[47] Faisal Shahzad, Abdul Mannan, Ali Raza Javed, et al. "Cloud-based multiclass anomaly detection and categorization using ensemble learning". In: *Journal of Cloud Computing* 11.1 (2022), p. 74. doi: 10.1186/s13677-022-00329-y.

[48] Meysam Vakili, Mohammad Ghamsari, and Masoumeh Rezaei. "Performance analysis and comparison of machine and deep learning algorithms for IoT data classification". In: *arXiv preprint arXiv:2001.09636* (2020).

[49] Ibomoiye Domor Mienye and Yanxia Sun. "A Survey of Ensemble Learning: Concepts, Algorithms, Applications, and Prospects". In: *IEEE Access* 10 (2022), pp. 99129–99149. doi: 10.1109/ACCESS.2022.3207287.

[50] Gen. David L. *Comparison of Flask, Django, and FastAPI: Advantages, Disadvantages, and Use Cases*. 2020. url: https://medium.com/@tubelwj/comparison-of-flask-django-and-fastapi-advantages-disadvantages-and-use-cases-63e7c692382a.

[51] J. P. Hwang. *Plotly Dash vs Streamlit — Which is the best library for building data dashboard web apps?* 2020. url: https://towardsdatascience.com/plotly-dash-vs-streamlit-which-is-the-best-library-for-building-data-dashboard-web-apps-97d7c98b938c.

[52] Anwar Mahmood et al. *UNSW-NB15 Dataset*. Accessed: 2024. 2017. url: https://research.unsw.edu.au/projects/unsw-nb15-dataset.

[53] Muhammad Aidiel Rachman Putra, Tohari Ahmad, and Dandy Pramana Hostiadi. "Analysis of Botnet Attack Communication Pattern Behavior on Computer Networks". In: *International Journal of Intelligent Engineering and Systems* 15.4 (2022). doi: 10.22266/ijies2022.0831.48.

[54] Miel Verkerken et al. "A Novel Multi-Stage Approach for Hierarchical Intrusion Detection". In: *IEEE Transactions on Network and Service Management* 20.3 (2023), pp. 3915–3929. doi: 10.1109/TNSM.2023.3259474.

[55] N.A. Stoian. *Machine Learning for anomaly detection in IoT networks : Malware analysis on the IoT-23 data set*. July 2020. url: http://essay.utwente.nl/81979/.

[56] Q. Wen et al. "Time series data augmentation for deep learning: a survey". In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence* (2021), pp. 4653–4660. doi: 10.24963/ijcai.2021/631.

[57] *scikitlearn-User Guide*. Accessed: 2024. SciKit-Learn. url: https://scikit-learn.org/stable/user_guide.html.

[58] Craig Leres Van Jacobson and Steven McCanne. *tcpdump(1) man page*. Accessed: 2024. url: https://www.tcpdump.org/manpages/tcpdump.1.html.

[59] Gordon Fyodor Lyon. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Sunnyvale, CA, USA: Insecure.Com, LLC, 2009. isbn: 978-0-9799587-1-7. url: https://nmap.org/book/.

[60] Salvatore Sanfilippo. *hping3(8) - Linux man page*. Accessed: 2024. url: https://linux.die.net/man/8/hping3.

[61] *ping(8) - Linux man page*. Accessed: 2024. Linux. url: https://linux.die.net/man/8/ping.

Appendix A

Project Structure and Code Overview

In this appendix, only the most relevant scripts will be included to avoid making the content redundant and overly detailed.

A.1 API

This folder contains scripts for the FastAPI application and the Streamlit panel:

- **fastapi.py**: Implements a FastAPI application for anomaly detection using two models (binary and multi-class). It includes endpoints for predicting anomalies and sending notifications when an anomaly is detected.
- **streamlit_api.py**: Streamlit panel that provides an interface for the real-time detection of anomalies and allows users to submit data for both binary and multi-class forecasts.

A.2 Helpers Directory

A.2.1 Aux_Testbed_Pipeline

This subfolder contains scripts that facilitate various phases of the anomaly detection pipeline:

- **01_Scenario_Complete.py**: ERuns the complete testbed scenario.
- **02_label2.py**: Manage the labelling process for datasets.
- **03_Normalization.py**: Normalises dataset features.
- **04_data_splitter.py**: Divides datasets into training and test subsets.
- **05_train_anomaly_detection_models.py**: Train the models to detect anomalies.
- **05_train_parallel.py**: It trains models in parallel to increase efficiency.
- **06_ensemble_predictions.py**: Combines predictions from multiple models.
- **pipeline.py**: Main script that runs the pipeline.
- **semi_pipe.py**: Semi-automated version of the pipeline for quick tests.

A.2.2 Aux_Scripts

Folder with support scripts for data processing and analysis tasks:

- **00_Common_Dataset_Features.py**: Extracts common features from datasets.
- **00_Dataset_Division.py**: Divide datasets into specific categories.
- **00_Excel_combination.py**: Combines multiple Excel files into one.
- **CTU_13_merge.py**: Combine CTU-13 datasets for unified analysis.
- **unified_datasets.py**:Unify different datasets into a single comprehensive dataset.

A.3 Pipeline Directory

The `Pipeline` directory is structured to facilitate the complete workflow of data processing, normalisation, training and evaluation of machine learning models.

A.3.1 0.Cleaning

Scripts for cleaning datasets, preparing them for analysis and modelling.

A.3.2 1.Training

Subdirectories for different normalization techniques, such as `min_max_score` and `z_score`, each containing dedicated scripts for cleaning, coding, feature selection, and training for specific datasets such as UNSW, CTU, and CTU-UNSW.

A.4 Scripts Directory

This folder contains scripts that execute multiple stages of the pipeline:

- **Running_API.py**: Starts the FastAPI server and the Streamlit frontend.
- **Min-Max Normalization Pipeline**: Scripts for Min-Max normalisation, feature selection and model training.
- **Z-Score Normalization Pipeline**: Scripts for Z-score normalisation, coding, feature selection and training.

Appendix B

API Implementation

B.1 FastAPI Application

```
# fastapi_app.py

# uvicorn fastapi_app.py:app --reload

import joblib
import logging
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
import numpy as np
import requests
import os
from typing import Union

logging.basicConfig(level=logging.INFO)

app = FastAPI()

# Get the current directory of this script
script_dir = os.path.dirname(__file__)

# Model paths for binary and multi-class prediction (relative paths)
binary_model_path = os.path.join(script_dir, "..", "Pipeline", "Models",
"UNSW_zscore_voting_combined_binary_v1.pkl")
multiclass_model_path = os.path.join(script_dir, "..", "Pipeline", "Models",
"UNSW_zscore_voting_combined_multi_v1.pkl")

# Variables to store the loaded models
binary_model = None
multiclass_model = None

# Function to load the models
async def load_models():
    global binary_model, multiclass_model
    binary_model = joblib.load(binary_model_path)
    multiclass_model = joblib.load(multiclass_model_path)
```

```
    logging.info("Binary and multi-class models loaded successfully.")

@app.on_event("startup")
async def startup_event():
    await load_models()

@app.get("/")
def read_root():
    return {"message": "Welcome to the API"}

def send_pushover_notification(message):
    try:
        logging.info("Sending Pushover notification...")
        response = requests.post("https://api.pushover.net/1/messages.json", data={
            "token": pushover_api_token, #API Keys Removed for Safety Purposes
            "user": pushover_user_key, #API Keys Removed for Safety Purposes
            "message": message
        })
        response.raise_for_status()
        logging.info("Notification sent successfully.")
        return response.json()
    except Exception as e:
        logging.error(f"Error sending Pushover notification: {e}")

# Model for binary data input
class InputDataBinary(BaseModel):
    ct_srv_dst: Union[int, float]
    trans_depth: Union[int, float]
    synack: float
    tcprtt: float
    ackdat: float
    sttl: Union[int, float]
    dttl: Union[int, float]
    ct_state_ttl: Union[int, float]

# Model for multi-class data input
class InputDataMultiClass(BaseModel):
    smean: Union[int, float]
    dmean: Union[int, float]
    sttl: Union[int, float]


# Endpoint for anomaly detection (binary: attack or not)
@app.post("/detect_binary/")
def api_detect_anomaly(input_data: InputDataBinary):
    try:
        # Accessing attributes directly
        data = np.array([
            input_data.ct_srv_dst,
```

```
            input_data.trans_depth,
            input_data.synack,
            input_data.tcprtt,
            input_data.ackdat,
            input_data.sttl,
            input_data.dttl,
            input_data.ct_state_ttl
        ]).reshape(1, -1)

        prediction = binary_model.predict(data)[0]
        print(f"Model prediction: {prediction}")

        if prediction == 1:
            send_pushover_notification("Anomaly detected!")
            return {"anomaly": True, "message": "Anomaly detected!"}
        else:
            return {"anomaly": False, "message": "Normal Network Traffic."}
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

# Endpoint for multi-class prediction (detect type of attack or benign)
@app.post("/predict_multiclass/")
def api_predict_multiclass(input_data: InputDataMultiClass):
    try:
        # Accessing attributes directly
        data = np.array([
            input_data.smean,
            input_data.dmean,
            input_data.sttl
        ]).reshape(1, -1)

        prediction = multiclass_model.predict(data)[0]

        # Mapping output classes (adjust as necessary)
        label_map = {
            1: "Benign traffic",
            0: "DoS Attack",
            2: "Scanning"
        }

        result = label_map.get(prediction, "Unknown")

        return {"prediction": result}
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

B.2 Streamlit Dashboard

