Automated Intrusion Detection and Prevention for IoT Security Using Machine Learning and Large Language Models

University of Turku
Department of Computing
Master of Science (Tech) Thesis
Cyber Security Engineering
June 2026
Claudiu Iulian Cercel

Supervisors:
Tahir Mohammad (University of Turku)
Ismayil Hasanov (University of Turku)
Madura Rajapakshe (Finnplay Technologies Oy)

UNIVERSITY OF TURKU
Department of Computing

Claudiu Iulian Cercel: Automated Intrusion Detection and Prevention for IoT Security Using Machine Learning and Large Language Models

Master of Science (Tech) Thesis, 76 p., 19 app. p.
Cyber Security Engineering
June 2026

The Internet of Things (IoT) has recently seen exponential growth in usage and the introduction of severe security vulnerabilities that expose edge networks to sophisticated threats, such as Distributed Denial of Service (DDoS) attacks and zero-day exploits. Traditional Intrusion Detection Systems (IDS) often struggle in these constrained environments where signature-based methods fail against novel attacks, while anomaly-based approaches frequently generate false-positive results. Furthermore, modern automated tools that include Machine Learning (ML), Deep Learning (DL), and Artificial Intelligence (AI) models often act as opaque "black boxes" and impose heavy computational complexity and latency through sequential detection flows in their architecture, while also lacking an active Intrusion Prevention System (IPS).

This thesis proposes a lightweight, AI-driven parallel hybrid Intrusion Detection and Prevention System (IDPS) specifically designed for resource-constrained edge gateways, such as Raspberry Pi hardware. The architecture diverges from traditional cascade models by simultaneously deploying a supervised signature engine (composed of Random Forest, XGBoost, and LightGBM ML models) and an unsupervised anomaly engine (employing a DL based Autoencoder and ML based Isolation Forest and Local Outlier Factor models) to ensure rapid, low latency threat processing. To bridge the semantic gap of ML classifiers, the framework integrates a Large Language Model (LLM) in the form of Ministral to act as an Explainable AI (XAI) translator, generating human-readable threat narratives and allowing human-in-the-loop verification. Finally, the system automates mitigation by dynamically generating iptables firewall rules to drop malicious nodes based on AI confidence levels and human confirmation. Trained and evaluated on the CICIoT2023, the framework demonstrated high precision: the binary-optimized signature engine achieved a macro-average F1-score of 0.98, while the anomaly engine reached 0.96. Containerized deployment and real-time testing confirmed the system's ability to detect and actively block high-volume common attacks, such as TCP SYN and ICMP floods, directly at the network edge. Although several architectural challenges were identified during real-time testing, such as Layer 2 blindness and TCP self backstabbing, this research provides a robust proof-of-concept for democratizing intelligent, autonomous cybersecurity in IoT ecosystems.

Keywords: IoT, IDPS, Edge Computing, Edge Deployment, ML, DL, LLM, XAI, Hybrid detection, Zero-Day Threats, Active Mitigation, Human-in-the-Loop

Contents

List of Figures

List of Tables

List of acronyms

ACK Acknowledgment

AE Autoencoder

AI Artificial Intelligence

AIDS Anomaly-based IDS

API Application Programming Interface

ARLHIDS-IoT Accurate and Robust Lightweight Hybrid-Network Intrusion Detection System for IoT

ARP Address Resolution Protocol

CNN Convolutional Neural Network

DDoS Distributed Denial of Service

DDQN Double Deep Q-Network

DL Deep Learning

DoS Denial of Service

EoT Edge of Things

FIN Finish

FL Federated Learning

FN False Negative

FP False Positive

GAMO Generative Adversarial Minority Oversampling

GAN Generative Adversarial Network

GASS Gradient-based One-Side Sampling

GBDT gradient boosting decision tree

GDPR General Data Protection Regulation

GPU Graphics Processing Unit

GRU Gated Recurrent Unit

GUI Graphical User Interface

GUI Graphical User Interface

ICMP Internet Control Message Protocol

IDPS Intrusion Detection and Prevention System

IDS Intrusion Detection System

IF Isolation Forest

IIoT Industrial Internet of Things

IoT Internet of Things

IoV Internet of Vehicles

IP Internet Protocol

IPS Intrusion Prevention System

ISP Internet Service Provider

LIDAR Light Detection and Ranging

LightGBM Light Gradient Boosting Machine

LIME Local Interpretable Model-Agnostic Explanations

LLM Large Language Model

LOF Local Outlier Factor

LSEP Legal, Social, Ethical, and Professional

LSTM Long Short-Term Memory

MAC Media Access Control

MITM Man in the Middle

ML Machine Learning

MSE Mean Squared Error

OPF Optimum-Path Forest

OSI Open Systems Interconnection

PPS Packets per Second

QoS Quality of Service

RAM Random Access Memory

RFID Radio Frequency Identification

RF Random Forest

RNN Recurrent Neural Networks

RST Reset

S-BiGRU Stacked Bidirectional GRU

SHAP SHapely Additive exPlanations

SIDS signature-based IDS

SMOTE Synthetic Minority Oversampling Technique

SOC Security Operations Center

SQL Structured Querry Language

SVM Support Vector Machines

SYN Synchronize

TCP Transmission Control Protocol

TN True Negative

TP True Positive

TTL Time-To-Live

UDP User Datagram Protocol

UFW Uncomplicated Firewall

XAI Explainable Artificial Intelligence

XGBoost Extreme Gradient Boosting

XSS cross-site scripting

Usage of AI

AI has been used to identify and correct grammatical mistakes and expression issues.

AI, YouTube, and books have been used to learn about the principles, logic, and how the subjects of AI and machine learning work. These have also been used to learn about the details of how data assets are prepared for training, data classification, training variables, and how to generate performance evaluation tables and what each evaluation criterion measures.

AI has been used to assist in coding: fixing recommendations when bugs appear, providing implementation ideas for the final script, and identifying bugs when deploying the script to Docker. It also generated the prompts sent to XAI for detection analysis.

AI has been used for partial literature scraping, summary generation of those papers, and assistance in organizing the paper's structure.

The AI tools used are Google's Gemini and NotebookLM, Deepseek, and the built-in grammar checker model from Overleaf's Latex editor.

1 Introduction

Since its conceptual creation in the early 2000's, the Internet of Things (IoT) has evolved from sophisticated surveillance and Radio Frequency Identification (RFID) tags in stores to user friendly devices that can be easily set up in homes to transform them into "Smart Homes". Except for the introduction of such solutions into the consumer market, the IoT ecosystem has managed to enter industrial complexes through the Industrial Internet of Things (IIoT), healthcare, autonomous vehicles, and smart cities. This exponential growth can also be seen in the market growth graphs provided by Statista [1]. According to their Market Insights that start in 2018, the total IoT connected devices grew from 5.34 billion to an estimated 28.01 billion in 2026. This growing trend is projected to expand even further, with an estimated adoption number of 55.94 billion active and connected devices by 2031. The graph from Figure 1.1 shows that the consumer IoT market is the most dominant, projected to reach 28.14 billion connections by 2031, followed by IIoT, which is projected to reach 13.75 billion connections by 2031, as well as steady growth in the automotive and smart cities markets, with a strong start in healthcare and other industries.

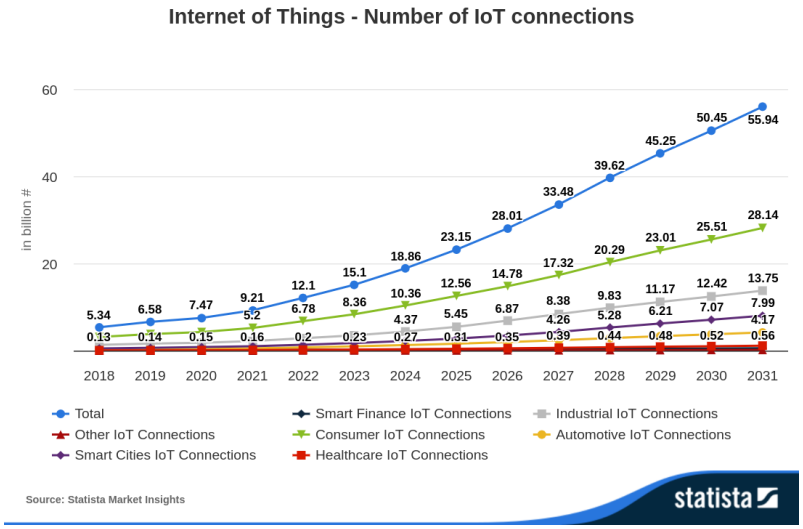


Figure 1.1: Number of active IoT devices from 2018 with projections for 2031 [1]

All this innovation, which has led to the exponential increase in adoption across multiple areas, has also resulted in an increase in security vulnerabilities due to the lack of interest from the manufacturers who prioritized rapid production with no time investment in security features. This way, the industry manages to invite more and more attackers to try their luck in sabotaging a home or an important infrastructure component. The most common attacks orchestrated by hackers revolve around Denial of Service (DoS) or DDoS attacks via compromised botnets, as well as data leaks and manipulation. Furthermore, the evolution of AI into a completely new industry has lowered the entry barrier for hackers in this space of tampering with the IoT infrastructure, thereby increasing the frequency, speed, and complexity of the attacks. To combat these attacks without degrading the Quality of Service (QoS), such as latency and performance, there is an urgent need to shift the security mechanisms away from centralized cloud architectures and deploy lightweight, intelligent, and trustworthy IDS directly at the network edge.

1.1 Problem Statement

While the AI revolution has created modern DL and ML models that offer high accuracy in detecting complex cyber threats in tests, their deployment in massive, rapidly scaling IoT infrastructures presents multiple bottlenecks. As good as the traditional DL models are at detecting threats, they often act as opaque "black boxes", hiding the reasoning behind their detection decisions. This lack of transparency limits the trust of traffic analysts and everyday users, creating skepticism about the practical implementation.

Current literature that explores this landscape frequently finds the mechanisms of the hybrid approach more reliable because they combine signature-based and anomaly-based detections, often deployed in a sequential or "cascade" flow. However, these approaches still struggle to balance accurate zero-day detection with resource efficiency. Furthermore, a significant gap has been found in the operational pipeline: most research focuses exclusively on threat detection (IDS) without suggesting active mitigation or intrusion prevention (IPS) methods. This gap presents an urgent need for a decentralized, edge-deployable framework that not only detects threats efficiently using a parallel hybrid flow but also translates its reasoning into a human-readable format for verification and ultimately transforms it into an active, dynamic firewall blocking mechanism.

1.2 Research Questions

To address the previously mentioned gaps, this thesis investigates and expands upon these research questions:

1. ***RQ1*** - What would be the best framework that can efficiently detect previously documented attacks and novel zero-day attacks while maintaining a low resource usage footprint?

According to the literature, the most efficient way to detect known and novel attacks is by using a hybrid machine learning approach that combines supervised signature detection with unsupervised anomaly detection. This allows for low resource usage when deployed on the edge.

2. ***RQ2*** - How can the detection be trusted without bothering the SOC analyst with a secondary, non necessary task of extra monitoring and debugging.
   The best method to avoid manual analysis and extra debugging is to include a translator for the reasoning behind the detection in the form of a notification or an alert dialog in case of low detection confidence that hints the Security Operations Center (SOC) analyst towards identifying whether the alert contains benign or malicious traffic.

3. ***RQ3*** - How can the framework be improved with active blocking to achieve a complete IDPS?
   The framework can be assisted with an IPS script that translates the detected anomaly into a firewall rule for **iptables** or **UFW** (Uncomplicated Firewall).

4. ***RQ4*** - What impact can such a framework have on the industry?
   The impacts include the democratization of the types of edge gateways used, a proper system that can act as a performant firewall for the IoT network, and an autonomous tool that assists the SOC analyst.

1.3 Research Objectives

The general objective of this thesis is to design, implement, and evaluate a novel, lightweight AI-driven firewall agent serving as an IoT edge gateway. To achieve this, the research will follow a list of specific objectives:

Primary objectives

1. **Develop a Parallel Hybrid IDS system**: implement a computationally inexpensive detection system that compares signature-based and anomaly-based detection results, diverging from traditional cascade architectures to respect the strict IoT QoS constraints.

2. **Implement XAI for Human-in-the-Loop Verification**: utilize one of the largely available LLM's as a translator between the "black box" IDS system and the human by explaining the features used to detect the malicious traffic in a human readable notification or alert dialog that integrates the human-in-the-loop principle

3. **Automate the IPS system**: Engineer a dynamic firewall rule generator that automatically executes **iptables** to quarantine malicious nodes based on detection accuracy or human response

Secondary objectives

1. **Test the agent in a containerized real-time environment**: The ultimate goal is to create an AI agent that is capable and resource efficient enough to be deployed on any edge gateway, including smaller ones such as the Raspberry Pi. This can be first tested in a virtualized environment where issues can be identified and corrected.

2. **Test the performance with other literature tests**: In order to prove that the framework is performant in tests, it must be compared with tests from other literature. This also proves whether the parallel approach is faster and more accurate compared to the cascade approaches.

1.4 Scope

The scope of this research is centered on network-level cyber-attack detection and mitigation at the Edge of Things (EoT). The proposed framework will be trained and evaluated using modern benchmark datasets that contain state of the art attack vectors, primarily the **CICIoT2023** dataset [2], followed by other references from the literature such as *UNSW-NB15* [3] and *BoT-IoT* [4].

Architecturally, the study plans to adopt an **Edge Deployment** strategy, where the agent is containerized and deployed on a resource-constrained virtual environment to prove that the concept can be scaled to real-world edge implementations. There is also a possibility of using a **Dual Cloud-Edge Deployment** strategy, where the intensive phases, such as dataset balancing, model training, and LLM fine-tuning, can be done on a GPU cluster, with the final touches being deployed on the virtual container. Other areas such as hardware security, cloud-side analytics, and cryptographic protocol analysis fall outside the scope of this thesis.

1.5 LSEP analysis

As the IoT rapidly integrates into critical infrastructure, smart cities, and domestic environments, the deployment of AI to secure these networks introduces complex challenges. Therefore, an evaluation of the proposed edge-deployable, human-in-the-loop firewall architecture through a Legal, Social, Ethical, and Professional (LSEP) lens is necessary to ensure that the system in not only technically viable but also responsible and compliant with modern societal standards.

1.5.1 Legal analysis

The legal landscape surrounding IoT data collection is increasingly stringent, being driven by global data protection regulations such as the General Data Protection

Regulation (GDPR) [5]. This thesis touches three main concerns over the GDPR:

- **Data Locality and Privacy-by-Design**: Traditional centralized cloud-based IDS requires continuous transmission of raw network telemetry to external servers, which violates data locality and increases the risk of data breaches. The proposed architecture mitigates this issue by operating at the network edge. By analyzing traffic locally on the gateway, the system enforces a strict "Privacy-by-Design" policy.
- **Minimizing Data Exposure**: This framework does not process any personal data. The only data generated by the LLM are human-readable metadata in the form of notifications or security alerts that await human interaction, drastically reducing the legal risks associated with unauthorized data exposure.
- **Micro-Segmentation and Blast Radius**: Legally, limiting the scope of a potential breach is necessary. By generating dynamic **iptables** rules at the edge gateway, the system isolates compromised nodes, limiting the "blast radius" of an attack and preventing movement into more critical network infrastructures.

1.5.2 Social analysis

In a scenario where the IoT infrastructure has no security, it could endanger everyday users and public infrastructure by exposing them to severe risks, including privacy invasions and integration into malicious botnets capable of launching massive DDoS attacks.

- **Democratizing Cybersecurity**: High-end, enterprise-grade security appliances are often expensive for standard smart homes or small-scale industrial deployments. Proving that an advanced, active-mitigation firewall can run efficiently on a forced low-resource containerized environment, demonstrates that

robust security can be achieved without massive expenses. This democratizes cybersecurity, making it accessible to average citizens and small municipalities.

- **Usability and Public Trust**: The largest barrier to IoT adoption is the lack of public trust due to frequent attacks. By integrating XAI into the IDS system, the framework bridges the gap between complex cybersecurity metrics and non-technical end-users, increasing societal trust in the IoT infrastructure.

- **Preventing Cascading Failures**: In critical domains such as smart cities and healthcare, malicious tampering can translate into public safety hazards. The detection and blockage of such threats locally prevents cascading failure across societal infrastructure.

1.5.3 Ethical analysis

The deployment of AI in security environments raises critical ethical dilemmas, particularly regarding autonomous decision-making and algorithmic transparency:

- **Overcoming the "Black Box" dilemma**: Deep learning models frequently act as opaque "black boxes", hiding the reasoning behind their classifications, thus lowering trust among the human factor. By implementing XAI into the framework, the reasoning is translated into a human-readable notification or alert that highlights exactly which traffic features drove the attack prediction, thereby making the AI's thought process transparent.

- **Accountability via Human-in-the-Loop**: Fully autonomous IPS risk making algorithmic mistakes, often referred to as false positives or false negatives. By including the human-in-the-loop logic in the framework, the AI can readjust itself based on feedback from the human supervisor.

1.5.4 Professional analysis

From a professional engineering perspective, deploying an IDS in an IoT environment imposes strict hardware limitations and QoS requirements:

- **Respecting Resource Constraints**: IoT edge gateways operate with severe hardware limitations that lower the processing power, available memory, and energy load, making the deployment of complex and computationally heavy models impractical for these environments. The proposed framework uses an approach that is inexpensive, thereby making low-end deployment possible
- **Solving the Cloud Bottleneck**: Cloud-based detection suffers from high detection latency and massive bandwidth consumption. By processing threats and generating the firewall rules at the gateway level, it clears up bandwidth and lowers response time.
- **Bridging Detection and Mitigation**: A significant gap in current cybersecurity research is that most frameworks stop at the IDS level without offering active IPS solutions. The proposed framework includes a mitigation wing that takes care of the IPS approach.

1.6 Organization

The rest of this thesis is structured as follows: Chapter 2 provides a literature review of current approaches, revealing their advantages and limitations. Chapter 3 details the methodology of the proposed framework, based on the literature review. Chapter 4 focuses on implementation, detailing all development stages with their own observations. Chapter 5 presents a performance analysis from two perspectives: the training performance and the deployment performance, both showcasing important findings for the next steps. Chapter 6 concludes the thesis with a summary of

contributions and proposes future steps for research.

2 Literature review

Due to the rapid evolution of IoT as an infrastructure and its adoption in various sectors, the threat landscape has expanded as exponentially as the evolution of the technology. This has triggered the creation of an extensive number of studies that explain the situation and propose various solutions for the infrastructure's security. This chapter presents a comprehensive overview of the existing literature that tackles this landscape, concentrating on the evolution, typical attack vectors, and their impact on the industry. Furthermore, the chapter explores which traditional detection techniques have been developed, such as signature-based and anomaly-based detection, explaining how they work and their limitations in the present day. The next section provides an overview of how machine learning approaches can impact the performance of the infrastructure based on computational complexity and memory management and explains how traditional ML approaches differ from deep learning solutions. Lastly, the chapter investigates the emerging role of LLMs and XAI in transitioning from passive monitoring to active, understandable intrusion prevention.

Furthermore, the chapter concludes with a section that highlights the current research gaps, outlining the lack of experimentation in hybrid detection systems, combined with the need for black box expansion and active mitigation.

2.1 IoT Attack Landscape

This section examines the rapid integration of IoT networks and the parallel evolution of cyber threats that target them. By analyzing the common attack vectors, the techniques employed by adversaries, and the resulting impact on the QoS, the vulnerabilities of constrained edge environments are established.

2.1.1 Growth of the IoT usage

The exponential growth of IoT as an industry has led to important achievements across different domains, including healthcare, smart cities, the Industrial Internet of Things (IIoT), and the Internet of Vehicles (IoV). As shown in Chapter 1, the number of connected IoT devices is exploding each day, with predictions indicating approximately 55.94 billion connected devices by 2031. Studies such as [6] estimate that the market will exceed $934 billion by 2033. To understand how these numbers exist, it is important to look back at the history of how this industry evolved.

The first practical implementation of IoT dates back to 1982 when the computer department of Carnegie Mellon University created the first coke machine connected to the "Internet". The motivation behind the coke machine was that the two developers were too far away from it, and the machine was always empty or had warm coke when they wanted to grab a can. This led to the implementation of microswitches and a clock in the machine so that they could monitor how full it was and how recently it had been filled. The machine was retired in 2014, but its story is still present on a website dedicated to it [7].

In 1990, John Romkey presented the first foundation of what a modern smart device means at the Infosec conference [8]. He created the first controlled "smart" toaster that communicated with a host via TCP/IP, with the basic functionality of turning the toaster on or off when the user needed it. The very next year, Quentin Stafford-Fraser and Paul Jardetzky invented the first webcam that was used to monitor the

coffee level in their machine because they found it was always empty when they needed it [9]. The webcam was connected to the internet in 1993, and everyone on the globe was able to report when the coffee jar was empty. According to Cambridge University's webpage, the webcam service was shut down in 2001 [10].

The current term "Internet of Things" was coined in 1999 by Kevin Ashton in a presentation for Procter & Gamble, where he indicated using RFID to manage the supply chain [11].

Entering the 2000's, the first commercially available IoT device was a smart fridge sold by LG [12]. This fridge allowed the user to track what items they had in the fridge, play TV shows and MP3 music, and recommend recipes based on the food inside the fridge. Unfortunately, the launch price was $20000, slowing down the adoption.

In 2002, the well known company iRobot launched the first Roomba self aware vacuum robot. According to iRobot, the technology used in the robot was the same LIDAR (Light Detection and Ranging)technology that was used to scan the pyramids of Giza and the remains of the twin towers after the 9/11 incident [13].

In 2005, the world welcomed the first piece of hardware that hobbyists call "DIY IoT". Created by Massimo Banzi, the Arduino project started as a platform to help students deploy their code on a BASIC Stamp microcontroller for the same price as the microcontroller itself [14]. The very next year, the second largest player in the embedded scene emerged: the first Raspberry Pi prototype was developed by Eben Upton, which at the time was based on the Atmega 644 microcontroller [15], making it closer in architecture to the Arduino compared to the microcomputer we know today.

2007 marked the release of the iPhone by Apple, changing forever how we access IoT devices directly from the phone. In 2010, Tony Fadell, the father of the iPod and co-creator of the iPhone, founded Nest and released the first smart thermostat

because he could not find a "decently working" thermostat for his home [16]. The last big inventions in the field of IoT were made by Philips in 2012 with the Hue Bulb [17] and in 2014 by Amazon with Alexa and Echo [18].

2.1.2 Evolution of IoT Threats and Common Attack Vectors

As the industry of IoT evolved exponentially, with more and more vulnerable devices being manufactured, attackers began to take an interest in this new area of exposure. Because the IoT is designed to serve as a smart tool, attackers have started tampering with the devices to exploit the victim's personal information without modifying the initial purpose of the device, leaving the victims unaware of the event.

In the initial attacks, the hackers tried to adopt a traditional vector that consisted of compromising a device or its local network in order to gain control. Such an approach was implemented in 2016 with the Mirai botnet [19]. In November, the botnet infected more than 600000 IoT devices using self-propagation, with most of its victims being internet routers without proper security and internet-connected cameras. The botnet helped conduct a massive DDoS attack that overwhelmed the network with requests it could not handle.

By 2018, hackers managed to change their methods, migrating from attacks on the networks and devices themselves to attacks over the wireless communication protocol for smart devices. They discovered that by hacking the Z-Wave protocol, it would be easier for them to expose all the devices on a network. Such a vulnerability was discovered in the spring, affecting over 100 million smart devices, with the potential to disable everything related to security devices such as cameras and smart locks, leaving houses open to thieves.

Another vulnerability was discovered in the Alexa Echo line of smart speakers. The exploit is simple, involving only a recorded voice sample from the owner. With that sample, the attackers can access everything offered by the speaker, often without

the victim even noticing. An example would be that while the victim is listening to music or podcasts, the attacker can use the sample to order products or services that drain the victim's bank account.
Furthermore, the recent evolution of AI has started to help new people learn and join the attack scene, thus increasing the frequency, speed, and complexity. With that in mind, we can easily say that most attacks carried out by hackers are DDoS attacks, which overwhelm networks with unnecessary traffic, Man-in-the-Middle (MITM) attacks, where the attacker does something completely different while fooling the victim, data breaches, in which the attacker exposes private information to the public or uses it, and zero-day exploits for new devices with unknown vulnerabilities.

2.1.3 Techniques Used by Attackers and Their Impact

When looking at the common attack vectors, we can also categorize the techniques that they use to achieve success in their tampering. These techniques target different layers of the IoT architecture, and according to the survey that discusses the effects of security measures on QoS [20], these layers are the perception layer, network layer, and application and software layer. At the perception layer, attackers use spoofing, signal jamming, and node capturing to gain unauthorized physical access or to tamper with the radio frequencies. At the network layer, attackers prefer to execute routing attacks such as sinkholes or black holes to drop or redirect data. At the application layer, attackers use common techniques such as SQL (Structured Query Language) injection, cross-site scripting (XSS), and malicious code injection to bypass authentication.

These cyberattacks extend far beyond simple jams or data theft. According to the previously mentioned summary [20], they degrade the QoS of the entire network by increasing latency, packet loss, and jitter. In healthcare, for example, ransomware or DDoS attacks can not only endanger patients' lives if they are directed towards

medical monitors and life support apparatus, but also threaten their privacy by exposing vital medical history data. In industrial and smart city environments, however, synchronized disruptions can lead to severe power grid failures, manufacturing supply chain halts, and tampering with transportation systems and traffic management systems, directly endangering public safety. On the private landscape, such attacks can endanger individual homes, making them vulnerable to burglaries and even threatening important digital infrastructure, such as bank accounts.

2.2 Traditional Detection Techniques and Tools

In order for a network to secure itself against the previously mentioned threats, IDS can serve as the primary defensive measure. This section reviews traditional approaches to IDS, explaining their operational techniques and exposing their limitations, which ultimately make them inadequate for modern, large-scale IoT deployments.

Signature-based Detection

The signature-based IDS identifies threats by comparing incoming network traffic against a predefined attack patterns database. These patterns are known as "signatures", and they can be modified by the user at any time. Tools that rely on this mechanism, such as Snort[21] and Suricata [22], are highly efficient in the resources they use, and they offer great basic protection for the network. The downside, however, is a fundamental flaw in their inability to detect novel or zero-day exploits, transforming them into an open gate for attackers who use new methods of exposing the network.

Anomaly-based Detection

Anomaly-based IDSs can identify threats by establishing a statistical or behavioral baseline of normal network traffic and flagging any significant deviation from it as a potential threat. This method represents a large defensive tool for the infrastructure against zero-day attacks, novel malware, and previously unseen vulnerabilities due to their analysis of network behavior instead of comparing simple signatures. Unfortunately, anomaly-based detection also has a significant flaw. It is unreliable for large-scale, unpredictable, and dynamic networks due to an exceptionally high false positive rate that could occur during firmware updates or device duty cycles, especially in smart city infrastructure.

Hybrid detection

Hybrid detection methods often attempt to combine the strengths of both signature-based and anomaly-based approaches in order to maximize coverage. The signature-based module ensures rapid responses to known attacks, while the anomaly-based module scans the network and blocks any zero-day attack attempts. Despite offering much greater accuracy in detections, traditional hybrid systems are highly inefficient in terms of resources due to their operation on a cascade flow, which increases computational complexity and time delays. The following paragraphs describe different hybrid approaches and how they attempt to solve the resource bottleneck.

ZeroDefense [23] is an adaptive hybrid fusion-based IDS designed for IoT networks that uses an anomaly-first cascade pipeline. It first screens all incoming network flows through three unsupervised anomaly detectors, specifically through Isolation Forest, Autoencoder (AE), and Local Outlier Factor, to catch potential zero-day threats. Only the traffic marked as safe is forwarded to the supervised layer, which uses multi-class classification with the help of Random Forest, XGBoost, LightGBM, and TabNet for precise attack labeling. Ultimately, the frame-

work proposes a confidence-aware fusion engine that integrates the outputs and uses a conservative fallback mechanism that flags traffic as an unknown threat if the classifiers disagree or output low confidence. This fallback mechanism exists because the outputs are not dynamically integrated and use only fixed weights, thus allowing space for possible false-positives.

ARLHNIDS-IoT [24] is a multi-tiered lightweight framework that reverses the cascade order by placing the signature-based IDS (SIDS) as the first line of defense. It uses an ensemble of seven tree-based models optimized via the OPTUNA hyperparameter optimization framework [25] to rapidly classify known threats using a majority voting classifier. If SIDS fails to recognize a match, the traffic is immediately forwarded to the Anomaly-based IDS (AIDS) layer that uses a reinforcement learning-based Double Deep Q-Network (DDQN) [26]. This AIDS inspects the network independently from the SIDS to identify potential unknown DDoS attacks, ensuring comprehensive coverage across both known and zero-day vulnerabilities. The issue discovered in the paper is that there is no human-in-the-loop to help the anomaly based system learn to detect a false-positive.

Edge AI Bridge [27] introduces a decentralized micro-layer hybrid architecture that sits between the IoT device and the main gateway. It implements a two-stage hybrid detection pipeline that first uses unsupervised machine learning for device-specific behavioral profiling and anomaly detection. To mitigate the high false-positive rates, the suspicious traffic is passed onto a lightweight signature-matching module. This module verifies the anomaly against known signatures before generating structured metadata, thereby stopping attacks at the physical entry point while preserving raw data privacy. The issue identified is that although suspicious traffic reaches the SIDS layer, it can still go undetected, thus leaving room for false-positives.

The **Multitiered Hybrid IDS (MTH-IDS)** [28] is a framework engineered for

securing both intra-vehicle and external vehicular networks in the space of IoV. The framework uses a parallel mechanism that applies tree-based supervised learning models to detect well known attacks. In order to have better coverage over zero-day attacks, it uses an anomaly-based mechanism utilizing k-means clustering [29], promising to achieve near-perfect detection rates across complex datasets. However, the system does not include a way to reduce the number of false-positives.

HDA-IDS [30] is a hybrid IDS tailored to protect IoT networks from DoS and botnet attacks. It uses SMOTE for oversampling and information gain to optimize feature selection, but it innovates heavily in the detection phase. It stacks multiple supervised tree-based classifiers (such as AdaBoost, XGBoost, and Random Forest) to identify known attacks while employing a semi-supervised CL-GAN (CNN-LSTM-GAN) model to flag unknown, emerging anomalies.

The **Reliable Event and Anomaly Detection Framework (READ-IoT)** [31] uses a hybrid cloud and fog deployment model. It uses a classic cascade mechanism with a SIDS first approach. The novel factor is that it uses reputation-aware provisioning to ensure that detection components are reliably executed across decentralized nodes, improving the overall accuracy and latency of such cloud IDS.

The **System for Preventing IoT Device Attacks on Home Wi-Fi Routers (SPIDAR)** [32] offers a cost-effective hybrid detection system aimed specifically at smart home networks. It is deployed on lightweight commercial hardware such as the Raspberry Pi, and it combines signature-based detection via the Snort IDS package with behavior-based detection that analyzes traffic patterns using predefined thresholds and machine learning models. This approach protects the home network from common threats such as brute force attacks, DDoS, and SQL injections without the need for a complex enterprise-level server.

HID-Smart [33] uses an anomaly-first hybrid cascade approach for home appliances. The anomaly-based layer uses the Random Forest algorithm to extract

and evaluate the general network behavior. After that, the traffic is passed to a signature-based detection that performs based on user-specific features, allowing the framework to work specifically on user network behavior.

Bostani and Sheikhan's Hybrid IDS [34] integrates anomaly-based and specification-based techniques to secure 6LoWPAN IoT networks against sophisticated routing attacks such as sinkholes [35]. This architecture uses the unsupervised Optimum-Path Forest (OPF) algorithm for clustering and MapReduce for scalable, distributed anomaly detection. The framework ensures low communication overhead while maintaining high detection precision by deploying it directly on the router.

2.3 Machine Learning in IoT Threat Detection

To overcome the rigidity and high false-positive rates of traditional systems, researchers have turned their attention to the emerging technology, AI. This section covers traditional Machine Learning and advanced Deep Learning techniques, and explores how these algorithms enhance accuracy, manage high-dimensional data, and adapt to zero-day vulnerabilities.

2.3.1 Traditional Machine Learning Approaches

Traditional ML approaches are widely used to enhance detection and introduce new, bare-bones ideas. As seen in the previously explained approaches, the most prominent traditional ML methods used are supervised and unsupervised learning. The first is used to enhance signature-based detection by learning how the threat is marketed, while the second is used to detect any anomalies in network behavior due to the lack of threats, thereby helping in detecting new and novel attacks. The most common supervised algorithms used in the cited articles are Random Forest (RF), Extreme Gradient Boosting (XGBoost), LightGBM, and Support Vector Machines

(SVM). These algorithms are highly performant in categorizing known IoT attacks with extreme precision. For detecting zero-day threats however, the most effective algorithms are the unsupervised ones, such as Isolation Forest (IF), Local Outlier Factor (LOF), and even K-Means clustering that can identify malicious traffic using reconstruction errors. Although these algorithms are extremely effective in detecting anomalies in the traffic, they are extremely dependent on the training, due to severe class imbalance in IoT datasets, where benign traffic is outnumbering the rare attack samples. In this regard, data augmentation techniques, such as Synthetic Minority Oversampling Technique (SMOTE) or Generative Adversarial Minority Oversampling (GAMO), must be strictly applied to training partitions in order to prevent bias.

2.3.2 Deep Learning Approaches

Deep Learning (DL) uses multi-layered neural networks, such as Deep Autoencoders, Convolutional Neural Networks (CNNs), and Recurrent Neural Networks (RNNs), including LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) , to identify spatial and temporal patterns across network flows. For example, Stacked Bidirectional GRU (S-BiGRU) is an architecture equipped with attention mechanisms that effectively models bidirectional temporal dependencies in network flows. However, such approaches are heavy on hardware resources and user privacy due to their nature of processing data as well. To address this, Federated Learning (FL) frameworks, such as Choir IDS [36] and Flex IDS [37] have been proposed. FL enables edge devices to collaboratively train global intrusion models by exchanging only the encrypted model weights rather than the raw, sensitive telemetry, thereby preserving data integrity and privacy. Because DL models, just like ML models, operate in an opaque "black box", XAI tools like SHAP (SHapely Additive exPlanations) and LIME (Local Interpretable Model-Agnostic Explanations) are integrated

to provide global feature importance and quick local approximations for human verification.

2.4 LLMs in IoT Threat Detection

While Machine Learning has drastically improved threat detection, it often operates as an opaque "black box" that does not provide any insights into its thinking process or any autonomous mitigation. This section explores a novel approach in IoT security: the integration of LLMs and XAI to democratize threat intelligence and facilitate human-in-the-loop active defense. To realize this integration, recent literature has proposed several architectural paradigms that transition LLMs from passive analytical tools to active components for network defense. These approaches address the limitations of traditional IDSs and establish a foundation for intelligent, LLM-integrated firewalls that are capable of operating dynamically at the network edge.

2.4.1 Bridging the Semantic Gap with XAI

A critical challenge in traditional anomaly-based detection is the complete disconnection between statistical deviations and actionable context. To resolve this and democratize threat intelligence, hybrid frameworks stack generative models on top of conventional machine learning classifiers, as explained in Mahmood M. [38]. When a model triggers an alert, the LLM ingests the statistical summary of the network flow and generates a human-readable narrative of the event. This semantic translation acts as an XAI layer that accelerates human-in-the-loop integration and reduces alert fatigue.

2.4.2 Autonomous Agents and Active Defense

Beyond explainability, cutting-edge implementations employ LLMs as autonomous reasoning engines that are capable of making decisions. Frameworks in this domain, such as the ones proposed in Saheed Y.[39] and Chou C.[40] papers, utilize an action space that allows the model to extract data, query knowledge bases, and execute mitigation scripts. This allows them to dynamically adapt to zero-day threats, while evaluating them against the current dataset benchmarks, such as the previously mentioned CICIoT2023 [2]demonstrates their robustness in identifying complex attack vectors that include both common attacks, such as SYN floods and DoS attempts, and rare silent attacks.

2.5 Research gaps

Drawing upon the review of the current methodologies, this section aims to synthesize the critical shortcomings identified in contemporary IoT security literature. The main highlight is the unfulfilled need for lightweight, parallel hybrid detection models that not only explain their reasoning but also actively execute mitigation rules at the network edge. Despite the significant advancements in IoT cybersecurity, several critical research gaps remain unexplored. Firstly, the vast majority of the current literature focuses exclusively on passive threat detection (IDS) and stops short of active intrusion prevention (IPS) or automated dynamic mitigation. Secondly, most hybrid detection frameworks employ a sequential or cascade flow detection, with a signature or anomaly engine on top and the other waiting for a response, creating unnecessary latency bottlenecks, which makes a parallel flow architecture more plausible. Thirdly, while XAI provides interpretability and makes the training results more understandable to humans, in the case of SHAP, there is a real need for better integration that can act as a translator rather than a theoretical

value calculator. Lastly, because LLMs have started to become a better representation of the AI space, it is necessary to integrate them as a possible translator for the "black boxed" ML or DL detection engines, fully decentralizing the architecture and embracing a human-in-the-loop approach, transforming the framework from a time consuming tool into a useful assistant for the SOC analyst.

3 Methodology

The purpose of this chapter is to outline the complete methodology for the development of the IoT IDPS system based on machine learning and AI technologies. The approach starts with a general view of the architecture, followed by an explanation of the purpose and functionality of each component, finishing with an overview of all the technologies used to create the components of the architecture

3.1 Proposed System Overview

In order to develop a reliable, functional, and competitive architecture, the methodology depends on the literature review to gather enough sources that explain, argue, or motivate the choices made in the methodology chapter. In order to construct the architecture discussed in the following paragraphs and sections, a list of similar approaches has been gathered and explained in the previous chapter. Among those approaches, the most appealing candidate for inspiration had been the **ZeroDefence** paper [23]. The assets used from the paper to construct the architecture revolve around machine learning algorithms and how they are handled, and the borrowed assets will be justified once the discussion reaches that specific asset.

The architecture in mind can be seen in the Figure 3.1 and starts with the traffic collection and transformation. Because the detection component does not process raw data, it is important to transform it to the necessary data type for the detection component. The traffic is captured in a 5 second time frame and transformed into

the data type used by the detection component.

The detection component follows a parallel analysis approach where, compared to traditional waterfall approaches, it performs both signature and anomaly detection at the same time, thus, in theory, reducing the analysis time by half. To prove if this is the case, further tests will be conducted and compared to a classic waterfall architecture, namely the ZeroDay detection system itself. The detection will be done using machine learning algorithms:

1. Signature detection:
    - Random Forest
    - XGBoost
    - LightGBM
2. Anomaly detection:
    - Autoencoder
    - Isolation Forest
    - Local Outlier Factor

The results provided by the machine learning algorithms will be compared and sent to an LLM agent that will process and interpret the results to help the human analyst take action. This way, the architecture ensures a human in-the-loop approach that helps bridge the ability of human interpretation and the black box approach of the machine learning algorithms, thus increasing trust in the automated system. The LLM is responsible for notifying the human about risks identified by the system with high confidence and also alerting if the confidence percentage is low. This way, the human is both informed about the status of the identified attacks and implicated in the decision making process by analyzing and marking the finding as either clean or a threat, with the help of LLM.

In the case where an alert is triggered by the detection system and human intervention is needed, the system features an auto-adjust mechanism that is responsible for changing the decision making power of the algorithms and the detection components themselves.

Once the threat has been detected in either of the 2 cases: high confidence by the detection system or marked as a threat by the human analyst, the architecture proceeds to create firewall rules that block the suspect address on the identified ports and protocols. The firewall rules are fed to **iptables**.

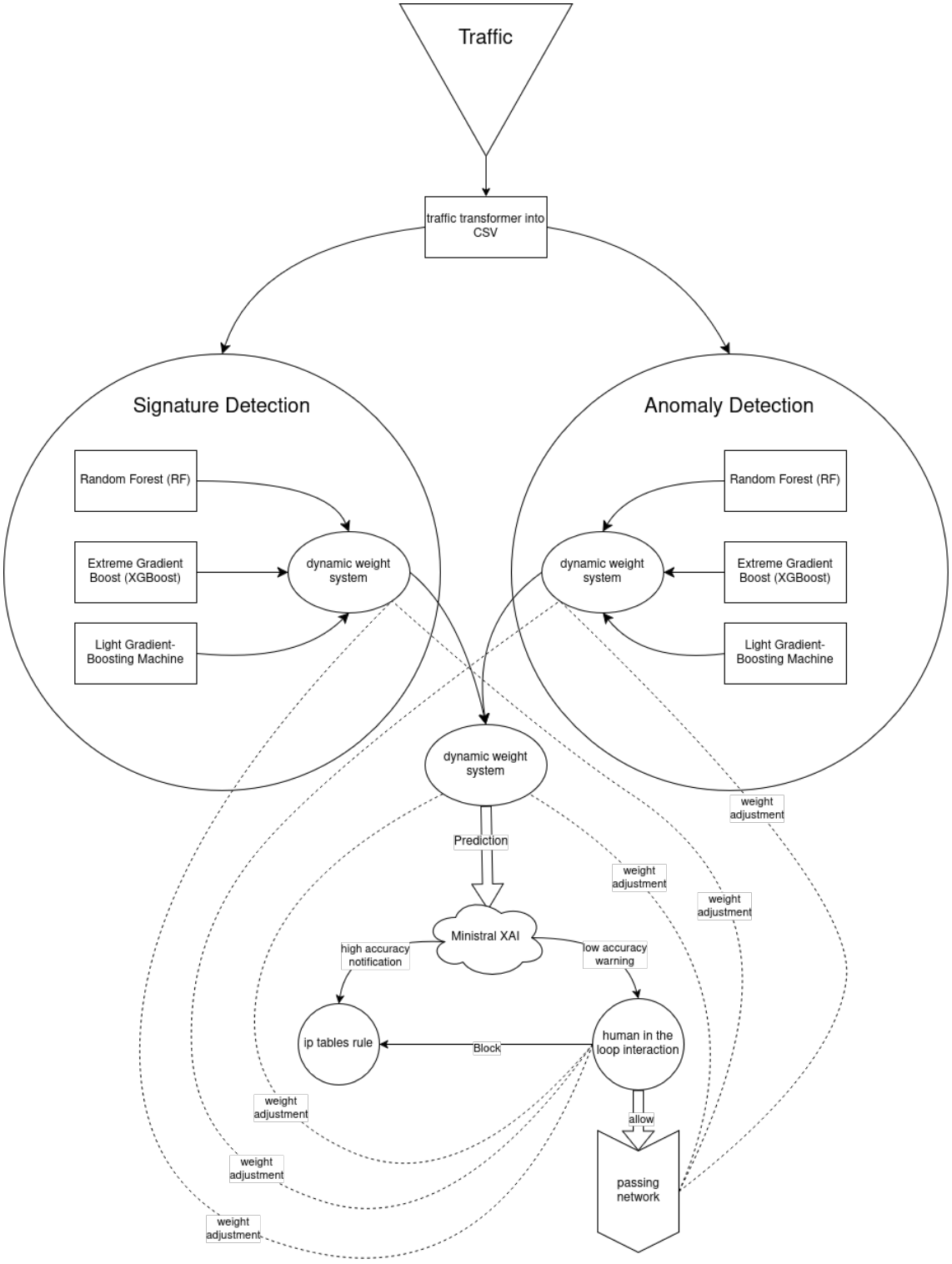


Figure 3.1: Complete architecture and flow of the network traffic through the IDPS system

3.2 Processing the Traffic

Because the IDS component of the architecture cannot process raw packet captures that do not show any behavioral logic, only headers and raw data, it is important to process them and prepare them for the IDS engine. This way, the architecture includes a "feature extraction engine" that works by collecting traffic in a set time-window and transforms it from a microscopic detailed and raw view into a macroscopic "flow" representation that shows the behavior of the network during that timeframe.

The transformer uses the dpkt python library, which is a low level parsing framework that can unpack the binary data behind the traffic. The transformer starts looking at the data link layer (OSI layer 2) from the internet interface you choose and decodes it to be able to access the network layer payload (OSI layer 3). The next step is to implement a filter that isolates IPv4 traffic and discards any ARP (Address Resolution Protocol) broadcasts or IPv6 packets, ensuring that the models evaluate only the protocols they were trained on. From the IPv4 traffic, the engine extracts header telemetry, including Header Length, Time-To-Live (TTL), and total payload size.