```
import altair as alt
import streamlit as st
import numpy as np
import pandas as pd
import requests  # Imports requests to call the FastAPI

# URL of the FastAPI
API_URL_BINARY = "http://localhost:8000/detect_binary/"
API_URL_MULTICLASS = "http://localhost:8000/predict_multiclass/"

def get_prediction(api_url, data):
    try:
        response = requests.post(api_url, json=data)
        response.raise_for_status()
        return response.json()
    except requests.exceptions.RequestException as e:
        st.error(f"Error calling the API: {e}")
        return None

def run_streamlit_dashboard():
    st.markdown(
        """
        <style>
        .stApp {
            background-color: #1A1A1A;
        }
        .sidebar .sidebar-content {
            background-color: #FFA500;
            color: #424242;
        }
        .stButton > button {
            background-color: #FF6F00;
            color: white;
        }
        .stAlert {
            background-color: #E3F2FD;
            color: #424242;
        }
        .footer {
            position: fixed;
            bottom: 0;
            right: 0;
            padding: 10px;
            z-index: 1000;
        }
        .footer img {
            width: 100px;
```

```
            height: auto;
        }
        .live-detection {
            position: absolute;
            top: 20px;
            left: 20px;
            background-color: #FFFFFF;
            border: 2px solid #FF6F00;
            padding: 10px;
            border-radius: 5px;
            box-shadow: 0px 4px 6px rgba(0, 0, 0, 0.1);
        }
        .title-container {
            width: 100%;
            max-width: 800px;
            margin: 0 auto;
            padding-right: 50px;
        }
        h1 {
            font-size: 60px;
            white-space: nowrap;
        }
        </style>
        """,
        unsafe_allow_html=True
    )

    st.markdown('<div class="title-container"><h1>Anomaly Detection Dashboard</h1></div

    tab1, tab2, tab3 = st.tabs(["Live Detection", "Detection History", "Settings"])

    with tab1:
        st.markdown('<div class="live-detection">', unsafe_allow_html=True)
        st.header("Live Detection")

        st.markdown(
            """
            <img src="https://upload.wikimedia.org/wikipedia/commons/0/05/Red_Pulsing_D
            """,
            unsafe_allow_html=True
        )

        times = pd.date_range(end=pd.Timestamp.now(), periods=24, freq='H').to_pydateti
        severity = np.random.choice([0, 1, 2], size=24)

        df = pd.DataFrame({"Time": times, "Severity": severity})

        df['Severity'] = df['Severity'].map({
            0: 'Benign',
```

```
        1: 'Scan',
        2: 'DoS'
    })

    chart = alt.Chart(df).mark_line().encode(
        x='Time:T',
        y=alt.Y('Severity:N', scale=alt.Scale(domain=['Benign', 'Scan', 'DoS']), title=
    ).properties(
        width=700,
        height=400
    )

    st.altair_chart(chart)

    st.header("Enter data for analysis:")

    # Selector to choose the type of detection
    detection_type = st.selectbox("Choose the type of detection:", ["Binary Detection",

    if detection_type == "Binary Detection":
        # Inputs for binary detection
        col1, col2 = st.columns(2)

        with col1:
            data = {
                "ct_srv_dst": st.number_input("ct_srv_dst", value=0.5),
                "trans_depth": st.number_input("trans_depth", value=2.0),
                "synack": st.number_input("synack", value=1.5),
                "tcprtt": st.number_input("tcprtt", value=0.6),
                "ackdat": st.number_input("ackdat", value=1.2)
            }

        with col2:
            data.update({
                "sttl": st.number_input("sttl", value=1.2),
                "dttl": st.number_input("dttl", value=1.2),
                "ct_state_ttl": st.number_input("ct_state_ttl", value=1),
            })

        if st.button("Send Data for Binary Detection"):
            with st.spinner("Analyzing data..."):
                prediction = get_prediction(API_URL_BINARY, data)

            if prediction:
                st.write(f"Result of Binary Detection: {prediction['message']}")
                if 'anomaly_history' not in st.session_state:
                    st.session_state.anomaly_history = []
                st.session_state.anomaly_history.append((prediction['anomaly'], predict
```

```
        else:
            # Inputs for multiclass prediction
            data_multiclass = {
                "smean": st.number_input("smean", value=5.6),
                "dmean": st.number_input("dmean", value=3.7),
                "sttl": st.number_input("sttl", value=1.2)
            }

            if st.button("Send Data for Multiclass Prediction"):
                with st.spinner("Analyzing data..."):
                    prediction = get_prediction(API_URL_MULTICLASS, data_multiclass)

                if prediction:
                    st.write(f"Result of Multiclass Prediction: {prediction['predictio
                    if 'anomaly_history' not in st.session_state:
                        st.session_state.anomaly_history = []
                    st.session_state.anomaly_history.append((None, prediction['predicti

    with tab2:
        st.header("Detection History")
        if 'anomaly_history' in st.session_state:
            for p1, p2 in st.session_state.anomaly_history:
                st.write(f"Result of Binary Detection: {p1}, Result of Multiclass Predi
        else:
            st.write("No detection recorded.")

    with tab3:
        st.header("Settings")
        st.write("Set up configs later!")

if __name__ == "__main__":
    run_streamlit_dashboard()
```

Appendix C

Pipeline Implementation

C.1 Overview

The Pipeline directory is structured to help the entire workflow of data processing, normalization, training, and evaluation of machine learning models. It is divided into two main subdirectories:

- **0.Cleaning**: Contains scripts for cleaning each dataset used in the project, ensuring that the datasets are preprocessed and ready for further analysis and modeling.
- **1.Training**: Contains scripts for training machine learning models with different normalization techniques, including Min-Max normalization and Z-score normalization.

C.2 0.Cleaning

The `0.Cleaning` directory includes the following script:

```
    import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

"""
This script processes the UNSW-NB15 dataset by performing the following tasks:

1. **Data Loading**: The ‘load_data‘ function loads the dataset from a CSV file and pro

2. **Data Cleaning and Conversion**:
    - The ‘clean_data‘ function performs multiple data preparation tasks:
        - Replaces invalid values (e.g., replacing ’-’ with NaN in the ’service’ column
        - Removes rows with missing values (NaN) to ensure the dataset is clean.
        - Filters the dataset to include only relevant attack categories (’DoS’, ’Recon
        - Converts column types based on predefined feature types (nominal, integer, bi
        - Displays the distribution of ’attack_cat’ and ’state’ categories to provide i

3. **Data Visualization**:
    - The ‘plot_data‘ function generates and saves pie charts to visualize the distribu
        - Binary classification labels (normal vs abnormal).
        - Multi-class labels (various attack types).
```

```
4. **Main Execution Flow**:
    - The `main` function orchestrates the data processing pipeline:
        - Loads the dataset and feature description file.
        - Cleans and preprocesses the data.
        - Saves the cleaned dataset to a specified location.
        - Generates and saves visual plots for further analysis.