The next step is to inspect the transport layer (OSI layer 4) to identify the protocol and connection state that can help detect specific attack signatures, such as DDoS floods or port scans. The engine classifies TCP, UDP, or ICMP packets using a numerical protocol ID assignment (in the previous case, the IDs would be 6, 17, or 1). Furthermore, if the identified protocol is TCP, the engine performs a flag tracking operation that can count the exact number of SYN, FIN, RST, and ACK packets within the analyzed window and can help the models identify anomalous handshakes or teardown sequences typical of malicious scanning or flooding.

Once the packet extraction is done, the engine moves to derivation by calculating the statistical distribution of the traffic over the entire window. The engine calculates

the Minimum, Maximum, Average, Standard Deviation, and Variance of the packet sizes, where a high variance might indicate advanced web browsing, and a variance of 0.0 can often indicate an automated flood of identical packets. The engine also calculates the fundamental velocity of the traffic window, known as the Transmission Rate, using the following formula:

$$Rate = \frac{N_{packets}}{t_{end} - t_{start}}$$

where $N_{packets}$ is the numerator and represents the total number of packets that were captured during the specific observation window, t_{end} is the timestamp of the very last arrived packet inside the current buffer, and t_{start} is the timestamp of the very first arrived packet inside the current buffer. The subtraction of the timestamps is known as the Denominator, and it represents the absolute duration of the analyzed traffic flow. The result is the Transmission rate in question, and it is measured in *Packets per Second* (pps).

On top of that, the engine also maintains a frequency dictionary that saves the IP addresses that have been identified in the time frame and the volume of traffic. After the feature extraction, the dictionary is reordered with the top IP being the address with the most traffic. If the IDS orchestrator marks the transformed vector as a high-confidence attack, the top IP is used to generate the firewall rule for automated blocking.

3.3 The IDS engine

The IDS engine is responsible for analyzing the transformed raw traffic and detecting whether the network behavior is a known attack or a new one. To accomplish this and require as little human interaction as possible, multiple machine learning algorithms have been used. To shorten the algorithm research hunt, the literature

review offered many insights regarding the most commonly used algorithms, and among the similarly explained approaches, the ZeroDefence [23] paper provided the most comprehensive layout of the algorithms used. Because the IDS is responsible for finding both known and new attacks, the engine is divided into 2 separate components: **Signature Detection** and **Anomaly Detection**. According to the literature review and the ZeroDefence paper, the use of both components is necessary because using each component separately can lead to problems in the detection process, often referred to as false-negatives. The signature detection can find known threats but is vulnerable to zero-day attacks, while anomaly detection can easily identify new, undocumented attacks; however, it can impose delays in detection for known attacks. This hybrid approach ensures that known attacks are rapidly detected by signature detection while keeping the system secure from zero-day attacks, thanks to anomaly detection.

3.3.1 Signature Detection

The signature detection component is responsible for finding known, documented attacks in the IoT space. The machine learning approach for this component is to use supervised learning, which is a type of learning where labels are applied to the training data, helping the algorithms understand which traffic is an attack or benign. This way, the algorithms only learn how to analyze the network behavior that ultimately leads to the final labeling of an attack (1) or benign (0). The training data used for this component is composed of both attack and benign traffic due to the algorithm's need to recognize the exact mathematical patterns of known threats. The ZeroDefense framework pointed out that the fastest and most reliable algorithms are **XGBoost**, **Random Forest**, and **LightGBM**.

Random Forest

The Random Forest algorithm is a tree learning method based on Bagging (Bootstrap Aggregation [41]) that prevents overfitting. The logic behind it lies in building hundreds of smaller and independent decision trees that avoid one large and often unreliable decision tree. Each small tree is trained on random subsets of the training data, and each split in the tree is made by looking at a random subset of the dataset features (columns). When a new traffic window arrives, it is passed through all the generated trees, with each of them returning a result or "vote" (attack - 1 or benign - 0), and the RF outputs the majority vote. This way, RF is stable and resistant enough to noisy network data.

XGBoost

XGBoost is a sequential learning method based on Boosting [42] that is known for aggressive error correction. Unlike RF, where the trees are independent, XGBoost builds trees in a sequence:

- **Tree 1** makes the first prediction on the network traffic. It will inevitably contain false positives or false negatives, and these are known as residual errors.
- **Tree 2** is built to predict and correct errors identified in the first tree.
- **Tree 3** predicts and corrects the errors from tree 2 and so on.

To optimize the loss and add new trees, the Gradient Descent algorithm is used [43]. In the end, because the algorithm explicitly hunts for its own mistakes during training, XGBoost is mathematically aggressive and generally provides high accuracy in detecting known DDoS or scanning attacks.

LightGBM

The LightGBM algorithm is a highly optimized boosting framework that is designed for extreme computational speed and low memory usage. At its base, LightGBM operates on the same boosting principle as XGBoost, but with 2 important changes that are crucial for IoT edge deployment. The algorithm uses histogram-based binning, where instead of scanning every single feature to find the best way of splitting a tree, it bundles continuous features into discrete "bins" that help reduce CPU usage.

3.3.2 Anomaly Detection

The anomaly detection engine is responsible for finding new, undocumented threats known as zero-day attacks. The machine learning approach for this component is to use unsupervised learning, which is a type of learning where the algorithm is feeded unlabeled data and it must figure out by itself whether the fed traffic is benign (0) or attack (1). The training data used for this component is composed of strictly benign traffic due to the algorithm's need to detect the attacks as data that they cannot work with, meaning that the anomaly component acts as a watcher over the network and marks any anomaly from the normal flow as a potential threat. The ZeroDefense framework pointed out that the fastest and most reliable algorithms are **Autoencoder**, **Isolation Forest**, and **Local Outlier Factor**.

Autoencoder

The algorithm is a deep neural network designed to compress and reconstruct data in order to measure deviation. It is mostly used to recognize images or certain definitive features of an image, such as numbers. At its core, it consists of an Encoder, a Bottleneck (the compressed data space), and a Decoder. The training logic works by feeding it the known number of dataset features (columns), which the

Encoder compresses into a smaller bottleneck (from 39 features to 8). The Decoder then tries to expand the bottleneck back to the original state, helping the algorithm learn the mathematical correlations of the features to minimize the "Reconstruction Error". Because the autoencoder is trained only with benign traffic, every attack attempt is completely alien to the AE, and while it tries to compress and reconstruct the traffic, it fails miserably, resulting in a high Reconstruction Error that flags the traffic as an anomaly.

Isolation Forest

The algorithm detects anomalous traffic using its core principle that outliers are "few and different" and can be easier to isolate. Instead of trying to understand what normal traffic looks like, IF actively tries to isolate data points. Its job is to select one random feature from the dataset and a random split value between the minimum and maximum of that feature. It then builds a tree by repeating the random splitting until all data points are completely isolated on their own leaf. The critical point is that benign traffic is densely packed together, requiring a large number of splits to isolate a single point, while a threat is mathematically distant from everything else, meaning that its isolation in the tree is done relatively quickly. Therefore, the shorter the path length, the higher the anomaly score.

Local Outlier Factor

The algorithm is a density-based anomaly detector that measures local deviation using the nearest neighbors. Compared to IF, which looks at the whole dataset, LOF looks at "neighborhoods". It calculates the mathematical distance between a specific 5-second traffic sample and its k nearest neighbors in the database's dimensional feature space. It computes a "Local Reachability Density" of the point compared to its neighbors, then judges the results. If a data point is located in a dense cluster, its

LOF score is -1, and it is marked as normal traffic. If a data point has a significantly lower density than its immediate neighbors, its LOF score spikes, making LOF a very efficient hunter of stealthy, low-and-slow attacks that, at first sight, blend in globally but are highly suspicious locally.

3.3.3 Training Datasets

For the models to be trained to efficiently detect any threats to the IoT network, it is necessary to introduce at least one dataset that contains enough information about both benign and infected traffic. This way, the dataset can be used for both supervised and unsupervised learning. The proposed system uses, thanks to the literature review, one of the most commonly mentioned datasets that is not only reliable but also has different attack vectors that diversify the training scheme: the **CICIoT2023** dataset [2]. This dataset contains 33 different attack vectors tested on a topology of over 100 IoT devices. The dataset contains 39 features that describe the addresses of the attacker, the number of packets sent, the total TTL, the ports used, the protocols used, etc., and 1 label column that describes whether the row is an attack or benign traffic. For the proposed system, all features are used, and the label column is used for supervised learning and in unsupervised learning for verification.

3.3.4 Merging the results

The final step in obtaining a short decision of 0 (benign traffic) or 1 (malicious traffic) is to merge all the results of the algorithms into one. Having all six algorithms to detect anomalies is useless if they disagree on which one is right and which one is not. The first logical method of obtaining a single decision is to implement a voting system, where the majority wins the decision of the detection category. This approach can be a good performer at first sight but often different machine learning

algorithms can produce different false positives or false negatives results, making some of them more accurate than others. To regulate the results it is necessary to implement an **Orchestrator** or a **Decision Fusion Engine**. This Orchestrator is responsible for offering more power to the algorithm or engine that has much more precise results, while keeping the result between 0 and 1. The ZeroDefense framework uses the orchestrator to combine the results for the algorithms for each engine, due to its structural architecture of being a waterfall based detection. The proposed architecture uses the orchestrator not only to fuse each engine algorithm's result, but it is also used to control the decision power of both engines, due to its parallel design. This way, the Decision Fusion Engine is split into two tiers: the Intra-Engine fusion, and the Inter-Engine fusion.

Tier 1: Intra-Engine Fusion

It represents the fusion that takes place inside each engine that gives the final combined response for their each algorithm. The fusion uses, according to the ZeroDefense paper [23], a Weighted Soft Voting mechanism based on empirical performance that levels each algorithm's decision power. These weights are a direct representation of the training performance of each classifier and are directly influencing how much power each model has over the engine decision, where the models with lower training results have a smaller weight, while the most perfomant ones are pushed as the main decision maker of their specific engine.

1. **Signature Engine Fusion**

   The supervised models are not able to output a straightforward binary 0 or 1 answer for the data they are analyzing; instead, they do so by using a *prediction probability*. The engine extracts the probability that the traffic belongs to a

threat and then applies the base weights:

$$P_{signature} = (0.39 * P_{XGB}) + (0.33 * P_{RF}) + (0.28 * P_{LGB})$$

As seen in the equation, the authors detected that the LOF model is the worst performing one compared to XGBoost which is marked as the dominant power of the engine. This way, the engine preserves its relevance in the general decision making, due to the power of each model to pull the fused confidence level up compared to a traditional voting system.

2. **Anomaly Engine Fusion** The unsupervised models use completely different mathematical concepts that cannot be averaged as easily as the signature results. The AE outputs a *Reconstruction Error*, the IF outputs an *Anomaly Score* based on tree path length, and the LOF outputs a *Density Ratio*. To unify these different scales, the system uses a *MinMaxScaler* [44] that aims to normalize the outputs of each model within a defined range, in our case between [0, 1], using the formula:

$$S'_{model} = \frac{S_{model} - min(S)}{max(S) - min(S)}$$

where S is the raw model output and S' is the normalized value. To ensure that all normalized values are interpreted as 0 (benign) and 1 (threat), the values from IF and LOF must be inverted (multiplied by -1) and then normalized. The final fused calculation is done using the formula:

$$Score_{Anomaly} = (0.3 * S'_{IF}) + (0.3 * S'_{AE}) + (0.4 * S'_{LOF})$$

Here, the equation illustrates that the anomaly engine is more leveled compared to the signature engine, with LOF as the overall dominator. Further-

more, compared to the signature engine, the anomaly engine outputs a distance or error score, meaning that the output does not represent a percentage of how much of a threat a traffic window is. For that, a threshold must be used so that any error score greater than it is considered a threat, while those below it are considered benign. The implemented threshold is named θ and it represented by the 99th percentile of the benign normalized score. This way the engine thinks that in a live environment, everything that scores higher than the threshold is considered alien and must be a threat to the network. At the end, the engine outputs a binary value called $is\_anomaly$ with the value 0 for benign and 1 for threat.

Tier 2: Inter-Engine Fusion

This fusion is known as the Orchestrator, the one that decides whether the traffic is benign or malicious based on each engine's results. The way it works is similar to the other fusion equations:

$$Score_{Hybrid} = (0.5 * P_{Signature}) + (0.5 * is\_anomaly)$$

This way, if any known threat gets detected by the signature engine, the score will be ≥ 0.5, triggering an alert regardless of the anomaly engine's result. If a zero-dat attack occurs, the score gets to 0.5 due to the anomaly engine. If the traffic is benign, both engines output 0 and the score is 0. To make the engines more reliable a confidence value is used to confirm if the traffic is a threat or not and when to call the human analyst. If the confidence is $\geq 80\%$, the engines are in a strong agreement and the IPS is called autonomously. If the confidence is lower, the SOC analyst is called for checking and confirming whether the alert is correct or not. The presented equation represents just the start of the analysis and it is subject to further adjustments in the future, based on testing results.

3.4 Explainable AI and Human-in-the-Loop

To make the IDS engine more compliant for edge deployment and real life situations, it is important to help the SOC analyst understand the reasoning behind the black boxed models. For that, the architecture introduces the concept of *XAI* with the sole purpose of processing results from the IDS engine and formulating them in the form of either a notification or an alert that requires SOC interaction. The implemented explainer consists of an LLM that takes a prompt containing the most important features from the dataset and notifies the human if the engine decides to block the threat, explaining why, or sends an alert message to the SOC analyst for him to verify the possible threat and respond whether to allow or block it. The LLM used for this architecture is Ministral with 3 billion parameters from Mistral. The LLM is triggered only if the traffic window is identified as a threat, and based on the confidence level of the orchestrator, it outputs the notification at $\geq 80\%$ confidence and the alert if the confidence is lower.

3.5 The IPS engine

The IPS engine is intended to be used as a defense mechanism for the gateway against threats after they have been discovered. Since most of the gateways are running on Linux due to hardware limitation preservation, the IPS engine is designed around the default Linux firewall service, **iptables**. Once the threat has been identified by the IDS engine, the system creates a DROP rule for **iptables** with the IP address found in the feature extraction engine and the ports from which the issue has been identified. In theory, this should prevent any other attacks coming from those addresses and transport protocols.

3.6 Used Tools

To facilitate the creation and testing environments for the above discussed system, a set of tools are necessary to be used. For the training phase o the models and basic usage of the dataset, the university's Jupyter Lab was used with Jupyter Notebooks written in Python. The final system assembly was done locally on the machine using Visual Studio Code and Python as programming language. The testing is done in a virtualized environment, where the gateway with the final script hosted on a container and the attacker in another virtual machine using Parrot OS. The virtualization software used for testing is Docker.

4 Implementation

This chapter shows how the system explained in the previous chapter is implemented using the mentioned tools. The main goal is to implement an IoT IDPS system that automatically detects possible IoT threats using machine learning and blocks them, with the help of explainable AI and a human in the loop to ensure that the system is not working in a "black box" without the possibility of understanding and verifying whether the models are accurate. The implementation consists of backend implementation for the final script, as well as training logic and XAI setup.

4.1 Dataset Preparation

Before starting the model training, it is important to download and prepare the data from the dataset. The dataset was found on Kaggle under the "Large-Scale Attacks in IoT Environment" repository [45], containing the data organized in different directories based on attack vectors and benign traffic. To install it in the Jupyter Lab environment, Kaggle offers a library and API (Application Programming Interface) call that downloads the dataset directly without needing any other logins. After that, the dataset directory is searched within the system files to load and extract data. After the dataset has been loaded, a chunk of RAM-safe (Random Access Memory) data must be extracted from it, with the aim of having the same volume of threat and benign traffic. In this case, 1 million rows have been targeted for extraction, with half of them being redirected for benign traffic and the

other half towards attack traffic. After that, the chunks must be cleaned of rows with empty or N/A features and balanced so that each traffic category has the same number of rows. To avoid row generation, the category with the largest number of rows is trimmed to match the number of the other category. Next, X and Y are extracted for use as training/testing and verification, respectively. Finally, X and Y are split into training and testing samples, and X samples are scaled using the MinMax scaler. The final result, which contains the amount of extracted samples and their distribution in each category, can be seen in Figure 4.1.

```
--- 1. Downloading/Locating Dataset ---
--- 2. Loading RAM-Safe Data Chunk ---
--- 3. Cleaning & Balancing Data (Undersampling) ---
Row counts before balancing:
label
0    499970
1    499490
Name: count, dtype: int64

Balancing dataset to exactly 499490 rows per class...
Final balanced distribution:
label
0    499490
1    499490
Name: count, dtype: int64

--- 4. Extracting X and y ---
Data extraction complete! Total rows ready for split: 998980
--- 5. Splitting Data ---
--- 6. Scaling Features (Crucial for AE & LOF) ---

Success! Ready for Anomaly & Signature Detection.
X_train_scaled shape: (799184, 39)
```

Figure 4.1: Data prepared for training the models

4.2 Training Logic

The training phase of the models was also conducted in Jupyter Lab. The bench was prepared in such a way that all algorithms were trained in their specific engines, so testing the performance was done not only on each model individually but also at the engine level.

4.2.1 Signature training

The signature engine utilizes supervised learning models and is designed to recognize specific mathematical patterns of known attacks. The training is done on a balanced dataset containing both benign and malicious traffic. The engine uses three different tree-based algorithms that are configured for accuracy and processing efficiency:

- **Random forest**: is initialized with 100 estimators, uses balanced class weights to prevent majority-class bias, and uses all available CPU cores to shorten the training time window
- **XGBoost**: is configured to use a learning rate of 0.1 and "logloss" as its evaluation metric for gradient optimization
- **LightGBM**: uses a gradient boosting decision tree (gbdt) architecture combined with Gradient-based One-Side Sampling (gass) to increase computational speed

The evaluation phase does not use any binary prediction that classifies the traffic as benign or malicious; rather, it uses the raw output of each algorithm, which represents the probability of how likely the traffic window belongs to the malicious class. Once the evaluation phase is done for each model individually, the code aggregates the values through the fusion system and performs an evaluation on the whole engine.

4.2.2 Anomaly training

Unlike the supervised engine, the anomaly engine is designed to detect zero-day threats to the network by analyzing the network behavior rather than identifying the mathematical connections to a possible malicious window. It uses unsupervised learning based algorithms and the training data concentrates on benign traffic only, with the main target of treating the malicious traffic as errors in training.

- **Autoencoder**: is the only deep learning algorithm used, and it measures data reconstruction loss. The model receives an input dimension of 39 features from the dataset and attempts to shrink or encode those features into a single hidden space of 16 neurons using the 'sigmoid' activation function, then tries to decode the output back to the original dimensions using the same 'sigmoid' activation function. The model is compiled using the Adam optimizer with a learning rate set to 0.00001 and Mean Squared Error (MSE) as the loss function. The training is done over 100 epochs, with a batch size of 128 and a validation split of 0.0001. The anomaly score is calculated using the Euclidean norm between the original packet features and the reconstructed output.

- **Isolation Forest and Local Outlier Factor**: are two statistical models that complement the autoencoder and are trained on the benign training set. The IF is initialized with 100 estimators, a sample size of 256, and a contamination rate set to 0.05. The LOF is evaluating the local density using 20 neighbors and the Euclidean distance metric. Both results returned by the models must be inverted by -1 so that the larger the score, the more likely the traffic is malicious.

Once the training is done, each model has its θ value calculated using the 99th percentile of the benign identified traffic, and an individual evaluation is made. After that, the scores are normalized using the Min Max Scaler and then grouped together using the fusion engine, with the code ending in an engine performance evaluation.

4.2.3 Performance evaluation items and adjustments

Once the training is done, one way to test whether the models are predicting well is to conduct a performance evaluation that focuses on using a standard multi-metric Classification Report. This report uses Precision, Recall, and the F1-Score as the

main foundational metrics, which are derived from the confusion matrix generated during the testing phase. The matrix in question categorizes predictions into True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives(FN) that are used to calculate the main metrics.

1. **Precision**: is known as the exactness of the model, illustrating what percentage of actual traffic belongs to that category of traffic (0 - benign; 1 - malicious). It is important in IDS systems because low precision means that the model can mark legitimate user traffic as malicious for no reason. The precision is calculated using this formula:

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

2. **Recall**: is known as the completeness of the model, illustrating what percentage of the actual attacks the model identified. For the architecture, it is more essential than precision because it shows the number of false negative attacks that went unidentified by the IDS system, thus indicating how vulnerable the network can be. To calculate it, recall uses this equation:

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

3. **F1-Score**: is the harmonic mean between precision and recall, providing a metric that balances the tradeoff between the two. This metric shows how well precision and recall are performing, and a low F1-score indicates a low metric that drags the mean down. The F1-score is calculated using the formula:

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

With the main metrics explained, a set of observations must be noted to illus-

trate how small adjustments of the training parameters can influence the results of the evaluation. Because the anomaly engine models uses different results for each algorithm, all of them must be normalized and a threshold must be calculated so that results higher than it are marked as malicious and the others as benign. That threshold is known in the code as the θ variable. The methodology 3 chapter mentions that theta should be calculated as the 99th percentile of the benign identified traffic, with the main objective being of mathematically enforcing a strict limit to False Positives, thus ensuring the engine only flagged traffic that was deviating beyond 99% of established normal behavior. Upon testing and observing the performance evaluations of each model, LOF showed to be the most influenced by the position of the percentile. When reviewing the evaluation made with $\theta = 99$ as shown in the Figure 4.2, the recall value for attack sits at 12% and the precision for benign sits at 53%, meaning that LOF is not able to detect subtle local density deviations that translates in bad detection of low-and-slow stealth intrusions. To fix this, θ was dropped to 95, with the results showing that LOF is having a healthy recovery, as shown in the Figure 4.3, with the benign precision climbing to 92% and malicious recall climbing to 91%

```
[Local Outlier Factor (LOF)] - Individual Theta: 0.0000
              precision    recall  f1-score   support

  Normal (0)       0.53      0.99      0.69     99898
 Anomaly (1)       0.92      0.12      0.21     99898

    accuracy                           0.55    199796
   macro avg       0.72      0.55      0.45    199796
weighted avg       0.72      0.55      0.45    199796
```

Figure 4.2: LOF low performance evaluation using $\theta = 99$

```
[Local Outlier Factor (LOF)] - Individual Theta: 0.0000
              precision    recall  f1-score   support

  Normal (0)       0.92      0.95      0.93     99898
 Anomaly (1)       0.95      0.91      0.93     99898

    accuracy                           0.93    199796
   macro avg       0.93      0.93      0.93    199796
weighted avg       0.93      0.93      0.93    199796
```

Figure 4.3: LOF increased performance evaluation using $\theta = 95$

4.3 XAI testing

To test whether the proposed section of the system which includes explainable AI could work with the features the dataset uses, within the Jupyter lab, an AI testing bench has been set up. The bench tests if the usage method works and if the LLM is able to pinpoint and explain what features turned out to be the decision factor for malicious detection. The bench also tests how the LLM responds based on the two confidence types: high confidence $\geq 80\%$ notification and low confidence $< 80\%$ alert dialog. To see results for both situations, two random anomaly results have ben extracted from the training results that have a high and low detection confidence and had been fed to the LLM engine to analyze and react. The prompts used are as follow:

- ***For low confidence***: You are an expert SOC analyst. Our hybrid IDS (Signature + Anomaly) has low confidence confidence % of that attack about this packet. feeds raw features with values. Provide a decision prompt for the human admin: 1. Briefly analyze the anomaly or signature risk. 2. Recommend either [Block] or [Allow]. Keep it under 3 sentences.

- ***For high confidence***: You are an automated IDS reporting tool. The system

successfully blocked/allowed a packet with high confidence. feeds raw features with values. Write a 1-sentence automated log entry summarizing why this packet was blocked/allowed.

This way, the results show that Ministral is able to intercept, analyze, and perfectly interpret the prompts, proving it can be helpful for the SOC analyst in both understanding the root cause of the attack and where they should look for investigating the low confidence detection, as seen in the Figure 4.4.

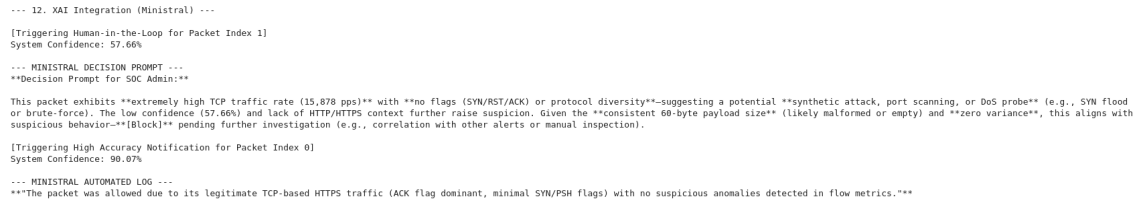


Figure 4.4: Ministral integration and interpretation testing with random analysis results

4.4 Final Script

Once the training is done, the final IDPS script can be put in place so that it can be tested with attacks and benign traffic. Before doing that, the trained models must be exported so that training is not done while analyzing live traffic. This way, the script is safe from malicious training attacks and reduces the usage of CPU, GPU, and memory. The necessary exported files are the trained models themselves, the calculated θ threshold, and the scaler of the features. The models, except for the autoencoder, are exported as *pkl* files, the autoencoder is saved as a *keras* file, θ is saved as a *txt* file, and the scaler is saved as a *pkl* file. Once the files have been exported, the script can be assembled into a python executable that can be exported in a Docker container. The executable itself is composed of 3 main functions that

handle traffic sniffing and extraction of the 39 features for the models, connect to the Ministral API, and manage the traffic window analysis and protection. The flow of this script follows the methodology flow discussed in the 3 chapter. For testing purposes, to see if the script is able to sniff packets across the device and analyze them, the code was modified so that it sniffs on the *wlan*0 network interface of the device. Furthermore, the code is able to sniff the interface, transform the packets into the 39 features necessary for the models, and determine whether the window is malicious or not. So far, the results in figure 4.5 illustrate that normal traffic occurring for the development machine contains flows or packets that are alien to the trained environment of the models. This shows that the testing phase of this script must be conducted using attacks designed for IoT infrastructure and not for general traffic.

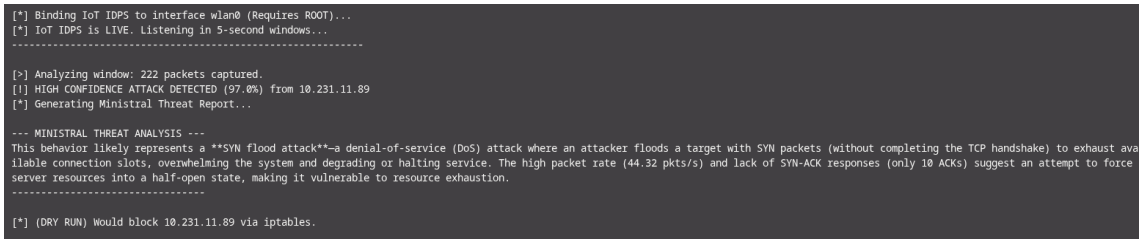


Figure 4.5: Initial testing of the proposed architecture on *wlan*0 interface

5 Performance Evaluation

This chapter performs an evaluation and analysis of the proposed architecture in two critical stages: the training of the models and the complete deployment. Each of these evaluations represents a critical point in determining whether the framework has any potential to be a valid proof of concept for a robust and scalable product in the future. The training performance measures how well the models trained on the dataset and any inconsistencies that have been identified. In this evaluation, the proposed architecture is compared with the implemented ZeroDefense architecture [23], focusing on data preparation methods, individual model training dynamics, and isolated engine performance, with both approaches being trained on the CICIoT2023 dataset [2]. The second testing phase focuses on evaluating the raw performance of the proposed framework in a simulated real-world environment, where the engine is faced with clean and malicious traffic, aiming to check and interpret how the engine behaves and to determine what problems occur and what needs to be fixed in the future.

5.1 Training performance

This section illustrates the principles used in the proposed architecture in comparison to the ZeroDefense paper [23] to achieve a similar goal: to have a performant model and engine metrics. The comparison includes setup differences for data collection, preparation, and scalability; differences in the models used for each engine

and differences in training results. Overall, these comparisons will also illustrate differences in scope for both papers, with one aiming for theoretical results and training precision, while the other focuses on creating a robust implementation for edge deployment that is open to scalability.

5.1.1 Setup differences

To demonstrate that the proposed architecture is not a 1 to 1 copy of the ZeroDefense framework, a couple of details have been modified to ensure better overall performance compared to their approach. These modifications range from how the data is prepared for training to the number of algorithms used and how the engines interact with each other.

Data preparation

The results of each training session are directly proportional to the amount of data that is used. Using this rationale, the more data used from the dataset, the better the results should be for the models. However, for the proposed architecture, the main limitation for training was the RAM usage on the training server. While the ZeroDefense paper used the whole CICIoT2023 dataset, the proposed framework aims to use only one million rows randomly picked from each directory of the dataset (random picks for attacks and random picks for benign traffic), maintaining a non-biased training scenario for each model The only exception being the anomaly engine, where the models must be trained on benign traffic only. This critical detail makes the performance comparison between the proposed architecture and the ZeroDefense framework more interesting because if the results are better, it could easily illustrate the architectural downsides of the original framework.

Data Scaling and Normalization

Although both architectures use an 80/20 stratified split for training and testing, a critical differentiation appears in the feature scaling methodology. The original framework employed a Z-score standardization [44] to process the network features. However, the proposed architecture replaces it with Min-Max Scaling, focusing mainly on simplifying the process. This occurs because, while their framework uses Z-score standardization for the features, they still need to apply the Min-Max scaler within the anomaly engine in order to normalize the results in a $[0, 1]$ interval. Other than that, the scaler has been fitted exclusively on the training distribution, meaning that all features are mathematically bounded to that interval. Another advantage it offers is that dataset features with extreme outliers, such as the massive packet byte counts during a DDoS attack, are blocked from disproportionately dominating the Euclidean distance calculations within the LOF model.

Data Classification

According to the ZeroDefense paper, the authors aimed to create and test how well a framework would perform on the CICIoT2023 dataset, meaning that they wanted to inspect how well each model and engine performs on different classes of data. These classes are logically differentiated by benign traffic and the 33 types of attacks that the creators of the dataset tested on the IoT network test bench. The proposed implementation aims to create a framework that is easy to train and deploy on an edge gateway, having scalability as a main architectural shaper. In this way, the proposed implementation only uses two classes: benign traffic and the 33 attack types, which are combined into one unitary class of attack. This drastically simplifies the model and engine evaluation, implying a more general view of performance and a far better training behavior for the models because it allows for the data to be balanced to the amount of collected benign traffic.

Supervised engine configuration

The original framework uses a supervised engine composed of four algorithms: Random Forest (RF), XGBoost, LightGBM, and TabNet [46], which is a deep neural network that uses sequential attention mechanisms to mimic the performance of tree-based models while providing interpretability, making it more efficient on tabular datasets such as the currently used CICIoT2023 compared to other classic deep-learning algorithms. The proposed architecture ditches the TabNet algorithm in order to lower the performance footprint on the gateway, and due to the low performance it offered on the ZeroDefense framework, which had a 2603.12 second training window and a poor macro-average F1-score of 54.55%. By using only the tree-based algorithms, the proposed system reduces the total computational training time from the original 6731.84 seconds to 16.32 seconds.

Autoencoder Bottleneck Optimization

The original engine uses a 46-32-46 deep autoencoder architecture, while the proposed architecture uses a much more optimized version of the dataset with only 39 features, making a 39-32-39 architecture too loose for the dynamic physical features extracted from the dataset. By lowering the bottleneck to just 16 neurons, the implementation forces the neural network to learn a stricter mathematical logic for the benign traffic, thereby achieving a reconstruction F1-score of 96%.

5.1.2 Supervised engine results

This section illustrates the differences in performance at the Signature Engine level. As previously mentioned, the proposed implementation aggregates the 33 specific attack vectors evaluated in the original ZeroDefense framework into a singular "Attack(1)" class that enables binary classification optimized for edge deployment and execution. Because of this critical difference in classification scope, the following

evaluation will discuss the metrics obtained from the local implementation while including the ZeroDefense multi-class results purely as narrative context.

Model-level Evaluation

The individual gradient boosting and tree-based models were evaluated on their ability to distinguish between normal IoT traffic and malicious spikes. Across all three classifiers, the proposed binary implementation achieved surprising stability and consistency, theoretically outperforming the multi-class unbalanced evaluation reported in the original paper.

- **Random Forest:** the local RF implementation achieved a general accuracy of 98.27% with a macro-average F1-score of 0.98. At the class level, the model demonstrates perfect recall (1.00) for benign traffic and perfect precision (1.00) for the attack class. As context, the original framework reported a macro-average F1-score of 80.5% when evaluating against its 34-class problem, highlighting an increased complexity of multi-class categorization in comparison to a binary approach. The table 5.1 illustrates the local results in a much more organized layout.

Table 5.1: Random Forest Performance: Class-Level and Macro Averages

Class	**Precision (%)**	**Recall (%)**	**F1-score (%)**
Benign (0)	97.00	100.00	98.00
Attack (1)	100.00	97.00	98.00
Macro Avg	98.00	98.00	98.00

- **XGBoost:** the original framework marked XGBoost as the most dominant model, achieving a 83.97% mean F1-score across its 34 classes. The local implementation benefits from the binary classification and achieves an accuracy of 98.21% with a macro-average F1-score of 0.98. Similar to the RF model,

XGBoost achieves a perfect precision score for the attack class and perfect recall for the benign class. This guaranties that when the models flag a payload as malicious, they are correct. However, the 0.97 recall score for attacks shows that a small portion of stealthy attacks can still be marked as benign.

Table 5.2: XGBoost Performance: Class-Level and Macro Averages

Class	**Precision (%)**	**Recall (%)**	**F1-score (%)**
Benign (0)	97.00	100.00	98.00
Attack (1)	100.00	97.00	98.00
Macro Avg	98.00	98.00	98.00

- **LightGBM:** this evaluation highlights a significant architectural advantage in the localized framework. The original paper reported a shocking struggle for this model on the multi-class dataset, reporting a mean F1-score of 36.41%. By applying the Min-Max scaler and simplifying the decision boundary to a binary threshold, the local LightGBM model emerged as a highly reliable component for the fused engine, with a 98.26% accuracy and a 0.98 macro-average F1-score.

Table 5.3: LightGBM Performance: Class-Level and Macro Averages

Class	**Precision (%)**	**Recall (%)**	**F1-score (%)**
Benign (0)	97.00	100.00	98.00
Attack (1)	100.00	97.00	98.00
Macro Avg	98.00	98.00	98.00

Engine-level evaluation

After the individual performance evaluation, the models were integrated using the same weighted fusion equation used in the ZeroDefense paper to form the unified Signature Engine.

The local fused ensemble achieved an overall accuracy of 98.26% and a macro-average F1-score of 0.98. Evaluating the engine at the class level, the metrics demonstrate overall stability and accuracy for a deployable IoT gateway:

- **Benign class (0):** Precision: 0.97, Recall: 1.00, F1-Score: 0.98.
- **Attack class (1):** Precision: 1.00, Recall: 0.97, F1-Score: 0.98

By converting the engine to a binary paradigm, the local ensemble pushes the supervised macro-average to a comfortable 0.98, peaking at the performance of its best standalone models.

Table 5.4: Supervised Engine (Fused Ensemble) Performance

Class	**Precision (%)**	**Recall (%)**	**F1-score (%)**
Benign (0)	97.00	100.00	98.00
Attack (1)	100.00	97.00	98.00
Macro Avg	98.00	98.00	98.00

To better visualize how the weight system balances the performance of the engine, two plots were generated that show how the macro average and the binary classification results affect the fused engine results.

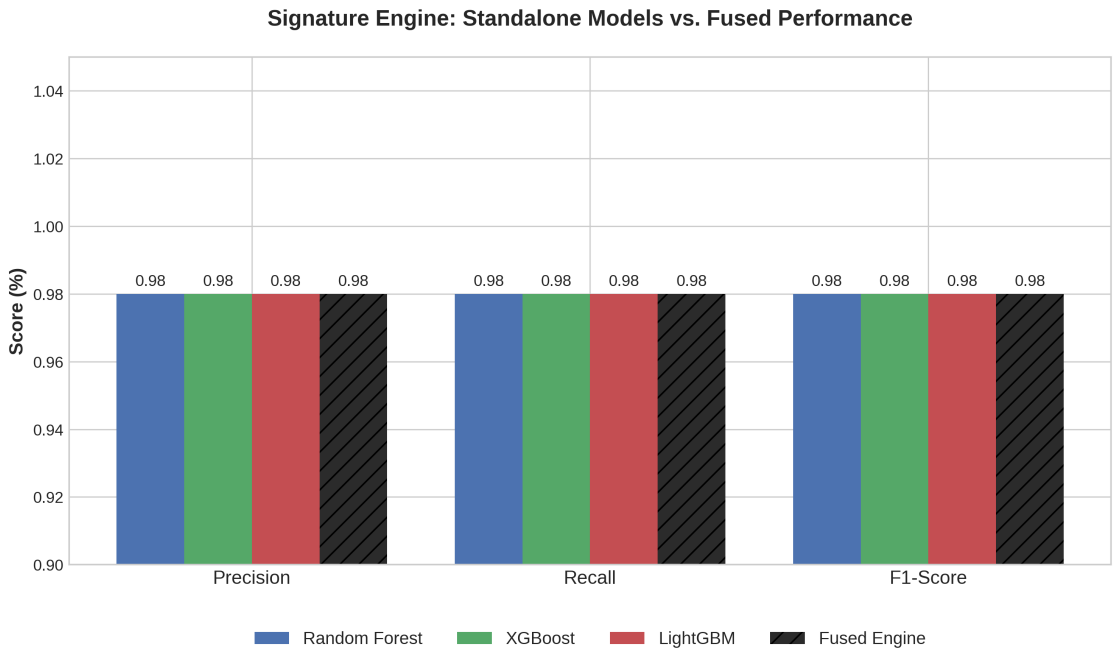


Figure 5.1: Influence of individual models macro average results over the fused signature engine

Figure 5.2: Overall classification results compared to macro average of the fused signature engine

Justification for the Fused Architecture

At a more detailed glance, the macro average F1-score of the fused ensemble appears identical to the standalone performance of the individual models. This raises an important architectural question: does the computational overhead of running a fused ensemble justify itself over deploying a single, highly performant model?
To answer this, it is important to move away from raw metric analytics to a more general view of the architecture and how it responds to attacks. Individual tree-based models, despite high accuracy, are vulnerable to specific algorithmic blind spots and can be bypassed by attacks. By utilizing this weighted mechanism, the fusion engine acts as a safety net. In the case where a novel attack bypasses the decision boundaries of the RF model, XGBoost and LightGBM can compensate, ensuring a safe degradation rather than a total detection failure. On top of that, because the detection is dependent on the Inter-Engine Orchestrator and the XAI integration, the framework relies heavily on calibrated confidence to determine whether to trigger an autonomous **iptables** drop ($\geq 80\%$ confidence) or to alert the SOC analyst. A single model can often predict a confidently false positive that heavily influences the orchestrator. Using the fused engine, these probabilities are reduced, preventing large quantities of false positive alerts and an autonomous lockout.
Finally, while deploying three classifiers involve a higher computational than deploying one, the selected algorithms (RF, XGBoost, LightGBM) are more lightweight in comparison to using heavy DL models, such as the dropped TabNet.

5.1.3 Anomaly engine results

This section focuses on evaluating the unsupervised Anomaly Engine, which is responsible for detecting zero-day attacks in the traffic. It is critical to first explain a major methodological difference in how the anomaly models are evaluated in comparison to the original ZeroDefense paper. The referenced authors focused on eval-

uating their individual models exclusively on their ability to detect rare minority attacks, where the classes containing fewer than 1000 training samples were chosen. Therefore, they only reported the minority class recall scores to justify their fusion weights. Furthermore, by adapting the local framework to a balanced binary classification paradigm, the evaluation process shifts from isolated minority to a general assessment of the engine's discriminative power. Each model had its 95th percentile threshold (θ) calculated and was allowed to extract full classification metrics that allow rigorous model-level and engine-level analysis.

Model-Level Evaluation

The individual unsupervised classifiers were evaluated on their ability to solely distinguish benign IoT traffic from anomalous payloads based on reconstruction error, isolation path length, and local density deviation.

- **Autoencoder:** across the local generalized binary distribution, the optimized AE with a 16 neuron bottleneck emerges as the overwhelmingly dominant detector. It achieved a macro-average F1-score of 0.96, while the high precision for benign traffic (0.97) and high recall for anomalies (0.97) prove that the tightened bottleneck helps the classifier learn the mathematical bounds of normal traffic. For context, the AE was reported as the weakest minority-attack detector in the original multi-class study, with a general 69.3% recall for rare attacks.

Table 5.5: Autoencoder (AE) Performance: Class-Level and Macro Averages

Class	**Precision (%)**	**Recall (%)**	**F1-score (%)**
Normal (0)	97.00	95.00	96.00
Anomaly (1)	95.00	97.00	96.00
Macro Avg	96.00	96.00	96.00

- **Isolation Forest:** displayed the weakest general performance, with a macro-average F1-score of 0.87. A more careful look at the class-level metrics reveals the main culprit: a low precision for normal traffic (0.83), which indicates a high false-positive rate linked to a lower anomaly recall (0.80). This behavior confirms a theoretical hypotheses that the algorithm's random-partitioning trees can struggle to separate genuine anomalies from slight normal telemetry fluctuations in such uniform IoT environments.

Table 5.6: Isolation Forest (IF) Performance: Class-Level and Macro Averages

Class	**Precision (%)**	**Recall (%)**	**F1-score (%)**
Normal (0)	83.00	95.00	88.00
Anomaly (1)	94.00	80.00	86.00
Macro Avg	88.00	87.00	87.00

- **Local Outlier Factor:** The authors favored LOF due to its superior rare-attack detection (78.4%). The local generalized evaluation confirms its robustness, achieving a macro-average F1-score of 0.93. It maintained a strong balance between benign precision (0.92) and anomaly recall (0.91). Its calculated θ of 0.0000 strongly highlights the extreme mathematical density of the benign IoT background traffic.

Table 5.7: Local Outlier Factor (LOF) Performance: Class-Level and Macro Averages

Class	**Precision (%)**	**Recall (%)**	**F1-score (%)**
Normal (0)	92.00	95.00	93.00
Anomaly (1)	95.00	91.00	93.00
Macro Avg	93.00	93.00	93.00

Engine-level Evaluation

After the individual performance testing, the models were integrated using the same weighted anomaly fusion equation established by the authors. A global threshold of $\theta = 0.1423$ was dynamically calculated from the 95th percentile of the fused benign scores. The fused engine achieved an overall macro-average F1-score of 0.96, perfectly matching the peak performance of the AE model while benefiting from the theoretical robustness of an ensemble:

- **Benign class (0):** Precision: 0.97, Recall: 0.95, F1-Score: 0.96
- **Attack class (1):** Precision: 0.95, Recall: 0.97, F1-Score: 0.96

When evaluating the fused engine, the advantages of the weighted system become clear. It is capable enough to successfully neutralize the IF's low results and love for false positives, lifting the ensemble's benign precision to 0.97. Furthermore, the engine proves itself as being a reliable zero-day anomaly detector, thanks to its 0.97 recall score, meaning that it can detect 97% of malicious traffic based purely on mathematical deviation, without relying on any existing attack signatures.

Table 5.8: Fused Anomaly Engine Performance

Class	**Precision (%)**	**Recall (%)**	**F1-score (%)**
Normal (0)	97.00	95.00	96.00
Anomaly (1)	95.00	97.00	96.00
Macro Avg	96.00	96.00	96.00

To better visualize how the weight system balances the performance of the engine, two plots were generated that show how the macro average and the binary classification results affect the fused engine results.