The script is designed to work specifically with the UNSW-NB15 dataset but can be adapted f
"""

# Function for loading data
def load_data(file_path):
    df = pd.read_csv(file_path)

    # Initial data analysis
    print("Initial dataset information:")
    df.info()

    # Display the first lines of the dataset
    print("\nFirst lines of the dataset:")
    print(df.head())

    print("Available columns in the DataFrame:")
    print(df.columns.tolist())

    return df

# Function to clean and convert data
def clean_data(df, features_file):
    # Replace `-' values with NaN in the `service' column if it exists
    if 'service' in df.columns:
        df['service'] = df['service'].replace('-', np.nan)
        print(df.isnull().sum())

    # Display the original shape of the data
    print(f"\nOriginal shape of the data: {df.shape}")

    # Remove rows with missing values
    df.dropna(inplace=True)

    # Display the shape of the data after removing NaNs
    print(f"Shape of the data after removing NaNs: {df.shape}")

    # Remove rows where `attack_cat' is not `DoS', `Scanning', or `Normal'
    if 'attack_cat' in df.columns:
        allowed_categories = ['DoS', 'Reconnaissance', 'Normal']
        initial_shape = df.shape
        df = df[df['attack_cat'].isin(allowed_categories)]
        print(f"\nRows removed with 'attack_cat' outside of {allowed_categories}: {initial_
```

```
    print(f"Shape of the data after filtering by 'attack_cat': {df.shape}")

# Display the count of unique values in the 'attack_cat' column
if 'attack_cat' in df.columns:
    print("\nDistribution of categories in 'attack_cat':")
    print(df['attack_cat'].value_counts())

# Display the count of unique values in the 'state' column
if 'state' in df.columns:
    print("\nDistribution of states in 'state':")
    print(df['state'].value_counts())
    print(df.shape)

# Load information about features
features = pd.read_csv(features_file)
features.head()

features['Type '] = features['Type '].str.lower()

# Selecting column names of all data types
nominal_names = features['Name'][features['Type '] == 'nominal']
integer_names = features['Name'][features['Type '] == 'integer']
binary_names = features['Name'][features['Type '] == 'binary']
float_names = features['Name'][features['Type '] == 'float']

# Selecting common column names from dataset and feature dataset
cols = df.columns
nominal_names = cols.intersection(nominal_names)
integer_names = cols.intersection(integer_names)
binary_names = cols.intersection(binary_names)
float_names = cols.intersection(float_names)

# Convert columns to numeric types
for c in integer_names:
    if c in df.columns:
        df[c] = pd.to_numeric(df[c], errors='coerce')
    else:
        print(f"Column {c} not found in the DataFrame.")

for c in binary_names:
    if c in df.columns:
        df[c] = pd.to_numeric(df[c], errors='coerce')
    else:
        print(f"Column {c} not found in the DataFrame.")

for c in float_names:
    if c in df.columns:
        df[c] = pd.to_numeric(df[c], errors='coerce')
    else:
```

```
            print(f"Column {c} not found in the DataFrame.")

    print(df.info())

    # Display the shape of the data after removing NaNs
    print(f"Shape of the data after removing NaNs: {df.shape}")

    # Display the count of unique values in the 'attack_cat' column
    if 'attack_cat' in df.columns:
        print("\nDistribution of categories in 'attack_cat':")
        print(df['attack_cat'].value_counts())

    # Display the count of unique values in the 'state' column
    if 'state' in df.columns:
        print("\nDistribution of states in 'state':")
        print(df['state'].value_counts())

    return df

def plot_data(df):
    # Plot binary label distribution
    plt.figure(figsize=(8, 8))
    plt.pie(df.label.value_counts(), labels=['normal', 'abnormal'], autopct='%0.2f%%')
    plt.title("Pie chart distribution of normal and abnormal labels", fontsize=16)
    plt.legend()
    plt.savefig('/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Plots/UNSW_Pie_chart_
    plt.show()

    # Plot multiclass label distribution
    plt.figure(figsize=(8, 8))
    plt.pie(df.attack_cat.value_counts(), labels=df.attack_cat.unique(), autopct='%0.2f%%')
    plt.title('Pie chart distribution of multi-class labels')
    plt.legend(loc='best')
    plt.savefig('/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Plots/UNSW_Pie_chart_
    plt.show()

def main():
    # Load the data
    file_path = '/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Datasets/00_Original_
    features_file = '/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Datasets/00_Origi

    data = load_data(file_path)

    # Clean and convert the data
    cleaned_data = clean_data(data, features_file)

    # Save processed data
    cleaned_data.to_csv('/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Datasets/01_C
```

```
    # Plot and save graphs
    plot_data(cleaned_data)

    print("Data prepared, saved, and graphs generated successfully.")

if __name__ == '__main__':
    main()
```

C.3 1.Training

This subfolder has multiple scripts:

One Hot Encoding Script

```
    import pandas as pd

"""
This script applies One-Hot Encoding to categorical features in the UNSW-NB15 dataset.

1. **One-Hot Encoding Function** (`one_hot_encoding`):
    - **Column Selection**:
        - The function first separates numerical columns from categorical ones, ignorin
        - The categorical columns are identified by selecting all non-numeric columns f
    - **Encoding Process**:
        - A new DataFrame is created, containing only the categorical columns.
        - One-Hot Encoding is applied to these categorical columns, converting each cat
    - **Concatenation and Cleanup**:
        - The original DataFrame and the newly encoded DataFrame are concatenated.
        - The original categorical columns are then dropped, leaving only the numerical

2. **Main Execution Flow**:
    - The `main` function handles the following:
        - Loads the pre-processed dataset from a CSV file.
        - Applies One-Hot Encoding using the `one_hot_encoding` function.
        - Saves the encoded dataset into a new CSV file for further processing or analy

This script is intended to prepare the UNSW-NB15 dataset for machine learning models by
"""

def one_hot_encoding(df):
    # Select numeric columns
    num_col = df.select_dtypes(include='number').columns

    # Select categorical columns (excluding the first column which could be the index c
    cat_col = df.columns.difference(num_col)
    cat_col = cat_col[1:]

    # Create a DataFrame with only categorical attributes
    df_cat = df[cat_col].copy()
```

```
    # Apply One-Hot Encoding
    df_cat_encoded = pd.get_dummies(df_cat, columns=cat_col)

    # Concatenate the original DataFrame with the encoded DataFrame
    df_encoded = pd.concat([df, df_cat_encoded], axis=1)

    # Remove the original categorical columns
    df_encoded.drop(columns=cat_col, inplace=True)

    return df_encoded

def main():
    # Load the data
    file_path = '/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Datasets/01_Cleaned_D
    df = pd.read_csv(file_path)

    # Apply One-Hot Encoding "function"
    df_encoded = one_hot_encoding(df)

    # Save the encoded data
    df_encoded.to_csv('/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Datasets/02_Enc

    print("One-Hot Encoding realizado e dados salvos com sucesso.")

if __name__ == '__main__':
    main()
```

Z-Score Normalization

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler

def normalization(df, cols):
    # Initialize the StandardScaler
    scaler = StandardScaler()

    # Apply normalization to each specified column
    for col in cols:
        arr = df[col].values.reshape(-1, 1)
        df[col] = scaler.fit_transform(arr)

    return df

def main():
    # Load the data
    file_path = '/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Datasets/02_Encoded_D
```

```
    df = pd.read_csv(file_path)

    # Select numeric columns
    num_col = list(df.select_dtypes(include='number').columns)
    if 'id' in num_col:
        num_col.remove('id')
    if 'label' in num_col:
        num_col.remove('label')

    print(f"Numeric columns selected for normalization: {num_col}")

    # Apply normalization
    df_normalized = normalization(df.copy(), num_col)

    # Save the normalized data
    df_normalized.to_csv('/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Datasets

    print("Data normalization completed and saved successfully.")

if __name__ == '__main__':
    main()
```

Label Encoding

```
import pandas as pd
import numpy as np
from sklearn import preprocessing

# Function for binary encoding
def label_encoding_binary(df):
    # Change attack labels to two categories: 'normal' and 'abnormal'
    bin_label = pd.DataFrame(df['label'].map(lambda x: 'normal' if x == 0 else 'abnorma

    # Create a DataFrame with binary labels (normal, abnormal)
    bin_data = df.copy()
    bin_data['label'] = bin_label

    # Encode binary labels (0, 1)
    le = preprocessing.LabelEncoder()
    enc_label = bin_label.apply(le.fit_transform)
    bin_data['label'] = enc_label

    # Save the LabelEncoder classes
    np.save("/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Datasets/03_Labelled/

    return bin_data

# Function for multi-class encoding
```

```
def label_encoding_multi(df):
    # Create a copy of the data to avoid altering the original
    multi_data = df.copy()
    multi_label = pd.DataFrame(multi_data.attack_cat)

    # Apply one-hot encoding to the 'attack_cat' column
    multi_data = pd.get_dummies(multi_data, columns=['attack_cat'])

    # Label encoding for multi-class labels
    le2 = preprocessing.LabelEncoder()
    enc_label = multi_label.apply(le2.fit_transform)
    multi_data['label'] = enc_label

    # Save encoder classes
    np.save("/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Datasets/03_Labelled/unsw

    return multi_data

# Main function that performs both classifications
def main():
    file_path = '/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Datasets/02_Normalize

    # Load the data
    df = pd.read_csv(file_path)

    # Apply binary encoding
    df_binary_encoded = label_encoding_binary(df)
    df_binary_encoded.to_csv('/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Datasets
    print("Binary classification applied and data saved.")

    # Apply multi-class encoding
    df_multi_encoded = label_encoding_multi(df)
    df_multi_encoded.to_csv('/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Datasets/
    print("Multi-class classification applied and data saved.")

if __name__ == '__main__':
    main()
```