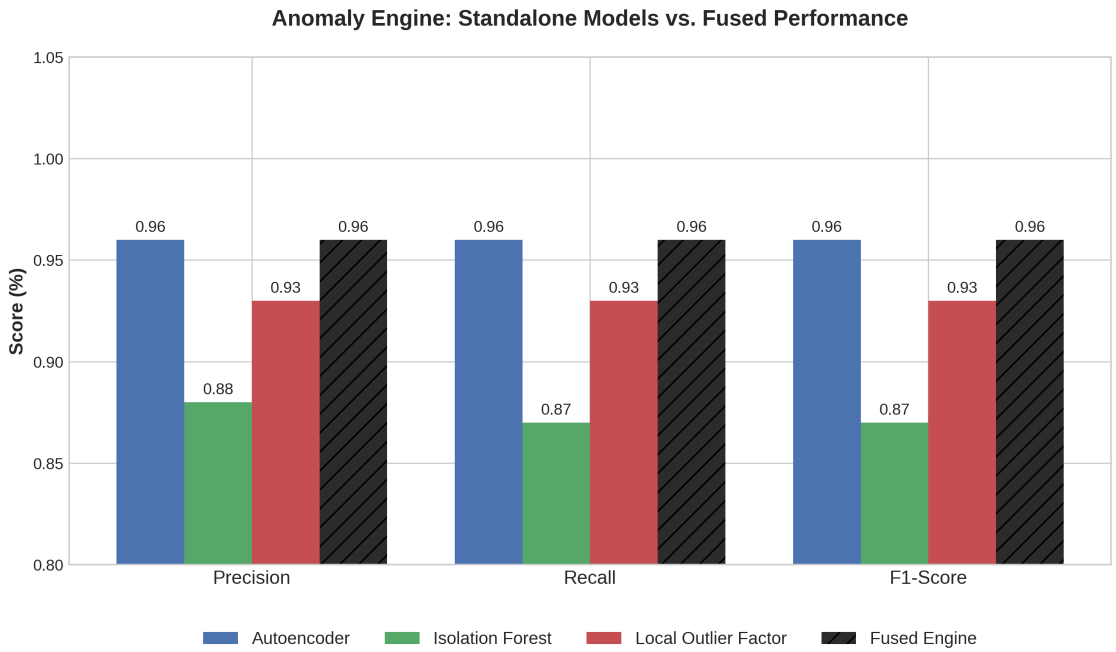


Figure 5.3: Influence of individual models macro average results over the fused anomaly engine

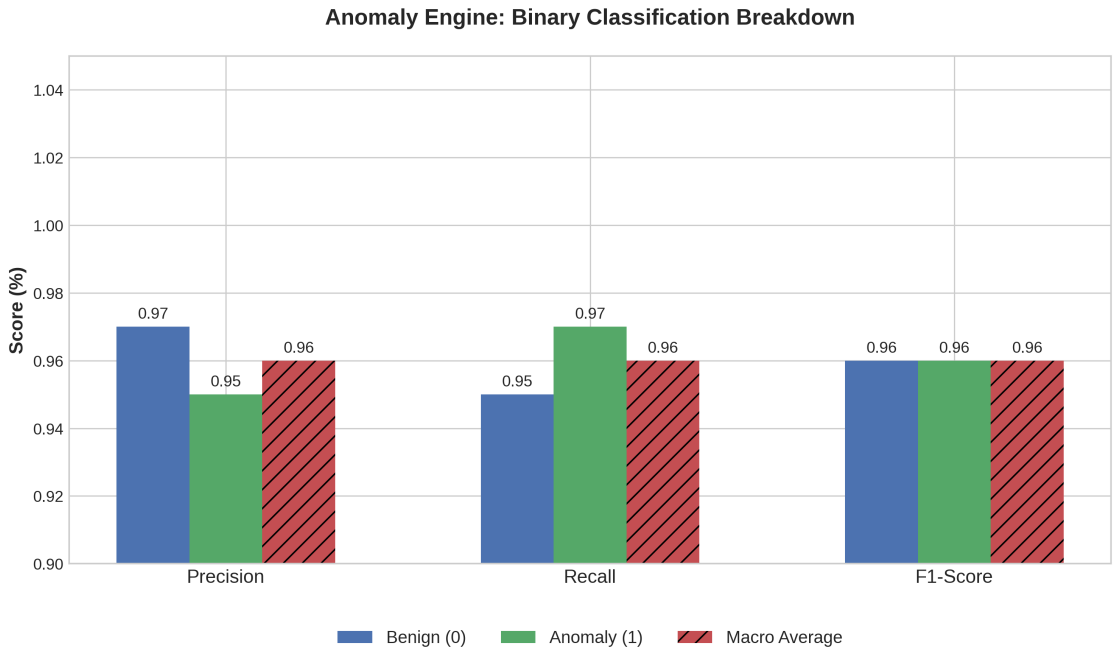


Figure 5.4: Overall classification results compared to macro average of the fused signature engine

5.2 Deployment Performance

This section illustrates the practical deployment and real-time evaluation of the proposed IDPS architecture within an isolated, containerized test bench. while the previous section presented theoretical metrics that prove the mathematical validity of the models, the deployment phase tests the overall system viability as an active, real-time edge security gateway.

5.2.1 Setup and configuration

While the ultimate goal for this framework is to deploy it on a physical, resource-constrained edge device, this proof-of-concept was deliberately deployed in an isolated, containerized environment using Docker. This approach allows for a highly controlled, observable, and reproducible test bench. The test environment consisted of two nodes that were deployed on an isolated bridge network:

1. **the IDPS Edge Node**: serving as the defender and the gateway for the IoT network;

2. **the Parrot OS container**: serving as the attacker that tries to penetrate or overwhelm the IoT network;

Furthermore, evaluating destructive, high-volume network threats requires a sterile environment to prevent accidental degradation of external physical networks. Deploying the IDPS and Parrot OS attacker on a dedicated, virtualized Docker network ensures that the high-volume malicious traffic is safely contained without triggering ISP (Internet Service Provider) or campus network alarms, while still providing identical Layer 3 and Layer 4 network behaviors for the models to analyze.

To accurately simulate the computational constraints of an actual IoT gateway, the IDPS was built on a minimal `Python 3.10-slim` base image that forces the framework to operate with strict memory and dependency management. Other critical

system-level dependencies were integrated, including `libcap-dev` for raw packet sniffing and **iptables** for active threat mitigation. Software-wise, the IDPS relies on `tensorflow 2.21.0` for the neural network backend, specifically the autoencoder, together with `xgboost`, `lightgbm`, and `dpkt` to parse packets from the raw traffic. On top of that, because the IDPS needs to process raw data and drop malicious traffic at the kernel level, the container was deployed with `NET_ADMIN` and `NET_RAW` capabilities. Parrot OS, on the other hand, is being set up using a kasmweb image, meaning that the virtual machine can be run inside a web browser, eliminating many resource constraints imposed by the host machine. In the end, the final setup directory would look like this:

- **Dockerfile**: the command list that needs to be run when building the containers;
- **docker-compose.yml**: the blueprint that tells docker what each container should pull, look like and their specific proprieties, or permissions;
- **requirements.txt**: the libraries list that tells pip what to install in order to make the final script work;
- **thesis_models**: the directory that contains the exported models from the Jupyter Lab after training;
- **IoT_IDPS_live.py**: the script itself that contains the logic explained in chapter 3 and chapter 4;

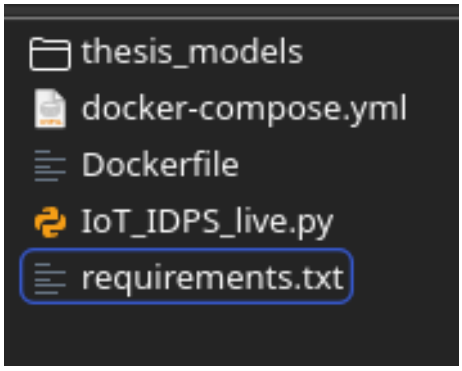


Figure 5.5: All docker configuration files in place

By successfully deploying the parallel detection engine, XAI translator, and dynamic **iptables** mitigation within this constrained container, the framework mathematically and operationally validates the proposed architecture. This software-first validation aims to prove that the system's logic functions in real-time under stress, establishing a solid foundation for future integration onto physical edge hardware.

5.2.2 Deployment Challenges and Optimizations

The transition from an initial functional test on the host machine to the live Docker container environment revealed significant framework-level compatibility issues that forced the script to be adapted. A critical serialization bug has been identified between the Keras 2 and 3 architectures, where residual `quantization_config` metadata caused the model deserializer to fail during the deployment of the Autoencoder model. On top of that, Docker's aggressive layer caching repeatedly masked these issues, leading to the rebuilding of the containers again and again with no progress. To fix this, the script was modified to bypass the whole deserialization process, thus eliminating the error. This was done by exporting only the weights of the classifier, using a raw `.weights.h5` file, and the topological blueprint of $39 - 16 - 39$ was hardcoded inside the final script using the Keras libraries. This way, the serialized model is no longer present, and the Keras deserializer cannot complain of missing

metadata, thus allowing the initialization of the models to be completed.

5.2.3 Real-Time Attacks and Interpretation

Once the test bench had been stabilized and the IDPS engine can gather an IP address within the virtualized network, the real testing phase can begin. The IDPS engine had been left with the `DRY_RUN` flag turned on, in order to just see what the engine is trying to block without actually doing it. The attacks conducted from the Parrot OS container appear in the CICIoT 2023 dataset as well, therefore the IDPS engine should be able to detect them properly and block Parrot OS's IP address. According to the figure 5.6, the address attributed to the IDPS container is `172.18.0.2`, and the address attributed to the Parrot OS is `172.18.0.3`.

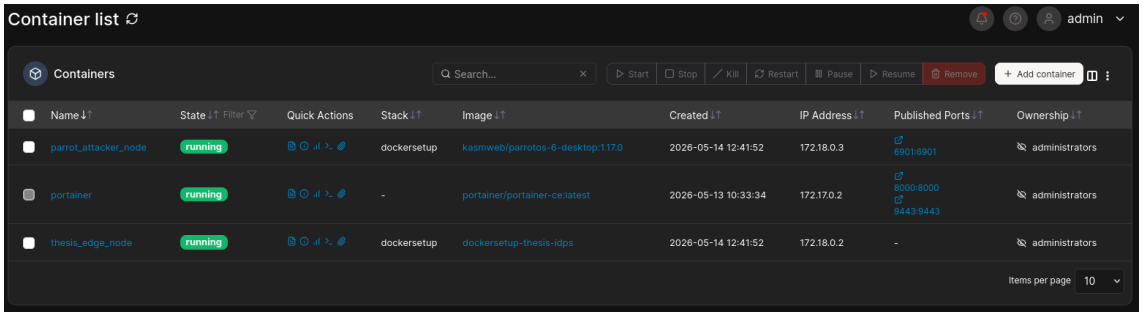


Figure 5.6: Atributed addresses for the testing containers seen in Portrainer GUI (Graphical User Interface)

The attacks that were attempted on the previously presented test bench are as follows:

1. **Reconnaissance (Aggressive Port Scan):** An `nmap` aggressive SYN scan (`-p- -T4 -A`) was launched on the edge node to see whether it could receive and analyze them. Fortunately, the IDPS successfully captured 131,100 packets within the 5-second capture window, and it was able to mark the malicious

traffic with a 98.8% High Confidence score, correctly identifying the attacker's IP address. The XAI component fails to properly recognize the type of attack, categorizing it as a classic SYN flood attack;

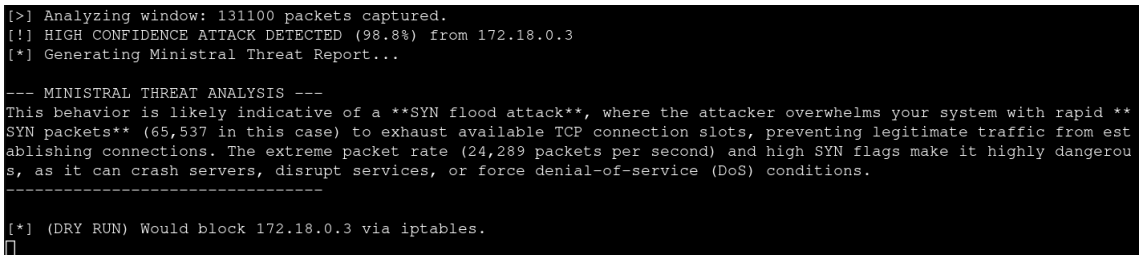


Figure 5.7: IDPS recognises and blocks the aggressive port scanning

2. **DoS attacks (ICMP SYN flood):** aim to test the system's durability against pure volumetric exhaustion with zero packet variance. In this regard, an ICMP flood was initiated using the `ping -f` command. The IDPS parsed over 500,000 packets per 5-second window without failing, and it registered a 97.2% confidence score. The XAI component seems to struggle with interpretation, the notification stating that it is either a classic TCP SYN flood or an ICMP SYN flood, therefore calling for a prompt adjustment;

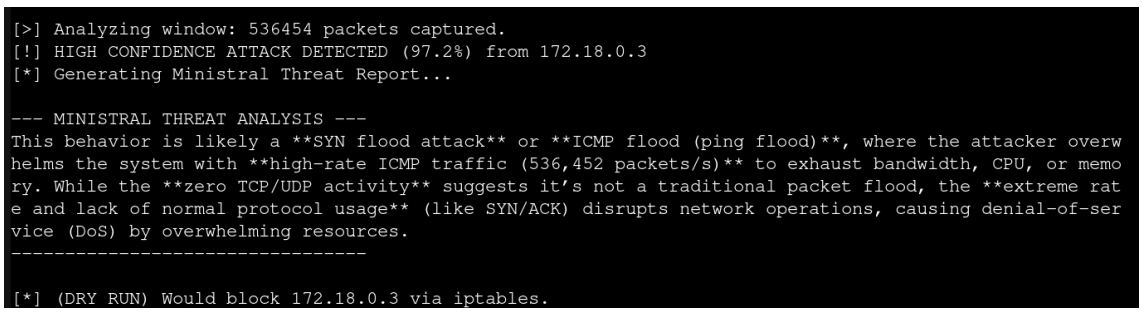


Figure 5.8: IDPS recognises and block the ICMP SYN flood attack

3. **Protocol Exhaustion (TCP SYN Flood):** a proper SYN flood attack was

launched with half-open connections in mind, using the `hping3` command. The system processed over 620,000 malformed packets in a single window and achieved a 98.7% confidence score. This time, the XAI component managed to properly detect the correct properties of the attack, further emphasizing that there is a need for prompts reconstruction;

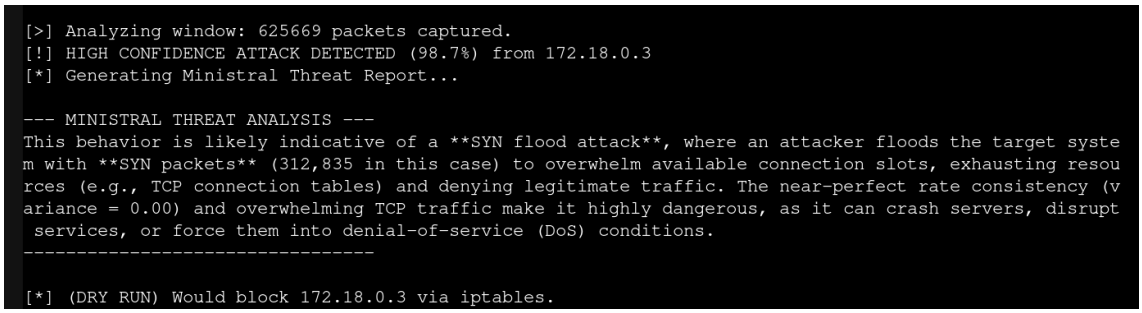


Figure 5.9: IDPS recognises and blocks the TCP SYN flood attack

4. **Man in the Middle attack (ARP Spoofing)**: this rare IoT attack tries to fool the gateway that the attacker's MAC (Media Access Control) address belongs to the router, making this attack sit in layer 2 and layer 3 of the OSI scheme, using the `arpspoof` command. At first the IDPS manages to detect the attacker's IP address but then it tries to block itself, probably because of failed SYN-ACK packets that the firewall interprets as a possible attack. The XAI component makes the interpretation of the result more difficult due to marking it as a classic SYN flood attack;

```
[*] Ignored window: Traffic was non-IPv4 (e.g., ARP, IPv6, or background noise).

[>] Analyzing window: 7 packets captured.
[!] HIGH CONFIDENCE ATTACK DETECTED (93.5%) from 172.18.0.3
[*] Generating Ministral Threat Report...

--- MINISTRAL THREAT ANALYSIS ---
This behavior likely represents a **SYN flood attack** or **UDP flood attack**—though the low packet
 rate and lack of SYN/ACK flags make it less classic than a traditional SYN flood. The **UDP/ICMP tr
affic with no TCP flags** suggests it could be a **denial-of-service (DoS) attack** exploiting resou
rce exhaustion (e.g., overwhelming a server with unsolicited packets), which disrupts services and d
egrades performance.
---------------------------------

[*] (DRY RUN) Would block 172.18.0.3 via iptables.

[>] Analyzing window: 29 packets captured.
[!] HIGH CONFIDENCE ATTACK DETECTED (96.7%) from 172.18.0.2
[*] Generating Ministral Threat Report...

--- MINISTRAL THREAT ANALYSIS ---
This behavior likely represents a **SYN flood attack**, where the attacker sends rapid **SYN packets
** (2 out of 20 TCP connections) to overwhelm the target system's ability to respond, exhausting ava
ilable TCP connection states and causing denial-of-service (DoS). The high variance in packet rate a
nd lack of normal traffic patterns further suggest malicious intent, as legitimate networks maintain
 stable, predictable activity.
---------------------------------

[*] (DRY RUN) Would block 172.18.0.2 via iptables.
```

Figure 5.10: IDPS struggles to identify the correct attacker and tries to block itself

5. **TCP ACK Fragmentation**: compared to the TCP SYN attack, this type of DoS sends incomplete or broken ACK packets, which the firewalls and IDPS struggle with because they try to hold the fragments in memory to reassemble them, crashing the system without a massive spike in bandwidth. The IDPS fails to properly detect this attack due to its attempt to block itself. It finds around 25 to 50 packets per traffic window, and the culprit address it identifies is its own IP. However, an interesting observation is that the IDPS correctly identifies the attacker's IP when the attack just starts, proving initial potential in identifying this silent attack. The XAI component seems to be able to catch a small amount of detail in the analyzed window where the correct attacker has been identified;

```
[>] Analyzing window: 2 packets captured.
[!] HIGH CONFIDENCE ATTACK DETECTED (95.4%) from 172.18.0.3
[*] Generating Ministral Threat Report...

--- MINISTRAL THREAT ANALYSIS ---
This behavior likely represents a **SYN flood attack** or **low-rate, high-volume probing**—though the
near-zero packet counts and flags suggest it may be a **SYN flood attempt disguised as idle traffic** (
e.g., a stealthy probe or misconfigured service). While not immediately destructive, it could indicate
**network reconnaissance** (e.g., for future DoS attacks) or **exploit targeting** (e.g., probing for o
pen ports), posing a risk by **exhausting resources** (e.g., TCP SYN queues) or enabling attackers to l
ater launch more harmful attacks.
---------------------------------

[*] (DRY RUN) Would block 172.18.0.3 via iptables.

[>] Analyzing window: 51 packets captured.
[!] HIGH CONFIDENCE ATTACK DETECTED (97.7%) from 172.18.0.2
[*] Generating Ministral Threat Report...

--- MINISTRAL THREAT ANALYSIS ---
This behavior likely represents a **SYN flood attack**—a denial-of-service (DoS) attack where an attack
er sends a high volume of **SYN packets** (without completing the TCP handshake) to overwhelm a server'
s resources, exhausting available connection slots and causing service disruption. The rapid **SYN flag
s (2 packets) with high variance and packet rate (7.52 pkts/s)** suggests an attempt to saturate the ta
rget's connection table, making legitimate traffic fail.
---------------------------------

[*] (DRY RUN) Would block 172.18.0.2 via iptables.
```

Figure 5.11: IDPS shows potential in identifying the correct attacker at the first glance

5.2.4 Architectural Limitations

While the proposed architecture demonstrates exceptional accuracy and resilience during high-volume attacks, the real-world deployment also exposed several complex edge cases that include both the Machine Learning analysis and final script implementation details. These problems include self backstabbing, noise misinterpretation, and layer 2 blindness.

TCP Backstabbing Paradox

During rare attacks that include ARP spoofing and fragmented TCP ACK floods, the IDPS displayed a tendency to classify its own IP address as the main malicious actor, causing a TCP Backscatter false positive paradox. Because the attacker is trying to flood the closed ports of the edge node, its native Linux kernel auto-

matically responds with `RST/ACK` (reset) packets. This behavior is marked by the IDS engine as malicious due to its high volume of broken packets, thus tending to backstab itself.

Noise misinterpretation

During non attack scenarios or in moments of relaxation, when most of the traffic is either refreshing or passing by, the IDPS experiences instability by marking the clean, healthy traffic as malicious and attempting to block itself again.

Layer 2 blindness

Because the feature extraction pipeline was designed to extract only layer 3 (IP) and layer 4 (TCP/UDP) traffic, the IDPS becomes blind to data link layer attacks such as ARP spoofing. This was demonstrated in one of the attacks, where the edge node was only able to recognize layer 3 traffic and initially tried to block the correct attacker, eventually moving to block itself afterward.

XAI Contextual Bias and Misinterpretation

During the deployment testing phase, the XAI integration demonstrated contextual bias. As shown in the generated logs, the Ministral model frequently generalized various volumetric attacks as "TCP SYN floods", regardless of the actual protocol flags triggered in the attack payload. This indicates that the current prompt structure fails to restrict the LLM's reasoning, allowing it to generalize based on training biases rather than strictly analyzing the provided features. Currently, the component introduces diagnostic confusion rather than assisting the SOC analyst.

5.3 Future steps

Although the system demonstrated its potential in identifying common attacks from the IoT space and blocking them, there are still several details that need to be addressed and fixed. These details include:

- **Train on the whole dataset**: due to hardware limitations, the training was performed only on around one million samples from the dataset. The future goal is to train the models on the entire dataset and overcome the challenges this goal might impose, such as unbalanced samples for binary categorization (0: benign, 1: attack)
- **Self backstabbing fix**: adjust the final script to identify its own and the router IPs so that they cannot be targeted by the models as malicious
- **XAI prompt engineering and context restrictions**: a better iteration of the current generalizing prompt would include stricter rule sets for differentiating protocols and supplying diverse examples of attack signatures in the system prompt.
- **Rare attack testing**: due to the partial blindness of the IDPS system against rare attacks, the main focus will be to extend training on those types of attacks and readjust the feature extraction engine to better meet those needs
- **Enable Layer 2 evaluation**: because the ARP spoofing attack was not identified and most ARP based traffic is ignored on the feature extraction engine, it is necessary to reconstruct it, making layer 2 analysis possible and including fixes to the previously mentioned problems that start from this point, including IP whitelists.
- **Implement a self adjusting weights system**: although the proposed system included a self adjusting weight system for the signature and anomaly

engine, the time constraints limited the implementation and testing of such a component of the architecture

- **Test on real hardware**: to further demonstrate that the IDPS can perform on low-resource edge devices, it can be deployed on real, computationally constrained hardware, such as the Raspberry Pi.