Correlation Analysis

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Function to load data
def load_data(file_path):
    df = pd.read_csv(file_path)
    return df
```

```
# Function to analyze and plot correlation
def correlation_analysis(bin_data, multi_data):
    # Add the 'label' column to the numeric columns
    num_col = list(bin_data.select_dtypes(include='number').columns)
    num_col.append('label')

    # Correlation Matrix for Binary Labels
    plt.figure(figsize=(20,8))
    corr_bin = bin_data[num_col].corr()
    sns.heatmap(corr_bin, vmax=1.0, annot=False)
    plt.title('Correlation Matrix for Binary Labels', fontsize=16)
    plt.savefig('/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Plots/UNSW_zscore
    plt.show()

    # Update the list of numeric columns for multi-class data
    num_col = list(multi_data.select_dtypes(include='number').columns)

    # Correlation Matrix for Multi Labels
    plt.figure(figsize=(20,8))
    corr_multi = multi_data[num_col].corr()
    sns.heatmap(corr_multi, vmax=1.0, annot=False)
    plt.title('Correlation Matrix for Multi Labels', fontsize=16)
    plt.savefig('/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Plots/UNSW_zscore
    plt.show()

def main():

    # Load data
    file_path = '/home/mario/Downloads/Tese/Tese_Mario_Carneiro/'
    bin_data_path = '/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Datasets/03_I
    multi_data_path = '/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Datasets/03

    bin_data = load_data(bin_data_path)
    multi_data = load_data(multi_data_path)

    # Analyze and plot correlation
    correlation_analysis(bin_data, multi_data)

    print("Correlation analysis completed and graphs saved successfully.")

if __name__ == '__main__':
    main()
```

Feature Selection

```
import pandas as pd
```

```
# Function to perform feature selection with high correlation for binary classification
def feature_selection_binary(bin_data):
    # Add the 'label' column to the numeric columns
    num_col_bin = list(bin_data.select_dtypes(include='number').columns)

    # Find attributes with correlation greater than 0.3 with the binary label
    corr_bin = bin_data[num_col_bin].corr()
    corr_ybin = abs(corr_bin['label'])

    # Select only correlations greater than 0.3
    highest_corr_bin = corr_ybin[corr_ybin > 0.3]

    # Check the type of highest_corr_bin
    print(f"Type of highest_corr_bin: {type(highest_corr_bin)}")
    print(f"Contents of highest_corr_bin: {highest_corr_bin.head()}")

    # Sort the Series by correlation values
    highest_corr_bin = highest_corr_bin.sort_values(ascending=True)

    # Select the columns found using the Pearson correlation coefficient
    bin_cols = highest_corr_bin.index

    # Create a new DataFrame with only the selected columns
    bin_data = bin_data[bin_cols].copy()

    return bin_data

# Function to perform feature selection with high correlation for multi-class classificatio
def feature_selection_multi(multi_data):
    # Check for the presence of the 'label' column in the DataFrame
    if 'label' not in multi_data.columns:
        raise ValueError("The 'label' column is not present in the data.")

    # Select only numeric columns to calculate correlation
    num_col = list(multi_data.select_dtypes(include='number').columns)

    # Ensure 'label' is in the list of numeric columns
    if 'label' not in num_col:
        num_col.append('label')

    # Calculate correlation only on numeric columns
    corr_multi = multi_data[num_col].corr()

    # Check the correlation with the 'label' column
    if 'label' not in corr_multi:
        raise ValueError("Could not calculate correlation with the 'label' column.")

    # Find attributes with correlation greater than 0.1 with the multi-class label
    corr_ymulti = abs(corr_multi['label'])
```

```
    highest_corr_multi = corr_ymulti[corr_ymulti > 0.1]

    # Sort the Series by correlation values
    highest_corr_multi = highest_corr_multi.sort_values(ascending=True)

    # Print the correlations and selected columns
    print("\nComplete correlation with 'label':")
    print(corr_multi['label'])
    print("\nAttributes with high correlation with 'label':")
    print(highest_corr_multi)

    # Select the columns found using the Pearson correlation coefficient
    multi_cols = highest_corr_multi.index

    # Ensure 'label' is included in the selected columns
    if 'label' not in multi_cols:
        multi_cols = multi_cols.append(pd.Index(['label']))

    # Create a new DataFrame with only the selected columns
    multi_data_selected = multi_data[multi_cols].copy()

    return multi_data_selected

def main():
    # Path of the file with the correlation matrix for binary labels
    bin_data_path = '/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Datasets/03_L
    bin_data = pd.read_csv(bin_data_path)

    # Perform feature selection with high correlation for binary
    bin_data_selected = feature_selection_binary(bin_data)

    # Save the resulting dataset for binary
    bin_data_selected.to_csv('/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Data

    print("Binary feature selection completed and binary data saved successfully.")

    # Path of the file with the matrix of data for multi-class labels
    multi_data_path = '/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Datasets/03
    multi_data = pd.read_csv(multi_data_path)

    # Perform feature selection with high correlation for multi-class
    multi_data_selected = feature_selection_multi(multi_data)

    # Save the resulting dataset for multi-class
    multi_data_selected.to_csv('/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Da

    print("Multi-class feature selection completed and multi-class data saved successfu

if __name__ == '__main__':
```

```
    main()
```

Binary Training

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pickle
import time
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression, SGDClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn import metrics
from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA
from os import path

# **Load your binary classification dataset**
bin_data = pd.read_csv('/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipeline/Datasets/04

# Remove the 'id' column if it exists
if 'id' in bin_data.columns:
    bin_data = bin_data.drop(columns=['id'])

# Load the classes of the LabelEncoder from the saved file
le1 = LabelEncoder()
le1.classes_ = np.load("/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipeline/Datasets/03

# **Data Splitting**
X = bin_data.drop(columns=['label'], axis=1)
Y = bin_data['label']

# Print all features
print("Feature in the dataset:")
print(bin_data.columns)

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.20, random_state=50)

# **PCA for Dimensionality Reduction**
pca = PCA(n_components=0.95, random_state=50)  # Keeps 95% of the variance
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

# List of models to train
```

```
models = {
    "LogisticRegression": LogisticRegression(random_state=123, max_iter=2000, n_jobs=-1
    "KNN": KNeighborsClassifier(n_neighbors=4, n_jobs=-1),
    "RandomForest": RandomForestClassifier(random_state=123, n_jobs=-1, n_estimators=75
    "DecisionTree": DecisionTreeClassifier(random_state=123),
    "MLPClassifier": MLPClassifier(random_state=123, solver='adam', max_iter=1500),
    "SGDClassifier": SGDClassifier(loss='hinge', random_state=123, max_iter=1000, tol=1
}

# Loop through models and train
for name, model in models.items():
    print(f"\nTraining {name}...")
    start_time = time.time()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    print(f"Mean Absolute Error - {metrics.mean_absolute_error(y_test, y_pred)}")
    print(f"Mean Squared Error - {metrics.mean_squared_error(y_test, y_pred)}")
    print(f"Root Mean Squared Error - {np.sqrt(metrics.mean_squared_error(y_test, y_pre
    print(f"R2 Score - {metrics.explained_variance_score(y_test, y_pred) * 100}")
    print(f"Accuracy - {metrics.accuracy_score(y_test, y_pred) * 100}")
    print(metrics.classification_report(y_test, y_pred, target_names=le1.classes_))

    # Save predictions
    df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
    df.to_csv(f'/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipeline/Predictions/UNS

# Plot predictions
    plt.figure(figsize=(20, 8))
    plt.plot(y_pred[:200], label="prediction", linewidth=2.0, color='blue', alpha=0.7)
    plt.plot(y_test[:200].values, label="real_values", linewidth=2.0, color='lightcoral
    plt.legend(loc="best")
    plt.title(f"{name} Binary Classification")
    plt.savefig(f'/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipeline/Plots/UNSW_zs
    plt.show()


    # Save model
    pkl_filename = f"/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipeline/Models/UNS
    if not path.isfile(pkl_filename):
        with open(pkl_filename, 'wb') as file:
            pickle.dump(model, file)
        print(f"Saved {name} model to disk in {time.time() - start_time:.2f} seconds\n"
    else:
        print(f"{name} model already saved\n")
```