- **External validation**: after improving the previously mentioned issues, the best testing method is to deploy it on proper edge devices in a real-life scenario. This way, the framework gets real validation from an external actor, such as surveillance companies or local authorities that implement smart city solutions.

6 Conclusions

This thesis touched upon the introduction of machine learning and LLM as automated solutions for active analysis and protection in order to ease the effort of SOC engineers. The thesis presented a machine learning and LLM based IDPS system for the IoT infrastructure that analyzes, discovers, blocks, and notifies any malicious activity related to this type of network. The system was created using the publicly available CICIoT 2023 dataset, with a focus on creating a robust proof of concept capable of being deployed on edge gateways. These contributions directly respond to the research questions formulated in the Section 1.2:

- Addressing RQ1, this research designed a parallel hybrid detection system for both known and zero-day attacks that uses machine learning and deep learning algorithms, proving its robustness and speed in identifying threats while maintaining a strict low-resource footprint required for edge networks.
- To answer RQ2, an LLM-based XAI translator was introduced. By converting raw feature metrics used by the detection models into human-readable prompts, the system establishes trust and reduces the SOC analyst's burden, calling for manual verification only during low-confidence scenarios.
- In response to RQ3, the framework was successfully upgraded from a passive monitor to a complete IDPS through an automated blocking component. By dynamically sending system calls to the **iptables** firewall, the framework quarantines malicious nodes based on AI confidence. Upon testing, this feature was

kept with the `DRY_RUN` flag turned on to see which IP addresses the system would block without pushing the actual rules to the firewall.

- Finally, regarding RQ4, this implementation demonstrates a significant potential impact on the industry by democratizing edge security. It proves that robust, AI-driven mitigation can be scaled to constrained IoT, IIoT, and Smart City environments without relying on expensive, centralized enterprise hardware.

Discovered Insights

Upon developing the proposed system into a functional proof of concept, a series of interesting facts have been discovered:

- Training the models is highly dependent on data preparation: as seen in the performance evaluation between the local framework and the one proposed by the ZeroDefense paper [23], the local classifier focused on providing a plain response (either an attack or not) compared to their approach, where the prepared data had been unbalanced and portrayed a much more accurate result in relation to key aspects such as rare attacks for the zero-day-based models, to which the proposed local implementation is nearly blind.

- The XAI response is highly dependent on the formulated prompt and the data that it is fed. As seen in the deployment testing phase, Ministral struggled to properly classify the specific attacks conducted on the Parrot OS VM, where most of them were noted as TCP SYN flood attacks.

- The proposed implementation has a tendency to self sabotage, meaning that in passing or refreshing traffic, the classifier thinks that the traffic is malicious and proposes that its own IP should be blocked.

References

[1] Statista. "Internet of Things - Worldwide", Accessed: Mar. 10, 2026. [Online]. Available: https://www.statista.com/outlook/tmo/internet-of-things/worldwide?currency=EUR.

[2] T. A. IBAISI, *Ciciot2023 dataset*, 2025. DOI: 10.21227/v63c-9998.

[3] N. Moustafa and J. Slay, "Unsw-nb15: A comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)", in *2015 Military Communications and Information Systems Conference (MilCIS)*, 2015, pp. 1–6. DOI: 10.1109/MilCIS.2015.7348942.

[4] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset", *Future Generation Computer Systems*, vol. 100, pp. 779–796, 2019, ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2019.05.041.

[5] European Union, *Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation)*, Official Journal of the European Union, L 119/1, May 2016. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2016/679/oj.

[6] A. Alfahaid, E. Alalwany, A. M. Almars, F. Alharbi, E. Atlam, and I. Mahgoub, "Machine learning-based security solutions for iot networks: A comprehensive survey", *Sensors*, vol. 25, no. 11, 2025, ISSN: 1424-8220. DOI: 10.3390/s25113341.

[7] Carnegie Mellon University Computer Science Department. "CMU SCS coke machine home page". Page last modified on Feb. 14, 2005. Updates ceased Jan. 2014, Accessed: Mar. 23, 2026. [Online]. Available: https://www.cs.cmu.edu/~coke/.

[8] W. Stewart. "The internet toaster", Living Internet, Accessed: Mar. 23, 2026. [Online]. Available: https://www.livinginternet.com/i/ia_myths_toast.htm.

[9] Computerphile, *World's first webcam - computerphile*, YouTube video, Nov. 2017. [Online]. Available: https://www.youtube.com/watch?v=a4PX8vksBFU.

[10] University of Cambridge Computer Laboratory. "The trojan room coffee machine". The world's first webcam. Connected to the Web in November 1993., Accessed: Mar. 23, 2026. [Online]. Available: https://www.cl.cam.ac.uk/coffee/.

[11] Wikipedia contributors. "Kevin ashton", Wikipedia, The Free Encyclopedia, Accessed: Mar. 23, 2026. [Online]. Available: https://en.wikipedia.org/wiki/Kevin_Ashton.

[12] Wikipedia contributors. "Internet digital DIOS", Wikipedia, The Free Encyclopedia, Accessed: Mar. 23, 2026. [Online]. Available: https://en.wikipedia.org/wiki/Internet_Digital_DIOS.

[13] iRobot Corporation. "History", iRobot, Accessed: Mar. 23, 2026. [Online]. Available: https://about.irobot.com/history.

[14] Wikipedia contributors. "Arduino", Wikipedia, The Free Encyclopedia, Accessed: Mar. 23, 2026. [Online]. Available: https://en.wikipedia.org/wiki/Arduino.

[15] The Centre for Computing History. "Eben upton - the story of raspberry pi", YouTube, Accessed: Mar. 23, 2026. [Online]. Available: https://www.youtube.com/watch?v=UCt6d0SCxO4.

[16] Wikipedia contributors. "Google nest", Wikipedia, The Free Encyclopedia, Accessed: Mar. 23, 2026. [Online]. Available: https://en.wikipedia.org/wiki/Google_Nest.

[17] Wikipedia contributors. "Philips hue", Wikipedia, The Free Encyclopedia, Accessed: Mar. 23, 2026. [Online]. Available: https://en.wikipedia.org/wiki/Philips_Hue.

[18] Wikipedia contributors. "Amazon echo", Wikipedia, The Free Encyclopedia, Accessed: Mar. 23, 2026. [Online]. Available: https://en.wikipedia.org/wiki/Amazon_Echo.

[19] Pandemonium, *The minecraft hacker who broke the internet*, YouTube video, Apr. 2025. [Online]. Available: https://www.youtube.com/watch?v=Lo40iZDrAQQ.

[20] S. S. Sefati, B. Arasteh, S. Halunga, and O. Fratu, "A comprehensive survey of cybersecurity techniques based on quality of service (QoS) on the Internet of Things (IoT)", *Cluster Computing*, vol. 28, p. 792, 2025. DOI: 10.1007/s10586-025-05449-z.

[21] Cisco, *Snort network intrusion detection & prevention system*, https://www.snort.org/, Accessed: 2026-06-04, 2026.

[22] Open Information Security Foundation, *Suricata*, https://suricata.io/, Accessed: 2026-06-04, 2026.

[23] A. Wakili and S. Bakkali, "Zerodefense: An adaptive hybrid fusion-based intrusion detection system for zero-day threat detection in iot networks", *Journal of Electronic Science and Technology*, vol. 24, no. 1, p. 100 345, 2026, ISSN: 1674-862X. DOI: https://doi.org/10.1016/j.jnlest.2026.100345.

[24] A. Srivastava and D. Sinha, "Arlhnids-iot: An accurate and robust lightweight hybrid-nids for iot network security", *Computers & Security*, vol. 156, p. 104 515, 2025, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2025.104515.

[25] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework", in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19, Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 2623–2631, ISBN: 9781450362016. DOI: 10.1145/3292500.3330701.

[26] H. Kosovac Godart. "Double deep q-networks: An introductory guide", Medium, Accessed: May 2, 2026. [Online]. Available: https://medium.com/@helena.godart/double-deep-q-networks-an-introductory-guide-9b0d88310197.

[27] S. S. N, P. P, K. Jain, and P. Krishnan, "Edge ai bridge: A micro-layer intrusion detection architecture for smart-city iot networks", *IoT*, vol. 7, no. 2, 2026, ISSN: 2624-831X. DOI: 10.3390/iot7020033.

[28] L. Yang, A. Moubayed, and A. Shami, "Mth-ids: A multitiered hybrid intrusion detection system for internet of vehicles", *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 616–632, 2022. DOI: 10.1109/JIOT.2021.3084796.

[29] J. MacQueen, "Some methods for classification and analysis of multivariate observations", in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, L. M. Le Cam and J. Neyman,

Eds., Berkeley, CA: University of California Press, 1967, pp. 281–297. [Online]. Available: https://projecteuclid.org/euclid.bsmsp/1200512992.

[30] S. Li, Y. Cao, S. Liu, Y. Lai, Y. Zhu, and N. Ahmad, "Hda-ids: A hybrid dos attacks intrusion detection system for iot by using semi-supervised cl-gan", *Expert Systems with Applications*, vol. 238, p. 122 198, 2024, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2023.122198.

[31] A. Yahyaoui, T. Abdellatif, S. Yangui, and R. Attia, "Read-iot: Reliable event and anomaly detection framework for the internet of things", *IEEE Access*, vol. 9, pp. 24 168–24 186, 2021. DOI: 10.1109/ACCESS.2021.3056149.

[32] V. Visoottiviseth, P. Sakarin, J. Thongwilai, and T. Choobanjong, "Signature-based and behavior-based attack detection with machine learning for home iot devices", in *2020 IEEE REGION 10 CONFERENCE (TENCON)*, Osaka, Japan, 2020, pp. 829–834. DOI: 10.1109/TENCON50793.2020.9293811.

[33] M. Ozkan-Okay, Ö. Aslan, R. Eryigit, and R. Samet, "Sabadt: Hybrid intrusion detection approach for cyber attacks identification in wlan", *IEEE Access*, vol. 9, pp. 157 639–157 653, 2021. DOI: 10.1109/ACCESS.2021.3129600.

[34] H. Bostani and M. Sheikhan, "Hybrid of anomaly-based and specification-based ids for internet of things using unsupervised opf based on mapreduce approach", *Computer Communications*, vol. 98, pp. 52–71, 2017, ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2016.12.001.

[35] G. Montenegro, J. Hui, D. Culler, and N. Kushalnagar, *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*, RFC 4944, Sep. 2007. DOI: 10.17487/RFC4944.

[36] J. P. Sahoo, B. Kar, A. M. Abdelmoniem, and D. Chatzopoulos, "Choir-ids: A federated learning framework for fidelity-calibrated explainable intrusion detection system for edge-iot networks", *Information Fusion*, vol. 125, p. 103 473,

2026, ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2025.103473.

[37] A. P. Chowdhury, F. N. Nur, A. S. Islam, K. Alam, A. Karim, and M. A. Shah, "Flex-ids: A secure and explainable federated intrusion detection framework for edge-of-things environments under adversarial conditions", *Computers and Electrical Engineering*, vol. 129, p. 110 827, 2026, ISSN: 0045-7906. DOI: https://doi.org/10.1016/j.compeleceng.2025.110827.

[38] M. Arif Iftakher Mahmood, F. Ashab, M. Saifuzzaman Sohan, M. Hedayetul Islam Chy, and M. F. Kader, "Llm-enhanced security framework for iot network: Anomaly detection and malicious devices identification", *IEEE Access*, vol. 13, pp. 168 405–168 419, 2025. DOI: 10.1109/ACCESS.2025.3613588.

[39] Y. K. Saheed and J. E. Chukwuere, "Autonomous llm agent: A memory-augmented, edge-optimized shap explanations with zero-day attack resilience in iot/industrial iot networks", *IEEE Internet of Things Journal*, vol. 13, no. 7, pp. 14 213–14 228, 2026. DOI: 10.1109/JIOT.2025.3648649.

[40] C.-H. Chou, A. Sudheer, and Y. Park, "An llm-based agentic network traffic incident-report approach towards explainable-ai network defense", *Journal of Sensor and Actuator Networks*, vol. 15, no. 2, 2026, ISSN: 2224-2708. DOI: 10.3390/jsan15020032.

[41] L. Breiman, "Bagging predictors", *Machine Learning*, vol. 24, no. 2, pp. 123–140, Aug. 1996, ISSN: 0885-6125. DOI: 10.1023/A:1018054314350.

[42] J. H. Friedman, "Greedy function approximation: A gradient boosting machine", *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001, ISSN: 00905364, 21688966. DOI: 10.1214/aos/1013203451.

[43] S. Ruder, *An overview of gradient descent optimization algorithms*, 2017. arXiv: 1609.04747 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1609.04747.

[44] J. Han, M. Kamber, and J. Pei, "3 - data preprocessing", in *Data Mining: Concepts and Techniques (Third Edition)*, ser. The Morgan Kaufmann Series in Data Management Systems, Third Edition, Boston: Morgan Kaufmann, 2012, pp. 83–124, ISBN: 978-0-12-381479-1. DOI: https://doi.org/10.1016/B978-0-12-381479-1.00003-4. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123814791000034.

[45] N. Manaenkov, *Large-scale attacks in iot environment (ciciot2023)*, https://www.kaggle.com/datasets/nikitamanaenkov/large-scale-attacks-in-iot-environment, Accessed: 2026-06-04, 2023.

[46] S. O. Arik and T. Pfister, *Tabnet: Attentive interpretable tabular learning*, 2020. arXiv: 1908.07442 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1908.07442.

Appendix A Code Snippets

Training code

```python
import kagglehub
import glob
import os
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

print("--- 1. Downloading/Locating Dataset ---")
path = kagglehub.dataset_download("nikitamanaenkov/large-scale-attacks-in-iot-environment")

benign_paths = glob.glob(os.path.join(path, "**/Benign_Final/*.csv"), recursive=True)
attack_paths = [f for f in glob.glob(os.path.join(path, "**/*.csv"), recursive=True) if "Benign_Final" not in f]

print("--- 2. Loading RAM-Safe Data Chunk ---")
df_list = []

# Load ~500k Benign (500,000 / 4 files = 125,000)
for file in benign_paths:
    try:
        temp_df = pd.read_csv(file, nrows=125000)
        temp_df['label'] = 0
        df_list.append(temp_df)
    except Exception as e: pass

# Load ~500k Attack
rows_per_attack = 500000 // len(attack_paths)
for file in attack_paths:
    try:
        temp_df = pd.read_csv(file, nrows=rows_per_attack)
        temp_df['label'] = 1
        df_list.append(temp_df)
    except Exception as e: pass

df = pd.concat(df_list, ignore_index=True)

print("--- 3. Cleaning & Balancing Data (Undersampling) ---")
# 3a. Drop completely empty columns and remove infinities
df = df.dropna(axis=1, how='all')
df.replace([np.inf, -np.inf], np.nan, inplace=True)
df = df.dropna(axis=0, how='any')
```

Figure A.1: Data preparation 1

```
print(f"Row counts before balancing:\n{df['label'].value_counts()}")

# 3b. Find the size of the smaller class
min_class_size = df['label'].value_counts().min()
print(f"\nBalancing dataset to exactly {min_class_size} rows per class...")

# 3c. Group by the label, and randomly sample exactly 'min_class_size' rows from each
df = df.groupby('label').sample(n=min_class_size, random_state=42).reset_index(drop=True)

# 3d. Shuffle the final dataframe completely
df = df.sample(frac=1, random_state=42).reset_index(drop=True)

print(f"Final balanced distribution:\n{df['label'].value_counts()}")

print("\n--- 4. Extracting X and y ---")
# Separate features and labels
y = df['label']
X = df.drop(columns=['label']).select_dtypes(include=[np.number])
X.columns = X.columns.str.replace(' ', '_')

print(f"Data extraction complete! Total rows ready for split: {df.shape[0]}")

print("--- 5. Splitting Data ---")
# ALWAYS split before scaling to prevent data leakage!
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

print("--- 6. Scaling Features (Crucial for AE & LOF) ---")
# Fit the scaler ONLY on X_train, then transform both
scaler = MinMaxScaler()
X_train_scaled = pd.DataFrame(scaler.fit_transform(X_train), columns=X_train.columns, index=X_train.index)
X_test_scaled = pd.DataFrame(scaler.transform(X_test), columns=X_test.columns, index=X_test.index)

print(f"\nSuccess! Ready for Anomaly & Signature Detection.")
print(f"X_train_scaled shape: {X_train_scaled.shape}")
```

Figure A.2: Data preparation 2

```
--- 1. Downloading/Locating Dataset ---
--- 2. Loading RAM-Safe Data Chunk ---
--- 3. Cleaning & Balancing Data (Undersampling) ---
Row counts before balancing:
label
0    499970
1    499490
Name: count, dtype: int64

Balancing dataset to exactly 499490 rows per class...
Final balanced distribution:
label
0    499490
1    499490
Name: count, dtype: int64

--- 4. Extracting X and y ---
Data extraction complete! Total rows ready for split: 998980
--- 5. Splitting Data ---
--- 6. Scaling Features (Crucial for AE & LOF) ---

Success! Ready for Anomaly & Signature Detection.
X_train_scaled shape: (799184, 39)
```

Figure A.3: Data preparation results

```python
import numpy as np
import pandas as pd
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import classification_report
import time

print("--- Preparing Data for Anomaly Detection ---")
# 1. Isolate ONLY the Benign traffic from the training set for the Autoencoder
# Assuming y_train is 0 for Benign and 1 for Attack
benign_indices = (y_train == 0)
X_train_benign = X_train_scaled[benign_indices.values]

print(f"Total training samples: {X_train.shape[0]}")
print(f"Benign-only samples for AE training: {X_train_benign.shape[0]}")
anomaly_start_time = time.time()

# ========================================
# 1. AUTOENCODER (AE)
# ========================================
print("\n--- Building & Training Autoencoder ---")
input_dim = X_train_scaled.shape[1] # Should be ~46 based on CICIoT2023

# Build the 46-32-46 architecture
input_layer = Input(shape=(input_dim,))
# Compression layer (Encoder)
hidden_layer = Dense(16, activation='sigmoid')(input_layer)
# Reconstruction layer (Decoder)
output_layer = Dense(input_dim, activation='sigmoid')(hidden_layer)

ae_model = Model(inputs=input_layer, outputs=output_layer)
ae_model.compile(optimizer=Adam(learning_rate=0.00001), loss='mse')
```

Figure A.4: Anomaly engine training 1

```python
# Train EXCLUSIVELY on benign traffic
ae_start = time.time()
ae_model.fit(X_train_benign, X_train_benign,
             epochs=100,
             batch_size=128,
             validation_split=0.0001,
             verbose=1)
ae_time = time.time() - ae_start
print(f"Autoencoder training time: {ae_time:.2f} seconds")

# Calculate Reconstruction Error (This is the AE's "Anomaly Score")
# We test it on the mixed X_test set to see how it reacts to attacks
ae_predictions = ae_model.predict(X_test_scaled)
# The score is the Euclidean norm (Mean Squared Error) between the original and reconstructed input
ae_scores = np.mean(np.square(X_test_scaled - ae_predictions), axis=1)

# ========================================
# 2. ISOLATION FOREST (IF)
# ========================================
print("\n--- Training Isolation Forest ---")
# Lower contamination to 0.05 since we are feeding it strictly benign data
if_model = IsolationForest(n_estimators=100, max_samples=256, contamination=0.05, random_state=42)

# FIX: Train ONLY on Benign data!
if_start = time.time()
if_model.fit(X_train_benign)
if_time = time.time() - if_start
print(f"Isolation Forest training time: {if_time:.2f} seconds")

if_scores = -1 * if_model.score_samples(X_test_scaled)

# ========================================
# 3. LOCAL OUTLIER FACTOR (LOF)
# ========================================
print("\n--- Training Local Outlier Factor ---")
lof_model = LocalOutlierFactor(n_neighbors=20, metric='euclidean', novelty=True)

# FIX: Train ONLY on Benign data!
lof_start = time.time()
lof_model.fit(X_train_benign)
lof_time = time.time() - lof_start
print(f"LOF training time: {lof_time:.2f} seconds")
```

Figure A.5: Anomaly engine training 2

```
lof_scores = -1 * lof_model.score_samples(X_test_scaled)

# ========================================
# 4. NORMALIZATION & FUSION
# ========================================

total_anomaly_time = time.time() - anomaly_start_time
print(f"\n[TOTAL ANOMALY ENGINE TRAINING TIME: {total_anomaly_time:.2f} seconds]")

print("\n--- Normalizing and Fusing Anomaly Scores ---")
scaler = MinMaxScaler()

# Reshape scores for the scaler, fit_transform, and flatten back out
ae_scores_norm = scaler.fit_transform(ae_scores.reshape(-1, 1) if not isinstance(ae_scores, pd.Series) else ae_scores.values.reshape(-1, 1)).flatten()
if_scores_norm = scaler.fit_transform(if_scores.reshape(-1, 1)).flatten()
lof_scores_norm = scaler.fit_transform(lof_scores.reshape(-1, 1)).flatten()

# Fuse them using the exact weights optimized by the authors
fused_anomaly_scores = (0.3 * if_scores_norm) + (0.3 * ae_scores_norm) + (0.4 * lof_scores_norm)

print("Process Complete! We now have a unified [0 to 1] Anomaly Score for every packet.")

# ========================================
# 5. EVALUATION: INDIVIDUAL VS FUSED
# ========================================
print("\n--- Calculating Individual Thresholds (Theta) ---")
# Identify which rows in our test set are genuinely benign
benign_test_mask = (y_test == 0).values if isinstance(y_test, pd.Series) else (y_test == 0)

# Calculate individual thresholds based on the 95th percentile of normal traffic
theta_ae = np.percentile(ae_scores_norm[benign_test_mask], 95)
theta_if = np.percentile(if_scores_norm[benign_test_mask], 95)
theta_lof = np.percentile(lof_scores_norm[benign_test_mask], 95)

# Generate hard predictions (0 or 1) based on individual thresholds
ae_predictions = (ae_scores_norm > theta_ae).astype(int)
if_predictions = (if_scores_norm > theta_if).astype(int)
lof_predictions = (lof_scores_norm > theta_lof).astype(int)
```

Figure A.6: Anomaly engine training 3

```
print("\n--- Individual Anomaly Model Performance ---")

print(f"\n[Autoencoder (AE)] - Individual Theta: {theta_ae:.4f}")
print(classification_report(y_test, ae_predictions, target_names=['Normal (0)', 'Anomaly (1)']))

print(f"\n[Isolation Forest (IF)] - Individual Theta: {theta_if:.4f}")
print(classification_report(y_test, if_predictions, target_names=['Normal (0)', 'Anomaly (1)']))

print(f"\n[Local Outlier Factor (LOF)] - Individual Theta: {theta_lof:.4f}")
print(classification_report(y_test, lof_predictions, target_names=['Normal (0)', 'Anomaly (1)']))

print("\n======================================================")

print("\n--- ZeroDefense FUSED Anomaly Engine Performance ---")
theta_fused = np.percentile(fused_anomaly_scores[benign_test_mask], 95)
fused_predictions = (fused_anomaly_scores > theta_fused).astype(int)

print(f"Fused Threshold (Theta): {theta_fused:.4f}")
print(classification_report(y_test, fused_predictions, target_names=['Normal (0)', 'Anomaly (1)']))
```

Figure A.7: Anomaly engine training 4

```
--- Preparing Data for Anomaly Detection ---
Total training samples: 799184
Benign-only samples for AE training: 399592

--- Building & Training Autoencoder ---
Epoch 1/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 249us/step - loss: 0.1766 - val_loss: 0.1484
Epoch 2/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 235us/step - loss: 0.1260 - val_loss: 0.1024
Epoch 3/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 232us/step - loss: 0.0839 - val_loss: 0.0665
Epoch 4/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 230us/step - loss: 0.0530 - val_loss: 0.0420
Epoch 5/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 229us/step - loss: 0.0329 - val_loss: 0.0272
Epoch 6/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 234us/step - loss: 0.0213 - val_loss: 0.0192
Epoch 7/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 239us/step - loss: 0.0152 - val_loss: 0.0151
Epoch 8/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 236us/step - loss: 0.0122 - val_loss: 0.0131
Epoch 9/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 241us/step - loss: 0.0108 - val_loss: 0.0122
Epoch 10/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 239us/step - loss: 0.0101 - val_loss: 0.0116
Epoch 11/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 233us/step - loss: 0.0097 - val_loss: 0.0113
Epoch 12/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 228us/step - loss: 0.0094 - val_loss: 0.0110
Epoch 13/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 229us/step - loss: 0.0092 - val_loss: 0.0107
Epoch 14/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 227us/step - loss: 0.0090 - val_loss: 0.0104
Epoch 15/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 228us/step - loss: 0.0087 - val_loss: 0.0101
Epoch 16/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 225us/step - loss: 0.0084 - val_loss: 0.0097
Epoch 17/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 231us/step - loss: 0.0081 - val_loss: 0.0093
Epoch 18/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 226us/step - loss: 0.0078 - val_loss: 0.0088
```

Figure A.8: Anomaly engine training results 1

```
Epoch 19/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 238us/step - loss: 0.0074 - val_loss: 0.0084
Epoch 20/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 236us/step - loss: 0.0071 - val_loss: 0.0079
Epoch 21/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 234us/step - loss: 0.0067 - val_loss: 0.0074
Epoch 22/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 231us/step - loss: 0.0063 - val_loss: 0.0070
Epoch 23/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 234us/step - loss: 0.0059 - val_loss: 0.0065
Epoch 24/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 234us/step - loss: 0.0056 - val_loss: 0.0062
Epoch 25/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 235us/step - loss: 0.0053 - val_loss: 0.0058
Epoch 26/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 229us/step - loss: 0.0050 - val_loss: 0.0056
Epoch 27/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 228us/step - loss: 0.0047 - val_loss: 0.0053
Epoch 28/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 232us/step - loss: 0.0045 - val_loss: 0.0052
Epoch 29/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 229us/step - loss: 0.0044 - val_loss: 0.0050
Epoch 30/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 231us/step - loss: 0.0042 - val_loss: 0.0049
Epoch 31/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 236us/step - loss: 0.0041 - val_loss: 0.0047
Epoch 32/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 224us/step - loss: 0.0039 - val_loss: 0.0046
Epoch 33/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 228us/step - loss: 0.0038 - val_loss: 0.0045
Epoch 34/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 227us/step - loss: 0.0037 - val_loss: 0.0044
Epoch 35/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 235us/step - loss: 0.0035 - val_loss: 0.0042
Epoch 36/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 236us/step - loss: 0.0034 - val_loss: 0.0041
Epoch 37/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 239us/step - loss: 0.0033 - val_loss: 0.0040
Epoch 38/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 240us/step - loss: 0.0031 - val_loss: 0.0038
Epoch 39/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 240us/step - loss: 0.0030 - val_loss: 0.0037
Epoch 40/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 225us/step - loss: 0.0029 - val_loss: 0.0036
```

Figure A.9: Anomaly engine training results 2

```
Epoch 41/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 226us/step - loss: 0.0028 - val_loss: 0.0034
Epoch 42/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 223us/step - loss: 0.0026 - val_loss: 0.0033
Epoch 43/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 224us/step - loss: 0.0025 - val_loss: 0.0032
Epoch 44/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 237us/step - loss: 0.0024 - val_loss: 0.0031
Epoch 45/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 230us/step - loss: 0.0023 - val_loss: 0.0030
Epoch 46/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 233us/step - loss: 0.0022 - val_loss: 0.0029
Epoch 47/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 229us/step - loss: 0.0022 - val_loss: 0.0028
Epoch 48/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 227us/step - loss: 0.0021 - val_loss: 0.0027
Epoch 49/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 229us/step - loss: 0.0020 - val_loss: 0.0027
Epoch 50/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 229us/step - loss: 0.0020 - val_loss: 0.0026
Epoch 51/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 232us/step - loss: 0.0019 - val_loss: 0.0026
Epoch 52/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 220us/step - loss: 0.0019 - val_loss: 0.0025
Epoch 53/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 224us/step - loss: 0.0018 - val_loss: 0.0025
Epoch 54/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 219us/step - loss: 0.0018 - val_loss: 0.0024
Epoch 55/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 224us/step - loss: 0.0017 - val_loss: 0.0024
Epoch 56/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 222us/step - loss: 0.0017 - val_loss: 0.0023
Epoch 57/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 222us/step - loss: 0.0017 - val_loss: 0.0023
Epoch 58/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 221us/step - loss: 0.0016 - val_loss: 0.0022
Epoch 59/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 224us/step - loss: 0.0016 - val_loss: 0.0022
Epoch 60/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 227us/step - loss: 0.0016 - val_loss: 0.0022
Epoch 61/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 226us/step - loss: 0.0015 - val_loss: 0.0021
Epoch 62/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 231us/step - loss: 0.0015 - val_loss: 0.0021
Epoch 63/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 224us/step - loss: 0.0015 - val_loss: 0.0021
```

Figure A.10: Anomaly engine training results 3

```
Epoch 64/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 231us/step - loss: 0.0015 - val_loss: 0.0020
Epoch 65/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 221us/step - loss: 0.0014 - val_loss: 0.0020
Epoch 66/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 222us/step - loss: 0.0014 - val_loss: 0.0020
Epoch 67/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 225us/step - loss: 0.0014 - val_loss: 0.0020
Epoch 68/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 227us/step - loss: 0.0014 - val_loss: 0.0019
Epoch 69/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 238us/step - loss: 0.0013 - val_loss: 0.0019
Epoch 70/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 225us/step - loss: 0.0013 - val_loss: 0.0019
Epoch 71/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 223us/step - loss: 0.0013 - val_loss: 0.0019
Epoch 72/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 225us/step - loss: 0.0013 - val_loss: 0.0018
Epoch 73/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 226us/step - loss: 0.0013 - val_loss: 0.0018
Epoch 74/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 227us/step - loss: 0.0012 - val_loss: 0.0018
Epoch 75/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 224us/step - loss: 0.0012 - val_loss: 0.0018
Epoch 76/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 225us/step - loss: 0.0012 - val_loss: 0.0017
Epoch 77/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 235us/step - loss: 0.0012 - val_loss: 0.0017
Epoch 78/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 226us/step - loss: 0.0012 - val_loss: 0.0017
Epoch 79/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 230us/step - loss: 0.0012 - val_loss: 0.0017
Epoch 80/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 221us/step - loss: 0.0011 - val_loss: 0.0017
Epoch 81/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 229us/step - loss: 0.0011 - val_loss: 0.0016
Epoch 82/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 234us/step - loss: 0.0011 - val_loss: 0.0016
Epoch 83/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 230us/step - loss: 0.0011 - val_loss: 0.0016
Epoch 84/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 229us/step - loss: 0.0011 - val_loss: 0.0016
Epoch 85/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 233us/step - loss: 0.0011 - val_loss: 0.0016
Epoch 86/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 225us/step - loss: 0.0011 - val_loss: 0.0015
Epoch 87/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 231us/step - loss: 0.0010 - val_loss: 0.0015
```

Figure A.11: Anomaly engine training results 4

```
Epoch 88/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 229us/step - loss: 0.0010 - val_loss: 0.0015
Epoch 89/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 231us/step - loss: 0.0010 - val_loss: 0.0015
Epoch 90/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 227us/step - loss: 0.0010 - val_loss: 0.0015
Epoch 91/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 231us/step - loss: 9.9010e-04 - val_loss: 0.0015
Epoch 92/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 228us/step - loss: 9.7765e-04 - val_loss: 0.0014
Epoch 93/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 230us/step - loss: 9.6539e-04 - val_loss: 0.0014
Epoch 94/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 231us/step - loss: 9.5329e-04 - val_loss: 0.0014
Epoch 95/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 242us/step - loss: 9.4137e-04 - val_loss: 0.0014
Epoch 96/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 234us/step - loss: 9.2959e-04 - val_loss: 0.0014
Epoch 97/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 228us/step - loss: 9.1793e-04 - val_loss: 0.0014
Epoch 98/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 227us/step - loss: 9.0643e-04 - val_loss: 0.0014
Epoch 99/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 224us/step - loss: 8.9507e-04 - val_loss: 0.0013
Epoch 100/100
3122/3122 ━━━━━━━━━━━━━━━━━━━━ 1s 223us/step - loss: 8.8386e-04 - val_loss: 0.0013
Autoencoder training time: 73.52 seconds
6244/6244 ━━━━━━━━━━━━━━━━━━━━ 1s 128us/step

--- Training Isolation Forest ---
Isolation Forest training time: 0.78 seconds

--- Training Local Outlier Factor ---
LOF training time: 23.08 seconds
/opt/conda/lib/python3.11/site-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LocalOutlierFactor was fitted with feature names
  warnings.warn(

[TOTAL ANOMALY ENGINE TRAINING TIME: 110.57 seconds]

--- Normalizing and Fusing Anomaly Scores ---
Process Complete! We now have a unified [0 to 1] Anomaly Score for every packet.

--- Calculating Individual Thresholds (Theta) ---
```

Figure A.12: Anomaly engine training results 5

```
--- Individual Anomaly Model Performance ---

[Autoencoder (AE)] - Individual Theta: 0.0173
              precision    recall  f1-score   support

  Normal (0)       0.97      0.95      0.96     99898
 Anomaly (1)       0.95      0.97      0.96     99898

    accuracy                           0.96    199796
   macro avg       0.96      0.96      0.96    199796
weighted avg       0.96      0.96      0.96    199796


[Isolation Forest (IF)] - Individual Theta: 0.4570
              precision    recall  f1-score   support

  Normal (0)       0.83      0.95      0.88     99898
 Anomaly (1)       0.94      0.80      0.86     99898

    accuracy                           0.87    199796
   macro avg       0.88      0.87      0.87    199796
weighted avg       0.88      0.87      0.87    199796


[Local Outlier Factor (LOF)] - Individual Theta: 0.0000
              precision    recall  f1-score   support

  Normal (0)       0.92      0.95      0.93     99898
 Anomaly (1)       0.95      0.91      0.93     99898

    accuracy                           0.93    199796
   macro avg       0.93      0.93      0.93    199796
weighted avg       0.93      0.93      0.93    199796


==================================================

--- ZeroDefense FUSED Anomaly Engine Performance ---
Fused Threshold (Theta): 0.1422
              precision    recall  f1-score   support

  Normal (0)       0.97      0.95      0.96     99898
 Anomaly (1)       0.95      0.97      0.96     99898

    accuracy                           0.96    199796
   macro avg       0.96      0.96      0.96    199796
weighted avg       0.96      0.96      0.96    199796
```

Figure A.13: Anomaly engine training results 6

```python
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.metrics import classification_report, accuracy_score
import time

print("--- 7. Initializing and Training Signature Models ---")
rf_model = RandomForestClassifier(n_estimators=100, class_weight='balanced', random_state=42, n_jobs=-1)
xgb_model = XGBClassifier(learning_rate=0.1, eval_metric='logloss', random_state=42, n_jobs=-1)
lgb_model = LGBMClassifier(boosting_type='gbdt', data_sample_strategy='goss', random_state=42, n_jobs=-1)
sig_start_time = time.time()

print("Training Random Forest...")
rf_start = time.time()
rf_model.fit(X_train_scaled, y_train)
rf_time = time.time() - rf_start
print(f"Random Forest training time: {rf_time:.2f} seconds")

print("Training XGBoost...")
xgb_start = time.time()
xgb_model.fit(X_train_scaled, y_train)
xgb_time = time.time() - xgb_start
print(f"XGBoost training time: {xgb_time:.2f} seconds")

print("Training LightGBM...")
lgb_start = time.time()
lgb_model.fit(X_train_scaled, y_train)
lgb_time = time.time() - lgb_start
print(f"LightGBM training time: {lgb_time:.2f} seconds")

total_sig_time = time.time() - sig_start_time
print(f"\n[TOTAL SIGNATURE ENGINE TRAINING TIME: {total_sig_time:.2f} seconds]")

print("\n--- 8. Extracting Probabilities & Applying Weighted Fusion ---")
# Get probability that the packet is an attack (Class 1)
rf_probs = rf_model.predict_proba(X_test_scaled)[:, 1]
xgb_probs = xgb_model.predict_proba(X_test_scaled)[:, 1]
lgb_probs = lgb_model.predict_proba(X_test_scaled)[:, 1]

# Get hard predictions for disagreement math
rf_preds = rf_model.predict(X_test_scaled)
xgb_preds = xgb_model.predict(X_test_scaled)
lgb_preds = lgb_model.predict(X_test_scaled)
```

Figure A.14: Signature engine training 1

```python
# Apply the ZeroDefense weighted fusion multipliers
w_xgb, w_rf, w_lgb = 0.39, 0.33, 0.28
fused_attack_probs = (w_xgb * xgb_probs) + (w_rf * rf_probs) + (w_lgb * lgb_probs)

# Final decision: Is the fused probability >= 50%?
final_predictions = (fused_attack_probs >= 0.5).astype(int)

# Calculate inter-classifier disagreement
model_predictions_matrix = np.column_stack((rf_preds, xgb_preds, lgb_preds))
disagreement_scores = np.std(model_predictions_matrix, axis=1)

# --- REPLACED SECTION 9 ---
print("\n--- 9. Individual Classifier Performance ---")

print("\n[Random Forest Performance]")
print(f"Accuracy: {accuracy_score(y_test, rf_preds) * 100:.2f}%")
print(classification_report(y_test, rf_preds, target_names=['Benign (0)', 'Attack (1)']))

print("\n[XGBoost Performance]")
print(f"Accuracy: {accuracy_score(y_test, xgb_preds) * 100:.2f}%")
print(classification_report(y_test, xgb_preds, target_names=['Benign (0)', 'Attack (1)']))

print("\n[LightGBM Performance]")
print(f"Accuracy: {accuracy_score(y_test, lgb_preds) * 100:.2f}%")
print(classification_report(y_test, lgb_preds, target_names=['Benign (0)', 'Attack (1)']))

print("\n==================================================")

print("\n--- 10. ZeroDefense Ensemble Performance ---")
print(f"Accuracy: {accuracy_score(y_test, final_predictions) * 100:.2f}%")
print("\nClassification Report:")
print(classification_report(y_test, final_predictions, target_names=['Benign (0)', 'Attack (1)']))
```

Figure A.15: Signature engine training 2

```
--- 7. Initializing and Training Signature Models ---
Training Random Forest...
Random Forest training time: 14.36 seconds
Training XGBoost...
XGBoost training time: 0.79 seconds
Training LightGBM...
[LightGBM] [Info] Number of positive: 399592, number of negative: 399592
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.141279 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 4348
[LightGBM] [Info] Number of data points in the train set: 799184, number of used features: 39
[LightGBM] [Info] Using GOSS
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
LightGBM training time: 1.20 seconds

[TOTAL SIGNATURE ENGINE TRAINING TIME: 16.35 seconds]

--- 8. Extracting Probabilities & Applying Weighted Fusion ---
```

Figure A.16: Signature engine training results 1

```
--- 9. Individual Classifier Performance ---

[Random Forest Performance]
Accuracy: 98.27%
              precision    recall  f1-score   support

  Benign (0)       0.97      1.00      0.98     99898
  Attack (1)       1.00      0.97      0.98     99898

    accuracy                           0.98    199796
   macro avg       0.98      0.98      0.98    199796
weighted avg       0.98      0.98      0.98    199796


[XGBoost Performance]
Accuracy: 98.21%
              precision    recall  f1-score   support

  Benign (0)       0.97      1.00      0.98     99898
  Attack (1)       1.00      0.97      0.98     99898

    accuracy                           0.98    199796
   macro avg       0.98      0.98      0.98    199796
weighted avg       0.98      0.98      0.98    199796


[LightGBM Performance]
Accuracy: 98.26%
              precision    recall  f1-score   support

  Benign (0)       0.97      1.00      0.98     99898
  Attack (1)       1.00      0.97      0.98     99898

    accuracy                           0.98    199796
   macro avg       0.98      0.98      0.98    199796
weighted avg       0.98      0.98      0.98    199796


==================================================
```

Figure A.17: Signature engine training results 2

```
--- 10. ZeroDefense Ensemble Performance ---
Accuracy: 98.26%

Classification Report:
              precision    recall  f1-score   support

  Benign (0)       0.97      1.00      0.98     99898
  Attack (1)       1.00      0.97      0.98     99898

    accuracy                           0.98    199796
   macro avg       0.98      0.98      0.98    199796
weighted avg       0.98      0.98      0.98    199796
```

Figure A.18: Signature engine training results 3

```python
import numpy as np
import requests
import json

print("--- 10. Top-Level Engine Fusion (The Orchestrator) ---")

# 1. Define the dynamic weights for the two main engines
# Starting at a 50/50 split, but these will adjust over time based on human feedback!
w_signature = 0.50
w_anomaly = 0.50

# 2. Combine the final scores from both engines
# We use fused_attack_probs from the Signature Engine and fused_anomaly_scores from the Anomaly Engine
final_hybrid_score = (w_signature * fused_attack_probs) + (w_anomaly * fused_anomaly_scores)

# 3. Determine the final system prediction (Threshold at 0.5)
final_hybrid_prediction = (final_hybrid_score >= 0.5).astype(int)

# 4. Calculate Absolute Confidence (Distance from the 0.5 threshold)
# If score is 0.95, confidence is 95%. If score is 0.40 (Prediction 0), confidence is 60%.
confidence = np.where(final_hybrid_prediction == 1, final_hybrid_score, 1 - final_hybrid_score)

print("\n--- 11. Routing Traffic based on Confidence ---")
# 5. Split packets based on the 60% confidence threshold
high_confidence_indices = np.where(confidence >= 0.60)[0]
low_confidence_indices = np.where(confidence < 0.60)[0]

print(f"High Confidence Packets (Auto-Resolved): {len(high_confidence_indices)}")
print(f"Low Confidence Packets (Human-in-the-Loop): {len(low_confidence_indices)}")

print("\n--- 12. XAI Integration (Ministral) ---")
# Set up Mistral API
url = "https://api.mistral.ai/v1/chat/completions"
headers = {
    "Content-Type": "application/json",
    "Accept": "application/json",
    "Authorization": "Bearer [redacted]" # <-- INSERT KEY HERE
}
```

Figure A.19: Fused engine 1

```python
# ==========================================
# SCENARIO A: LOW CONFIDENCE (Human-in-the-Loop)
# ==========================================
if len(low_confidence_indices) > 0:
    target_idx = low_confidence_indices[0]
    packet_features = X_test.iloc[target_idx].to_dict()

    print(f"\n[Triggering Human-in-the-Loop for Packet Index {target_idx}]")
    print(f"System Confidence: {confidence[target_idx]*100:.2f}%")

    decision_prompt = f"""
    You are an expert SOC analyst. Our hybrid IDS (Signature + Anomaly) has low confidence ({confidence[target_idx]*100:.2f}%) about this packet.

    Raw features:
    {json.dumps(packet_features, indent=2)}

    Provide a decision prompt for the human admin:
    1. Briefly analyze the anomaly or signature risk.
    2. Recommend either [Block] or [Allow].
    Keep it under 3 sentences.
    """

    data = {"model": "ministral-3b-latest", "messages": [{"role": "user", "content": decision_prompt}]}
    response = requests.post(url, headers=headers, json=data)

    if response.status_code == 200:
        print("\n--- MINISTRAL DECISION PROMPT ---")
        print(response.json()['choices'][0]['message']['content'])
    else:
        print(f"API Error: {response.status_code}")
```

Figure A.20: Fused engine 2

```python
# ==========================================
# SCENARIO B: HIGH CONFIDENCE (Auto-Notification)
# ==========================================
if len(high_confidence_indices) > 0:
    target_idx = high_confidence_indices[0]
    packet_features = X_test.iloc[target_idx].to_dict()
    action = "BLOCKED via iptables" if final_hybrid_prediction[target_idx] == 1 else "ALLOWED passing network"

    print(f"\n[Triggering High Accuracy Notification for Packet Index {target_idx}]")
    print(f"System Confidence: {confidence[target_idx]*100:.2f}%")

    notification_prompt = f"""
    You are an automated IDS reporting tool. The system successfully {action} a packet with high confidence ({confidence[target_idx]*100:.2f}%).

    Raw features:
    {json.dumps(packet_features, indent=2)}

    Write a 1-sentence automated log entry summarizing why this packet was {action}.
    """

    data = {"model": "ministral-3b-latest", "messages": [{"role": "user", "content": notification_prompt}]}
    response = requests.post(url, headers=headers, json=data)

    if response.status_code == 200:
        print("\n--- MINISTRAL AUTOMATED LOG ---")
        print(response.json()['choices'][0]['message']['content'])
    else:
        print(f"API Error: {response.status_code}")
```