Binary Bagging

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import pickle
import seaborn as sns

# **Load Data from a CSV File**
data = pd.read_csv('/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Datasets/04_Select

# Remove the 'id' column if it exists
if 'id' in data.columns:
    data = data.drop(columns=['id'])

# Separate the features from the labels
X = data.drop(columns=['label'])
y = data['label']

# **Split the Data into Training and Testing Sets**
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=50)

# **Load the Models**
model_paths = {
    "LogisticRegression": "/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Models/UNSW
    "KNN": "/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Models/UNSW_zscore_knn_bin
    "RandomForest": "/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Models/UNSW_zscor
    "DecisionTree": "/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Models/UNSW_zscor
    "MLPClassifier": "/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Models/UNSW_zsco
    #"SGDClassifier": "/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Models/UNSW_zsc
}

# Load the trained models
models = {name: pickle.load(open(path, 'rb')) for name, path in model_paths.items()}

# **Create the ensemble using VotingClassifier**
voting_model = VotingClassifier(estimators=[
    ('lr', models["LogisticRegression"]),  # LogisticRegression
    ('knn', models["KNN"]),  # KNN
    ('rf', models["RandomForest"]),  # RandomForest
    ('dt', models["DecisionTree"]),  # DecisionTree
    ('mlp', models["MLPClassifier"]),  # MLPClassifier
   # ('sgd', models["SGDClassifier"])  # SGDClassifier
], voting='hard')  # or 'soft' to use probabilities

# **Train the ensemble**
voting_model.fit(X_train, y_train)
```

```
# **Evaluate Performance**
y_pred_train = voting_model.predict(X_train)
y_pred_test = voting_model.predict(X_test)

# Display results
print("Train Accuracy: ", accuracy_score(y_train, y_pred_train) * 100)
print("Test Accuracy: ", accuracy_score(y_test, y_pred_test) * 100)
print("Classification Report (Test):")
print(classification_report(y_test, y_pred_test))

# Calculate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred_test)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.ylabel('True Values')
plt.xlabel('Predicted Values')
plt.title("UNSW Ensembled Binary Classification")

plt.savefig('/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Plots/UNSW_ensemble_c

# Show the plot
plt.show()

# **Save the Combined Model**
ensemble_model_path = '/home/mario/Downloads/Tese/Tese_Mario_Carneiro/Pipe5/Models/UNSW
with open(ensemble_model_path, 'wb') as file:
    pickle.dump(voting_model, file)

print("Combined model saved successfully!")
```

This folder contains additional scripts; however, to avoid redundancy in the appendices, some scripts have been omitted.

Appendix D

Testbed Implementation

This appendix includes the script for the implementation of the testbed, which is designed to automate cyberattack scenarios and validate the models.

D.1 Testbed Script

```
import subprocess
import random
import time
import csv
import os
from datetime import datetime, timedelta

"""
This script is designed to capture network packets and record events related to network

1. Packet Capture:
   - The ‘start_packet_capture‘ function starts capturing packets using the ‘tcpdump‘ c
   It captures packets from a specific network interface and saves them in an output fi
   The capture time and maximum file size are configurable.
   The function also records the start of the capture in a CSV file.

2. Packet Capture Stops:
    - The ‘stop_packet_capture‘ function terminates the packet capture process.
    After capturing, it records the end in the CSV file and captures any errors generat

3. Execution of network scans:
    - The ‘execute_scan‘ function performs a network scan using the ‘nmap‘ command.
    The function records information about the scan, including the type of scan, the de

4. Execution os DoS Attacks:

5. Event Log:
    - The ‘log_event‘ function records events in the CSV file, ensuring that all releva
    This includes a timestamp for each event recorded.

6. CSV file configuration:
```

```
   - The ‘setup_csv‘ function sets up a CSV file to record events.
   If the file already exists, it opens in add mode; otherwise, it creates a new one and wr

7. Filename Management:
    - The ‘get_next_file_index‘ function generates a sequential index for the output files
    This ensures that new files don’t overwrite existing ones.

8. Main Execution Flow:
    - The main flow of the script will call these functions to perform packet captures and

This script is useful in cyber security scenarios for monitoring and recording network acti

# Common functions

def get_next_file_index(prefix, extension, directory):
    files = [f for f in os.listdir(directory) if f.startswith(prefix) and f.endswith(extens
    if not files:
        return 1
    indices = [int(f[len(prefix):-len(extension)]) for f in files if f[len(prefix):-len(ext
    return max(indices) + 1 if indices else 1

def setup_csv(csv_file):
    # Check if the CSV file already exists
    file_exists = os.path.isfile(csv_file)

    # Define the important fields for training the model
    fieldnames = [
    'timestamp', 'event_type', 'file_index', 'interface', 'output_file',
    'capture_duration', 'capture_command', 'scan_type', 'target_ip',
    'attack_command', 'attack_type', 'attack_intensity', 'packets_sent',
    'latency', 'status', 'error', 'additional_info', 'protocol', 'port',
    'len', 'frame_len', 'ip_src', 'ip_dst'
]

    # Open CSV file for writing (append mode)
    csv_file_object = open(csv_file, 'a', newline='')
    csv_writer = csv.DictWriter(csv_file_object, fieldnames=fieldnames)

    # If the file did not exist, write the header
    if not file_exists:
        csv_writer.writeheader()

    return csv_file_object, csv_writer

def log_event(csv_writer, event_type, event_info):
    event_info['event_type'] = event_type
    event_info['timestamp'] = str(datetime.now())

    # Filter the dictionary to keep only the fields that are in fieldnames
```

```
    filtered_event_info = {k: v for k, v in event_info.items() if k in csv_writer.field

    # Add missing fields with empty value
    for field in csv_writer.fieldnames:
        if field not in filtered_event_info:
            filtered_event_info[field] = ''

    csv_writer.writerow(filtered_event_info)

def start_packet_capture(interface, output_file, duration, csv_writer, max_size_mb=30):

    # Calculation of maximum file size in megabytes for the -C option
    max_size_kb = max_size_mb * 1024  #Convert MB to KB

    tcpdump_command = f"sudo timeout {duration} tcpdump -i {interface} -C {max_size_kb}
    try:
        # Capture standard output (stdout) and error output (stderr)
        tcpdump_process = subprocess.Popen(tcpdump_command, shell=True, stdout=subproce

        log_event(csv_writer, "packet_capture_start", {
            "interface": interface,
            "output_file": output_file,
            "capture_duration": duration,
            "capture_command": tcpdump_command
        })
        print(f"Started packet capture, saving to {output_file}")

        return tcpdump_process
    except Exception as e:
        # Error log if packet capture fails to start
        log_event(csv_writer, "error", {
            "message": "Failed to start packet capture",
            "error": str(e)
        })
        return None

def stop_packet_capture(tcpdump_process, csv_writer, interface, output_file):
    if tcpdump_process:
        tcpdump_process.terminate()
        stdout, stderr = tcpdump_process.communicate()  # Capture the output and error

        # Check and log any error output for debugging
        if stderr:
            log_event(csv_writer, "erg   ror", {
                "message": "Error during packet capture",
                "error": stderr.decode('utf-8')
            })

        tcpdump_process.wait()
```

```
        # Log of the end of packet capture
        log_event(csv_writer, "packet_capture_end", {
            "interface": interface,
            "output_file": output_file
        })

        print(f"Packet capture completed, file saved to {output_file}")

def execute_scan(scan_type, target_ip, csv_writer):
    command = f"sudo nmap {scan_type} {target_ip}"
    start_time = datetime.now()
    try:
        result = subprocess.run(command, shell=True, capture_output=True, text=True)
        end_time = datetime.now()

        scan_info = {
            "scan_type": scan_type,
            "target_ip": target_ip,
            "command": command,
            "capture_duration": str(end_time - start_time),
            "status": "success" if result.returncode == 0 else "failed",
            "additional_info": result.stdout if result.returncode == 0 else result.stderr
        }

        log_event(csv_writer, "scan", scan_info)
    except Exception as e:
        log_event(csv_writer, "error", {
            "message": "Failed to execute scan",
            "error": str(e)
        })

def execute_dos_attack(ip, attack_command, csv_writer):
    duration = random.randint(15, 60)  # Random attack duration (Prevent Overfitting of lat
    start_time = time.time()
    try:
        process = subprocess.Popen(attack_command, shell=True)
        time.sleep(duration)
        subprocess.run(f"pkill -f '{attack_command.split()[0]}'", shell=True)
        process.terminate()
        attack_duration = time.time() - start_time
        attack_info = {
            "attack_command": attack_command,
            "target_ip": ip,
            "attack_type": "DoS",
            "capture_duration": attack_duration
        }
        log_event(csv_writer, "dos_attack", attack_info)
    except Exception as e:
```

```
        log_event(csv_writer, "error", {
            "message": "Failed to execute DoS attack",
            "error": str(e)
        })

# Function to clear ARP cache and kill attack-related processes
def cleanup_environment():
    # Clear ARP cache
    #subprocess.run("sudo ip -s -s neigh flush all", shell=True)

    # Kill ARP spoofing, DoS, etc. processes.
    subprocess.run("sudo pkill tcpdump", shell=True)
    #subprocess.run("sudo pkill arpspoof", shell=True)
    subprocess.run("sudo pkill hping3", shell=True)
    subprocess.run("sudo pkill nping", shell=True)
    subprocess.run("sudo pkill ping", shell=True)

# Main function
def main():
    # Configs
    #target_ip = '192.168.1.92'  # IP PC Portátil 1
    #target_ip = '192.168.1.93'  # IP PC Portátil 2
    target_ip = '192.168.1.64'  # IP Pc Fixo
    gateway_ip = '192.168.1.254'  # Network Gateway IP
    interface = 'eth0'  # Network interface
    base_directory = '/home/pi/Tese/Tese_Mario_Carneiro/Datasets/01_Dataset_Pcap'

    # Configure randomness to choose the scenario
    scenarios = [1, 2, 3]
    total_duration = 8 * 3600  # 8 hours (Running Script hours)
    end_time = datetime.now() + timedelta(seconds=total_duration)

    while datetime.now() < end_time:
        scenario = random.choice(scenarios)
        print(f"Running Scenario {scenario}")

        # Clean the environment before each scenario
        # Comment: We ensure that each scenario starts in a clean state to avoid interf
        # caused by residues from previous scenarios, especially from attack or spoofin
        cleanup_environment()

        if scenario == 1:
            prefix = 'scan_'
            sub_directory = 'Scenario1'
            scan_types = ['-sn', '-F', '-A', '-p 1-100', '-T4', '-sS', '-O', '-sU']
            directory = os.path.join(base_directory, sub_directory)
            os.makedirs(directory, exist_ok=True)

            # CSV and pcap files setup
```

```
        file_index = get_next_file_index(prefix, ".pcap", directory)
        output_file = os.path.join(directory, f"{prefix}{file_index}.pcap")
        csv_file = os.path.join(directory, f"{prefix}{file_index}.csv")

        # CSV file setup and writer initialization
        csv_file_object, csv_writer = setup_csv(csv_file)

        # Packet capture
        capture_duration = random.randint(60, 150)  # Capture duration between 1 and 3 
        tcpdump_process = start_packet_capture(interface, output_file, capture_duration

        # Running an Nmap scan
        scan_type = random.choice(scan_types)
        execute_scan(scan_type, target_ip, csv_writer)

    elif scenario == 2:
        prefix = 'dos_'
        sub_directory = 'Scenario2'
        dos_commands = [
            f"sudo hping3 --flood --rand-source --syn {target_ip}",
            f"sudo ping -i 0.002 {target_ip}",          #ping -f {target_ip}" -> Ping -
            f"sudo nping --tcp -p 80 --flags syn --data-length 1200 --rate 1000 {target
        ]
        directory = os.path.join(base_directory, sub_directory)
        os.makedirs(directory, exist_ok=True)

        # CSV and pcap files setup
        file_index = get_next_file_index(prefix, ".pcap", directory)
        output_file = os.path.join(directory, f"{prefix}{file_index}.pcap")
        csv_file = os.path.join(directory, f"{prefix}{file_index}.csv")
        csv_file_object, csv_writer = setup_csv(csv_file)

        # Packet capture
        capture_duration = random.randint(60, 150)  # Capture duration between 1 and 3 
        tcpdump_process = start_packet_capture(interface, output_file, capture_duration

        # Executing a DoS attack
        attack_command = random.choice(dos_commands)
        execute_dos_attack(target_ip, attack_command, csv_writer)

    elif scenario == 3:
        # Scenario 3 settings
        prefix = 'traffic_'
        sub_directory = 'Scenario3'
        directory = os.path.join(base_directory, sub_directory)
        os.makedirs(directory, exist_ok=True)

        # CSV and pcap files setup
        file_index = get_next_file_index(prefix, ".pcap", directory)
```

```
            output_file = os.path.join(directory, f"{prefix}{file_index}.pcap")
            csv_file = os.path.join(directory, f"{prefix}{file_index}.csv")
            csv_file_object, csv_writer = setup_csv(csv_file)

            # Packet capture
            capture_duration = random.randint(120, 300)  # Capture duration between 2 a
            tcpdump_process = start_packet_capture(interface, output_file, capture_dura

            # Capturing benign traffic
            print(f"Capturing benign traffic for: {capture_duration}...")
            time.sleep(capture_duration)
            log_event(csv_writer, "traffic", {"duration": capture_duration})
            print("Benign traffic capture completed.")

        # Wait for packet capture to finish
        stop_packet_capture(tcpdump_process, csv_writer, interface, output_file)

        # Random idle period between 10s and 25s
        inactivity_period = random.randint(10, 25)
        print(f"Inactivity period: {inactivity_period} seconds.")
        time.sleep(inactivity_period)

        # Update file names for the next iteration
        file_index += 1
        output_file = os.path.join(directory, f"{prefix}{file_index}.pcap")
        csv_file = os.path.join(directory, f"{prefix}{file_index}.csv")

        # Close the CSV file of the current iteration
        csv_file_object.close()

    print("Execution completed.")

if __name__ == "__main__":
    main()
```

Appendix A

Training Evaluations

This appendix contains images illustrating the training evaluations for both binary and multi-class classification models.

A.1 Binary Classification - Min-Max-Score - UNSW

A.1.1 Label Correlation

```
Tipo de highest_corr_bin: <class 'pandas.core.series.Series'>
Conteúdo de highest_corr_bin: id        0.782954
sttl      0.734732
dttl      0.735507
tcprtt    0.588270
synack    0.513911
Name: label, dtype: float64
Seleção de features binária concluída e dados binários salvos com sucesso.
```

Figure A.1: Label Correlation for Binary Classification using Min-Max-Score

A.1.2 Logistic Regression

```
Mean Absolute Error - 0.07591907765934305
Mean Squared Error - 0.07591907765934305
Root Mean Squared Error - 0.27553416786188795
R2 Score - 40.68748735664379
Accuracy - 92.40809223406569
              precision    recall  f1-score   support

    abnormal       0.77      0.70      0.74       691
      normal       0.95      0.96      0.96      3906

    accuracy                           0.92      4597
   macro avg       0.86      0.83      0.85      4597
weighted avg       0.92      0.92      0.92      4597
```

Figure A.2: Logistic Regression Evaluation

A.1.3 K-Nearest Neighbors (KNN)

```
Mean Absolute Error - 0.00891886012616924
Mean Squared Error - 0.00891886012616924
Root Mean Squared Error - 0.09443971688950174
R2 Score - 93.01782926263576
Accuracy - 99.10811398738308
              precision    recall  f1-score   support

    abnormal       0.97      0.97      0.97       691
      normal       0.99      1.00      0.99      3906

    accuracy                           0.99      4597
   macro avg       0.98      0.98      0.98      4597
weighted avg       0.99      0.99      0.99      4597
```

Figure A.3: K-Nearest Neighbors Evaluation

A.1.4 Decision Tree

```
Training DecisionTree...
Mean Absolute Error - 0.009136393299978247
Mean Squared Error - 0.009136393299978247
Root Mean Squared Error - 0.09558448252712491
R2 Score - 92.84717637268872
Accuracy - 99.08636067000216
              precision    recall  f1-score   support

    abnormal       0.97      0.97      0.97       691
      normal       0.99      1.00      0.99      3906

    accuracy                           0.99      4597
   macro avg       0.98      0.98      0.98      4597
weighted avg       0.99      0.99      0.99      4597

Saved DecisionTree model to disk in 0.90 seconds
```

Figure A.4: Decision Tree Evaluation

A.1.5 Random Forest

```
Training RandomForest...
Mean Absolute Error - 0.009136393299978247
Mean Squared Error - 0.009136393299978247
Root Mean Squared Error - 0.09558448252712491
R2 Score - 92.84717637268872
Accuracy - 99.08636067000216
              precision    recall  f1-score   support

    abnormal       0.97      0.97      0.97       691
      normal       0.99      1.00      0.99      3906

    accuracy                           0.99      4597
   macro avg       0.98      0.98      0.98      4597
weighted avg       0.99      0.99      0.99      4597

Saved RandomForest model to disk in 1.13 seconds
```

Figure A.5: Random Forest Evaluation

A.1.6 MLP Classifier

```
Training MLPClassifier...
Mean Absolute Error - 0.06221448770937568
Mean Squared Error - 0.06221448770937568
Root Mean Squared Error - 0.24942832178679245
R2 Score - 54.31919278144945
Accuracy - 93.77855122906243
              precision    recall  f1-score   support

    abnormal       0.71      1.00      0.83       691
      normal       1.00      0.93      0.96      3906

    accuracy                           0.94      4597
   macro avg       0.85      0.96      0.90      4597
weighted avg       0.96      0.94      0.94      4597

Saved MLPClassifier model to disk in 1.71 seconds
```

Figure A.6: MLP Classifier Evaluation

A.1.7 SGD Classifier

```
Training SGDClassifier...
Mean Absolute Error - 0.1561888187948662
Mean Squared Error - 0.1561888187948662
Root Mean Squared Error - 0.39520731116069474
R2 Score - -3.1891268248114057
Accuracy - 84.38111812051338
              precision    recall  f1-score   support

    abnormal       0.49      1.00      0.66       691
      normal       1.00      0.82      0.90      3906

    accuracy                           0.84      4597
   macro avg       0.75      0.91      0.78      4597
weighted avg       0.92      0.84      0.86      4597

Saved SGDClassifier model to disk in 1.09 seconds
```

Figure A.7: SGD Classifier Evaluation

A.1.8 Ensembled Model

```
Train Accuracy:  99.89665488169703
Test Accuracy:  99.41266043071568
Classification Report (Test):
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       691
           1       1.00      0.99      1.00      3906

    accuracy                           0.99      4597
   macro avg       0.98      0.99      0.99      4597
weighted avg       0.99      0.99      0.99      4597
```

Figure A.8: Ensembled Model Evaluation

A.2 Binary Classification - Z-Score - UNSW

A.2.1 Label Correlation

```
Tipo de highest_corr_bin: <class 'pandas.core.series.Series'>
Conteúdo de highest_corr_bin: id        0.782954
sttl      0.734732
dttl      0.735507
tcprtt    0.588270
synack    0.513911
Name: label, dtype: float64
Seleção de features binária concluída e dados binários salvos com sucesso.
```

Figure A.9: Label Correlation for Binary Classification using Z-Score

A.2.2 Logistic Regression

```
Mean Absolute Error - 0.07591907765934305
Mean Squared Error - 0.07591907765934305
Root Mean Squared Error - 0.27553416786188795
R2 Score - 40.68748735664379
Accuracy - 92.40809223406569
              precision    recall  f1-score   support

    abnormal       0.77      0.70      0.74       691
      normal       0.95      0.96      0.96      3906

    accuracy                           0.92      4597
   macro avg       0.86      0.83      0.85      4597
weighted avg       0.92      0.92      0.92      4597
```

Figure A.10: Logistic Regression Evaluation (Z-Score)

A.2.3 K-Nearest Neighbors (KNN)

```
Training KNN...
13.45s - pydevd: Sending message related to process bei
Mean Absolute Error - 0.00891886012616924
Mean Squared Error - 0.00891886012616924
Root Mean Squared Error - 0.09443971688950174
R2 Score - 93.01782926263576
Accuracy - 99.10811398738308
              precision    recall  f1-score   support

    abnormal       0.97      0.97      0.97       691
      normal       0.99      1.00      0.99      3906

    accuracy                           0.99      4597
   macro avg       0.98      0.98      0.98      4597
weighted avg       0.99      0.99      0.99      4597
```

Figure A.11: K-Nearest Neighbors Evaluation (Z-Score)

A.2.4 Decision Tree

```
Training DecisionTree...
Mean Absolute Error - 0.009136393299978247
Mean Squared Error - 0.009136393299978247
Root Mean Squared Error - 0.09558448252712491
R2 Score - 92.84717637268872
Accuracy - 99.08636067000216
              precision    recall  f1-score   support

    abnormal       0.97      0.97      0.97       691
      normal       0.99      1.00      0.99      3906

    accuracy                           0.99      4597
   macro avg       0.98      0.98      0.98      4597
weighted avg       0.99      0.99      0.99      4597

DecisionTree model already saved
```

Figure A.12: Decision Tree Evaluation (Z-Score)

A.2.5 Random Forest

```
Training RandomForest...
Mean Absolute Error - 0.009136393299978247
14.67s - pydevd: Sending message related to process being
Mean Squared Error - 0.009136393299978247
Root Mean Squared Error - 0.09558448252712491
R2 Score - 92.84717637268872
Accuracy - 99.08636067000216
              precision    recall  f1-score   support

    abnormal       0.97      0.97      0.97       691
      normal       0.99      1.00      0.99      3906

    accuracy                           0.99      4597
   macro avg       0.98      0.98      0.98      4597
weighted avg       0.99      0.99      0.99      4597
```

Figure A.13: Random Forest Evaluation (Z-Score)

A.2.6 MLP Classifier

```
Training MLPClassifier...
Mean Absolute Error - 0.06852294974983685
Mean Squared Error - 0.06852294974983685
Root Mean Squared Error - 0.26176888613782356
R2 Score - 46.36897629755106
Accuracy - 93.14770502501631
              precision    recall  f1-score   support

    abnormal       0.78      0.76      0.77       691
      normal       0.96      0.96      0.96      3906

    accuracy                           0.93      4597
   macro avg       0.87      0.86      0.86      4597
weighted avg       0.93      0.93      0.93      4597

MLPClassifier model already saved
```

Figure A.14: MLP Classifier Evaluation (Z-Score)

A.2.7 SGD Classifier

```
Training SGDClassifier...
Mean Absolute Error - 0.1561888187948662
Mean Squared Error - 0.1561888187948662
Root Mean Squared Error - 0.39520731116069474
R2 Score - -3.1891268248114057
Accuracy - 84.38111812051338
              precision    recall  f1-score   support

    abnormal       0.49      1.00      0.66       691
      normal       1.00      0.82      0.90      3906

    accuracy                           0.84      4597
   macro avg       0.75      0.91      0.78      4597
weighted avg       0.92      0.84      0.86      4597

SGDClassifier model already saved
```

Figure A.15: SGD Classifier Evaluation (Z-Score)

A.2.8 Ensembled Model

```
Train Accuracy:  99.9564862659777
Test Accuracy:  99.65194692190559
Classification Report (Test):
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       691
           1       1.00      1.00      1.00      3906

    accuracy                           1.00      4597
   macro avg       0.99      0.99      0.99      4597
weighted avg       1.00      1.00      1.00      4597
```

Figure A.16: Ensembled Model Evaluation (Z-Score)

A.3 Multi-Class Classification - Min-Max-Score - UNSW

A.3.1 Label Correlation

```
Atributos com alta correlação com 'label':
smean    0.115371
dmean    0.124986
sttl     0.285711
label    1.000000
Name: label, dtype: float64
Seleção de features multiclasse concluída e dados multiclasse salvos com sucesso.
```

Figure A.17: Label Correlation for Multi-Class Classification using Min-Max-Score

A.3.2 Logistic Regression

```
Mean Absolute Error - 0.11399564902102974
Mean Squared Error - 0.15286439448875996
Root Mean Squared Error - 0.39097876475425103
R2 Score - 2.261344548590183
Accuracy - 90.54387237128354
                precision    recall  f1-score   support

           DoS       0.55      0.15      0.23       514
        Normal       0.95      0.97      0.96      5877
Reconnaissance       0.64      0.98      0.78       504

      accuracy                           0.91      6895
     macro avg       0.71      0.70      0.65      6895
  weighted avg       0.90      0.91      0.89      6895
```

Figure A.18: Logistic Regression Evaluation (Multi-Class)

A.3.3 K-Nearest Neighbors (KNN)

```
Training KNN...
14.24s - pydevd: Sending message related to process being replaced timed-out after 5 seconds
Mean Absolute Error - 0.03625815808556925
Mean Squared Error - 0.050761421319796954
Root Mean Squared Error - 0.22530295452966645
R2 Score - 65.62526624125056
Accuracy - 97.09934735315446
                precision    recall  f1-score   support

           DoS       0.81      0.86      0.83       514
        Normal       0.99      0.98      0.99      5877
Reconnaissance       0.92      0.94      0.93       504

      accuracy                           0.97      6895
     macro avg       0.91      0.93      0.92      6895
  weighted avg       0.97      0.97      0.97      6895
```

Figure A.19: K-Nearest Neighbors Evaluation (Multi-Class)

A.3.4 Decision Tree

```
Training DecisionTree...
Mean Absolute Error - 0.03770848440899202
Mean Squared Error - 0.050761421319796954
Root Mean Squared Error - 0.22530295452966645
R2 Score - 65.62298671750005
Accuracy - 96.88179840464105
                precision    recall  f1-score   support

           DoS       0.82      0.81      0.82       514
        Normal       0.99      0.98      0.99      5877
Reconnaissance       0.92      0.94      0.93       504

      accuracy                           0.97      6895
     macro avg       0.91      0.91      0.91      6895
  weighted avg       0.97      0.97      0.97      6895

Saved DecisionTree model to disk
```

Figure A.20: Decision Tree Evaluation (Multi-Class)

A.3.5 Random Forest

```
Training RandomForest...
Mean Absolute Error - 0.03451776649746193
Mean Squared Error - 0.047280638143582306
Root Mean Squared Error - 0.21744111419780368
R2 Score - 68.00611482246073
Accuracy - 97.18636693255984
                precision    recall  f1-score   support

           DoS       0.85      0.82      0.83       514
        Normal       0.99      0.99      0.99      5877
Reconnaissance       0.91      0.96      0.93       504

      accuracy                           0.97      6895
     macro avg       0.92      0.92      0.92      6895
  weighted avg       0.97      0.97      0.97      6895

Saved RandomForest model to disk
```

Figure A.21: Random Forest Evaluation (Multi-Class)

A.3.6 MLP Classifier

```
Training MLPClassifier...
Mean Absolute Error - 0.05337200870195794
Mean Squared Error - 0.07048585931834662
Root Mean Squared Error - 0.26549173116755753
R2 Score - 52.83998740563127
Accuracy - 95.51849166062364
                precision    recall  f1-score   support

           DoS       0.84      0.57      0.68       514
        Normal       0.97      0.99      0.98      5877
Reconnaissance       0.88      0.95      0.91       504

      accuracy                           0.96      6895
     macro avg       0.90      0.84      0.86      6895
  weighted avg       0.95      0.96      0.95      6895

Saved MLPClassifier model to disk
```

Figure A.22: MLP Classifier Evaluation (Multi-Class)

A.3.7 SGD Classifier

```
Training SGDClassifier...
Mean Absolute Error - 0.1192168237853517
Mean Squared Error - 0.16330674401740392
Root Mean Squared Error - 0.4041122913466057
R2 Score - -4.022789538695659
Accuracy - 90.28281363306743
                precision    recall  f1-score   support

           DoS       0.49      0.11      0.18       514
        Normal       0.95      0.97      0.96      5877
Reconnaissance       0.63      0.98      0.76       504

      accuracy                           0.90      6895
     macro avg       0.69      0.68      0.63      6895
  weighted avg       0.89      0.90      0.88      6895

Saved SGDClassifier model to disk
```

Figure A.23: SGD Classifier Evaluation (Multi-Class)

A.3.8 Ensembled Model

```
Train Accuracy:  98.2920859396247
Test Accuracy:  96.99804220143572
Classification Report (Test):
              precision    recall  f1-score   support

           0       0.84      0.81      0.83       352
           1       0.98      0.99      0.99      3906
           2       0.92      0.94      0.93       339

    accuracy                           0.97      4597
   macro avg       0.92      0.91      0.91      4597
weighted avg       0.97      0.97      0.97      4597

Modelo combinado salvo com sucesso!
```

Figure A.24: Ensembled Model Evaluation (Multi-Class)

A.4 Multi-Class Classification - Z-Score - UNSW

A.4.1 Label Correlation

```
Atributos com alta correlação com 'label':
smean    0.115371
dmean    0.124986
sttl     0.285711
label    1.000000
Name: label, dtype: float64
Seleção de features multiclasse concluída e dados multiclasse salvos com sucesso.
```

Figure A.25: Label Correlation for Multi-Class Classification using Min-Max-Score

A.4.2 Logistic Regression

```
Mean Absolute Error - 0.11196519216823786
Mean Squared Error - 0.1488034807831762
Root Mean Squared Error - 0.3857505421683503
R2 Score - 4.7601584838887545
Accuracy - 90.64539521392312
                precision    recall  f1-score   support

           DoS       0.57      0.16      0.25       514
        Normal       0.95      0.97      0.96      5877
Reconnaissance       0.65      0.98      0.78       504

      accuracy                           0.91      6895
     macro avg       0.72      0.70      0.66      6895
  weighted avg       0.90      0.91      0.89      6895
```

Figure A.26: Logistic Regression Evaluation (Multi-Class)

A.4.3 K-Nearest Neighbors (KNN)

```
Training KNN...
15.43s - pydevd: Sending message related to process being
Mean Absolute Error - 0.03872371283538796
Mean Squared Error - 0.05438723712835388
Root Mean Squared Error - 0.23321071400849894
R2 Score - 63.17144440597748
Accuracy - 96.9108049311095
                 precision    recall  f1-score   support

           DoS       0.80      0.85      0.82       514
        Normal       0.99      0.98      0.99      5877
Reconnaissance       0.92      0.93      0.93       504

      accuracy                           0.97      6895
     macro avg       0.90      0.92      0.91      6895
  weighted avg       0.97      0.97      0.97      6895

KNN model already saved
```

Figure A.27: K-Nearest Neighbors Evaluation (Multi-Class)

A.4.4 Decision Tree

```
Training DecisionTree...
Mean Absolute Error - 0.03857868020304569
Mean Squared Error - 0.05221174764321972
Root Mean Squared Error - 0.22849890074838375
R2 Score - 64.63883083226835
Accuracy - 96.82378535170413
                 precision    recall  f1-score   support

           DoS       0.81      0.81      0.81       514
        Normal       0.99      0.98      0.99      5877
Reconnaissance       0.92      0.94      0.93       504

      accuracy                           0.97      6895
     macro avg       0.91      0.91      0.91      6895
  weighted avg       0.97      0.97      0.97      6895

DecisionTree model already saved
```

Figure A.28: Decision Tree Evaluation (Multi-Class)

A.4.5 Random Forest

```
Training RandomForest...
Mean Absolute Error - 0.03509789702683104
Mean Squared Error - 0.048440899202320524
Root Mean Squared Error - 0.22009293310399705
R2 Score - 67.21289754538033
Accuracy - 97.15736040609137
                precision    recall  f1-score   support

           DoS       0.85      0.82      0.83       514
        Normal       0.99      0.99      0.99      5877
Reconnaissance       0.91      0.95      0.93       504

      accuracy                           0.97      6895
     macro avg       0.91      0.92      0.92      6895
  weighted avg       0.97      0.97      0.97      6895

RandomForest model already saved
```

Figure A.29: Random Forest Evaluation (Multi-Class)

A.4.6 MLP Classifier

```
Training MLPClassifier...
Mean Absolute Error - 0.050181290790427845
Mean Squared Error - 0.06845540246555475
Root Mean Squared Error - 0.26163983348403724
R2 Score - 53.989579727055535
Accuracy - 95.89557650471356
                precision    recall  f1-score   support

           DoS       0.84      0.64      0.72       514
        Normal       0.97      0.99      0.98      5877
Reconnaissance       0.88      0.94      0.91       504

      accuracy                           0.96      6895
     macro avg       0.90      0.86      0.87      6895
  weighted avg       0.96      0.96      0.96      6895

MLPClassifier model already saved
```

Figure A.30: MLP Classifier Evaluation (Multi-Class)

A.4.7 SGD Classifier

```
Training SGDClassifier...
Mean Absolute Error - 0.11559100797679478
Mean Squared Error - 0.1559100797679478
Root Mean Squared Error - 0.39485450455572596
R2 Score - 2.9122910495924748
Accuracy - 90.45685279187818
                precision    recall  f1-score   support

           DoS       1.00      0.02      0.04       514
        Normal       0.94      0.98      0.96      5877
Reconnaissance       0.64      0.98      0.77       504

      accuracy                           0.90      6895
     macro avg       0.86      0.66      0.59      6895
  weighted avg       0.92      0.90      0.87      6895

SGDClassifier model already saved
```

Figure A.31: SGD Classifier Evaluation (Multi-Class)

A.4.8 Ensembled Model

```
Train Accuracy:  98.29752515637749
Test Accuracy:  97.04154883619752
Classification Report (Test):
              precision    recall  f1-score   support

           0       0.84      0.81      0.83       352
           1       0.99      0.99      0.99      3906
           2       0.93      0.94      0.93       339

    accuracy                           0.97      4597
   macro avg       0.92      0.91      0.92      4597
weighted avg       0.97      0.97      0.97      4597

Modelo combinado salvo com sucesso!
```

Figure A.32: Ensembled Model Evaluation (Multi-Class)