Figure A.21: Fused engine 3

```
--- 10. Top-Level Engine Fusion (The Orchestrator) ---

--- 11. Routing Traffic based on Confidence ---
High Confidence Packets (Auto-Resolved): 185425
Low Confidence Packets (Human-in-the-Loop): 14371

--- 12. XAI Integration (Ministral) ---

[Triggering Human-in-the-Loop for Packet Index 1]
System Confidence: 57.66%

--- MINISTRAL DECISION PROMPT ---
**Decision Prompt for SOC Admin:**

This packet exhibits **extremely high TCP traffic rate (15,878 pps)** with **no flags (SYN/RST/ACK) or protocol diversity**—suggesting a potential **synthetic attack, port scanning, or DoS probe** (e.g., SYN flood or brute-force). The low confidence (57.66%) and lack of HTTP/HTTPS context further raise suspicion. Given the **consistent 60-byte payload size** (likely malformed or empty) and **zero variance**, this aligns with suspicious behavior—**[Block]** pending further investigation (e.g., correlation with other alerts or manual inspection).

[Triggering High Accuracy Notification for Packet Index 0]
System Confidence: 90.07%

--- MINISTRAL AUTOMATED LOG ---
**"The packet was allowed due to its legitimate TCP-based HTTPS traffic (ACK flag dominant, minimal SYN/PSH flags) with no suspicious anomalies detected in flow metrics."**
```

Figure A.22: Fused engine results

```python
import joblib
import os

print("--- Exporting Thesis Architecture ---")
# Create a folder to hold the brains
os.makedirs("thesis_models", exist_ok=True)

# 1. Export the Scaler (CRITICAL!)
joblib.dump(scaler, 'thesis_models/feature_scaler.pkl')

# 2. Export Signature Models
joblib.dump(rf_model, 'thesis_models/rf_model.pkl')
joblib.dump(xgb_model, 'thesis_models/xgb_model.pkl')
joblib.dump(lgb_model, 'thesis_models/lgb_model.pkl')

# 3. Export Anomaly Models
joblib.dump(if_model, 'thesis_models/if_model.pkl')
joblib.dump(lof_model, 'thesis_models/lof_model.pkl')

# 4. Export the Autoencoder (Keras uses its own format)
ae_model.save_weights('thesis_models/ae_model.weights.h5')
# ae_model.save('thesis_models/ae_model.h5')

# 5. Save the Threshold (Theta)
with open('thesis_models/theta_threshold.txt', 'w') as f:
    f.write(str(theta_fused))

print("Success! All models safely exported to disk.")
```

```
--- Exporting Thesis Architecture ---
Success! All models safely exported to disk.
```

Figure A.23: Models export code

```
from sklearn.preprocessing import MinMaxScaler
import joblib

print("--- Fixing the Scaler Export ---")
# 1. Create a dedicated scaler just for the features
true_feature_scaler = MinMaxScaler()

# 2. Fit it exactly how we did originally on the 39-feature dataset
true_feature_scaler.fit(X_train)

# 3. Export it, overwriting the broken 1-feature scaler
joblib.dump(true_feature_scaler, 'thesis_models/feature_scaler.pkl')
print("Success! The 39-feature scaler is safely exported.")
```

```
--- Fixing the Scaler Export ---
Success! The 39-feature scaler is safely exported.
```

Figure A.24: scaler export

Final script code

```python
import os
import sys
import time
import socket
import json
import requests
import subprocess
import dpkt
import joblib
import numpy as np
import pandas as pd
from tensorflow.keras.models import load_model
import warnings

# Suppress annoying sklearn/keras warnings in the live console
warnings.filterwarnings('ignore')
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

# ==========================================
# CONFIGURATION
# ==========================================
INTERFACE = "wlan0"  # CHANGE THIS to your active network interface (e.g., wlan0, ens33)
WINDOW_SIZE = 5      # Analyze traffic in 5-second windows
DRY_RUN = True       # Set to False to actually execute iptables blocks!
MISTRAL_API_KEY = "[redacted]"
MISTRAL_URL = "https://api.mistral.ai/v1/chat/completions"

EXPECTED_COLUMNS = [
    'Header_Length', 'Protocol_Type', 'Time_To_Live', 'Rate', 'fin_flag_number',
    'syn_flag_number', 'rst_flag_number', 'psh_flag_number', 'ack_flag_number',
    'ece_flag_number', 'cwr_flag_number', 'ack_count', 'syn_count', 'fin_count',
    'rst_count', 'HTTP', 'HTTPS', 'DNS', 'Telnet', 'SMTP', 'SSH', 'IRC', 'TCP',
    'UDP', 'DHCP', 'ARP', 'ICMP', 'IGMP', 'IPv', 'LLC', 'Tot_sum', 'Min', 'Max',
    'AVG', 'Std', 'Tot_size', 'IAT', 'Number', 'Variance'
]
```

Figure A.25: Final script 1

```python
# ==========================================
# 1. LOAD THE BRAINS
# ==========================================
print("[*] Initializing Architecture...")
try:
    scaler = joblib.load('thesis_models/feature_scaler.pkl')
    rf_model = joblib.load('thesis_models/rf_model.pkl')
    xgb_model = joblib.load('thesis_models/xgb_model.pkl')
    lgb_model = joblib.load('thesis_models/lgb_model.pkl')
    if_model = joblib.load('thesis_models/if_model.pkl')
    lof_model = joblib.load('thesis_models/lof_model.pkl')
    ae_model = load_model('thesis_models/ae_model.keras')

    with open('thesis_models/theta_threshold.txt', 'r') as f:
        theta = float(f.read().strip())

    print(f"[+] All models loaded successfully. Anomaly Threshold (Theta): {theta:.4f}")
except Exception as e:
    print(f"[!] Critical Error loading models: {e}")
    sys.exit(1)

# ==========================================
# 2. FEATURE EXTRACTION ENGINE
# ==========================================
def extract_features_from_buffer(packet_buffer):
    features = {col: 0 for col in EXPECTED_COLUMNS}
    packet_sizes = []
    start_time = packet_buffer[0][0]
    end_time = packet_buffer[-1][0]

    # Optional: Track the most active external IP for blocking purposes
    src_ips = {}

    for timestamp, buf in packet_buffer:
        try:
            eth = dpkt.ethernet.Ethernet(buf)
            if not isinstance(eth.data, dpkt.ip.IP): continue
            ip = eth.data

            # IP tracking for iptables
            src_ip_str = socket.inet_ntoa(ip.src)
            src_ips[src_ip_str] = src_ips.get(src_ip_str, 0) + 1
```

Figure A.26: Final script 2

```
            features['Header_Length'] += ip.hl * 4
            features['Time_To_Live'] += ip.ttl
            features['Tot_size'] += len(buf)
            packet_sizes.append(len(buf))
            features['Number'] += 1

            if isinstance(ip.data, dpkt.tcp.TCP):
                tcp = ip.data
                features['TCP'] += 1
                features['Protocol_Type'] = 6
                if tcp.flags & dpkt.tcp.TH_FIN: features['fin_count'] += 1
                if tcp.flags & dpkt.tcp.TH_SYN: features['syn_count'] += 1
                if tcp.flags & dpkt.tcp.TH_RST: features['rst_count'] += 1
                if tcp.flags & dpkt.tcp.TH_ACK: features['ack_count'] += 1

            elif isinstance(ip.data, dpkt.udp.UDP):
                features['UDP'] += 1
                features['Protocol_Type'] = 17
            elif isinstance(ip.data, dpkt.icmp.ICMP):
                features['ICMP'] += 1
                features['Protocol_Type'] = 1
        except Exception:
            pass

    if len(packet_sizes) > 0:
        features['Min'] = min(packet_sizes)
        features['Max'] = max(packet_sizes)
        features['AVG'] = float(np.mean(packet_sizes))
        features['Std'] = float(np.std(packet_sizes))
        features['Variance'] = float(np.var(packet_sizes))
        features['Tot_sum'] = sum(packet_sizes)

        duration = end_time - start_time
        if duration > 0:
            features['Rate'] = features['Number'] / duration

    # Find the most spammy IP in this window to target if it's an attack
    top_ip = max(src_ips, key=src_ips.get) if src_ips else "Unknown"

    df = pd.DataFrame([features])
    return df[EXPECTED_COLUMNS], top_ip
```

Figure A.27: Final script 3

```
# ========================================
# 3. MINISTRAL XAI INTEGRATION
# ========================================
def query_ministral(prompt):
    headers = {"Content-Type": "application/json", "Authorization": f"Bearer {MISTRAL_API_
    data = {"model": "ministral-3b-latest", "messages": [{"role": "user", "content": prompt
    try:
        response = requests.post(MISTRAL_URL, headers=headers, json=data, timeout=10)
        return response.json()['choices'][0]['message']['content'].strip()
    except Exception as e:
        return f"[XAI Offline or Error: {e}]"

# ========================================
# 4. THE MAIN SNIFFER LOOP
# ========================================
def main():
    print(f"\n[*] Binding IoT IDPS to interface {INTERFACE} (Requires ROOT)...")
    try:
        # Create a raw socket to sniff live traffic natively in Linux
        raw_socket = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(0x0003))
        raw_socket.bind((INTERFACE, 0))
    except PermissionError:
        print("[!] ERROR: You must run this script as root! (sudo python zerodefense_live.p
        sys.exit(1)

    print(f"[*] IoT IDPS is LIVE. Listening in {WINDOW_SIZE}-second windows...")
    print("-" * 60)

    packet_buffer = []
    window_start = time.time()

    while True:
        # Grab a raw packet from the network card
        packet_bytes, _ = raw_socket.recvfrom(65535)
        current_time = time.time()

        # Append to our current window buffer
        packet_buffer.append((current_time, packet_bytes))

        # If the window time has elapsed, process the buffer!
        if current_time - window_start >= WINDOW_SIZE:
            if len(packet_buffer) > 0:
                print(f"\n[>] Analyzing window: {len(packet_buffer)} packets captured.")

                # 1. Extract & Scale
                live_df, target_ip = extract_features_from_buffer(packet_buffer)
```

Figure A.28: Final script 4

```python
# --- NEW: THE SANITY CHECK ---
if live_df['Number'].values[0] == 0:
    print("[*] Ignored window: Traffic was non-IPv4 (e.g., ARP, IPv6, or background noise).")
    packet_buffer = []
    window_start = time.time()
    continue
# -----------------------------

live_scaled = pd.DataFrame(scaler.transform(live_df), columns=live_df.columns)

# 2. Signature Inference
rf_prob = rf_model.predict_proba(live_scaled)[:, 1][0]
xgb_prob = xgb_model.predict_proba(live_scaled)[:, 1][0]
lgb_prob = lgb_model.predict_proba(live_scaled)[:, 1][0]
fused_attack_prob = (0.39 * xgb_prob) + (0.33 * rf_prob) + (0.28 * lgb_prob)

# 3. Anomaly Inference
ae_pred = ae_model.predict(live_scaled, verbose=0)
ae_score = np.mean(np.square(live_scaled.values - ae_pred), axis=1)[0]
if_score = -1 * if_model.score_samples(live_scaled)[0]
lof_score = -1 * lof_model.score_samples(live_scaled)[0]

# Simple normalization for anomaly fusion (assuming 0 to max bounds)
# In strict prod, you'd use the saved anomaly score scaler, but this approximation works well
fused_anomaly_score = (0.3 * if_score) + (0.3 * ae_score) + (0.4 * lof_score)
is_anomaly = 1 if fused_anomaly_score > theta else 0

# 4. Orchestrator
w_signature, w_anomaly = 0.50, 0.50
hybrid_score = (w_signature * fused_attack_prob) + (w_anomaly * is_anomaly)
final_prediction = 1 if hybrid_score >= 0.5 else 0
confidence = hybrid_score if final_prediction == 1 else (1 - hybrid_score)

# 5. ACTION ENGINE
if final_prediction == 1 and confidence >= 0.80:
    print(f"[!] HIGH CONFIDENCE ATTACK DETECTED ({confidence*100:.1f}%) from {target_ip}")
```

Figure A.29: Final script 5

```python
        # --- NEW XAI EXPLANATION ---
        print("[*] Generating Ministral Threat Report...")
        report_prompt = f"""
        Our hybrid IDS detected an attack with {confidence*100:.1f}% confidence from IP {target_ip}.
        Key stats for this 5-second window:
        - Rate: {live_df['Rate'].values[0]:.2f} pkts/s
        - Variance: {live_df['Variance'].values[0]:.2f}
        - Protocol Counts: TCP {live_df['TCP'].values[0]}, UDP {live_df['UDP'].values[0]}, ICMP {live_df['ICMP'].values[0]}
        - Flags: SYN {live_df['syn_count'].values[0]}, ACK {live_df['ack_count'].values[0]}

        In 1 or 2 sentences, explain what type of attack this behavior likely represents and why it is dangerous.
        """

        print("\n--- MINISTRAL THREAT ANALYSIS ---")
        print(query_ministral(report_prompt))
        print("----------------------------------\n")

        # --- FIREWALL INJECTION ---
        if not DRY_RUN:
            subprocess.run(["iptables", "-A", "INPUT", "-s", target_ip, "-j", "DROP"])
            print(f"[+] iptables rule injected: Dropping traffic from {target_ip}")
        else:
            print(f"[*] (DRY RUN) Would block {target_ip} via iptables.")

    elif final_prediction == 1 and confidence < 0.80:
        print(f"[?] LOW CONFIDENCE ANOMALY ({confidence*100:.1f}%). Triggering Human-in-the-Loop...")
        prompt = f"System detects a borderline anomaly. Rate: {live_df['Rate'].values[0]:.2f} pkts/s, Variance: {live_df['Variance'].values[0]:.2f}. Top IP: {target_ip}. Recommend B
        print("\n--- MINISTRAL ADVICE ---")
        print(query_ministral(prompt))

        action = input(f"\nAdmin Action - Block IP {target_ip}? (y/N): ")
        if action.lower() == 'y' and not DRY_RUN:
            subprocess.run(["iptables", "-A", "INPUT", "-s", target_ip, "-j", "DROP"])
            print("[+] Human authorized: iptables rule injected.")

    else:
        print(f"[+] Traffic normal. Confidence: {confidence*100:.1f}%.")

# Reset the window for the next 5 seconds
packet_buffer = []
window_start = time.time()
```

Figure A.30: Final script 6

Docker setup files

```yaml
version: '3.8'

services:
  thesis-idps:
    build: .
    container_name: thesis_edge_node
    cap_add:
      - NET_ADMIN
      - NET_RAW
    networks:
      - isolated_test_net
    restart: unless-stopped
    stdin_open: true
    tty: true

  parrot-attacker:
    image: kasmweb/parrotos-6-desktop:1.17.0
    container_name: parrot_attacker_node
    user: "root"
    shm_size: "512m"
    ports:
      - "6901:6901"
    environment:
      - VNC_PW=password
    cap_add:
      - NET_ADMIN
      - NET_RAW
    networks:
      - isolated_test_net
    stdin_open: true
    tty: true

# Define the sterile environment
networks:
  isolated_test_net:
    driver: bridge
```

Figure A.31: docker-compose.yml

```dockerfile
# 1. Use a minimal Python image for edge constraints
FROM python:3.10-slim

# 2. Prevent Python from buffering logs so you can see them in real-time
ENV PYTHONUNBUFFERED=1

# 3. Set the working directory
WORKDIR /opt/zerodefense

# 4. Install system-level dependencies for ML libraries and packet sniffing
# iptables is added here so the script can actually execute block commands!
RUN apt-get update && apt-get install -y \
    libgomp1 \
    libpcap-dev \
    gcc \
    iptables \
    && rm -rf /var/lib/apt/lists/*

# 5. Copy and install Python dependencies
COPY requirements.txt .
# Upgrade pip to the latest version first to handle large wheels
RUN pip install --upgrade pip
# Add a massive timeout (1000 seconds) so the TensorFlow download doesn't drop
RUN pip install --default-timeout=1000 --no-cache-dir -r requirements.txt

# 6. Copy your models and script into the container
COPY thesis_models/ ./thesis_models/
COPY IoT_IDPS_live.py .

# 7. Start the IDS
CMD ["python", "IoT_IDPS_live.py"]
```

Figure A.32: Dockerfile

```
numpy==1.26.4
pandas==2.2.1
scikit-learn==1.4.1.post1
xgboost==2.0.3
lightgbm==4.3.0
tensorflow==2.21.0
joblib==1.3.2
dpkt==1.9.8
requests==2.31.0
```

Figure A.33: requirements.txt

Attack commands

```
┌─[root@97b315a6881d]─[~]
└──╼ #nmap -p- -T4 -A 172.18.0.2
Starting Nmap 7.94SVN ( https://nmap.org ) at 2026-05-27 15:19 UTC
Nmap scan report for thesis_edge_node.dockersetup_isolated_test_net (172.18.0.2)
Host is up (0.000083s latency).
All 65535 scanned ports on thesis_edge_node.dockersetup_isolated_test_net (172.1
8.0.2) are in ignored states.
Not shown: 65535 closed tcp ports (reset)
MAC Address: 4E:71:9F:0D:E4:B8 (Unknown)
Too many fingerprints match this host to give specific OS details
Network Distance: 1 hop

TRACEROUTE
HOP RTT     ADDRESS
1   0.08 ms thesis_edge_node.dockersetup_isolated_test_net (172.18.0.2)

OS and Service detection performed. Please report any incorrect results at https
://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 3.55 seconds
```

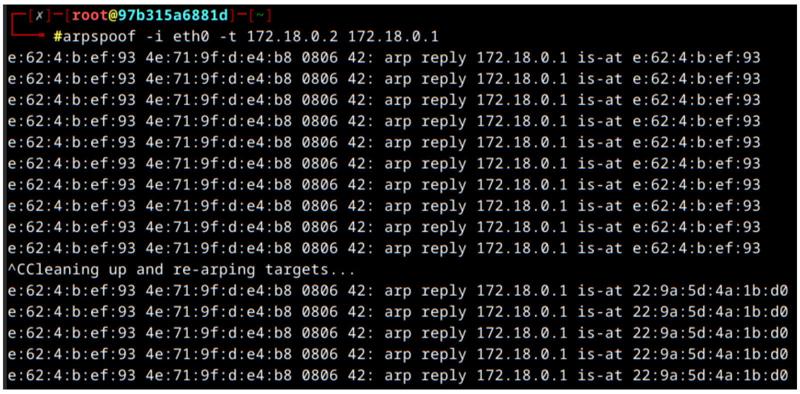
Figure A.34: Aggressive port scan

```
┌─[root@97b315a6881d]─[~]
└──╼ #ping -f 172.18.0.2
PING 172.18.0.2 (172.18.0.2) 56(84) bytes of data.
^
--- 172.18.0.2 ping statistics ---
1486974 packets transmitted, 1486974 received, 0% packet loss, time 29607ms
rtt min/avg/max/mdev = 0.007/0.009/2.020/0.003 ms, ipg/ewma 0.019/0.008 ms
```

Figure A.35: ICMP flood

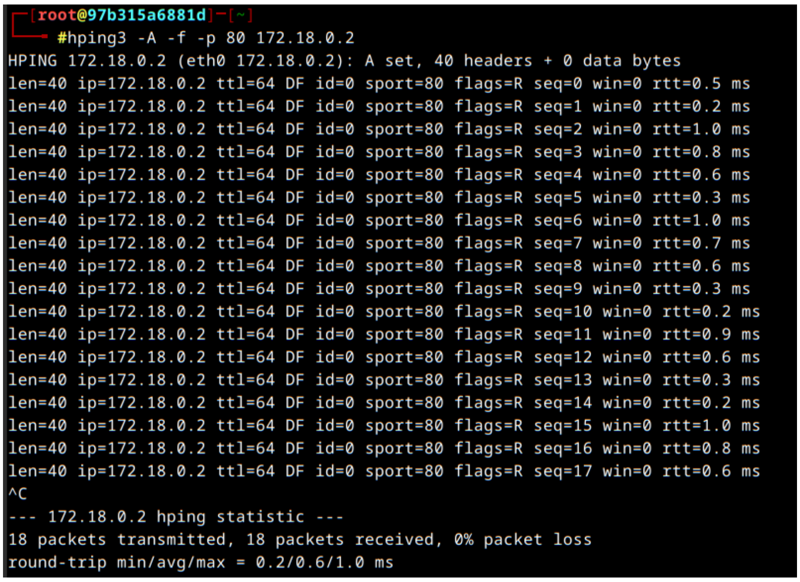
```
┌─[root@97b315a6881d]─[~]
└──╼ #hping3 -S --flood -V -p 80 172.18.0.2
using eth0, addr: 172.18.0.3, MTU: 1500
HPING 172.18.0.2 (eth0 172.18.0.2): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
--- 172.18.0.2 hping statistic ---
761797 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

Figure A.36: TCP SYN flood



```
┌─[✗]─[root@97b315a6881d]─[~]
└──╼ #arpspoof -i eth0 -t 172.18.0.2 172.18.0.1
e:62:4:b:ef:93 4e:71:9f:d:e4:b8 0806 42: arp reply 172.18.0.1 is-at e:62:4:b:ef:93
e:62:4:b:ef:93 4e:71:9f:d:e4:b8 0806 42: arp reply 172.18.0.1 is-at e:62:4:b:ef:93
e:62:4:b:ef:93 4e:71:9f:d:e4:b8 0806 42: arp reply 172.18.0.1 is-at e:62:4:b:ef:93
e:62:4:b:ef:93 4e:71:9f:d:e4:b8 0806 42: arp reply 172.18.0.1 is-at e:62:4:b:ef:93
e:62:4:b:ef:93 4e:71:9f:d:e4:b8 0806 42: arp reply 172.18.0.1 is-at e:62:4:b:ef:93
e:62:4:b:ef:93 4e:71:9f:d:e4:b8 0806 42: arp reply 172.18.0.1 is-at e:62:4:b:ef:93
e:62:4:b:ef:93 4e:71:9f:d:e4:b8 0806 42: arp reply 172.18.0.1 is-at e:62:4:b:ef:93
e:62:4:b:ef:93 4e:71:9f:d:e4:b8 0806 42: arp reply 172.18.0.1 is-at e:62:4:b:ef:93
e:62:4:b:ef:93 4e:71:9f:d:e4:b8 0806 42: arp reply 172.18.0.1 is-at e:62:4:b:ef:93
e:62:4:b:ef:93 4e:71:9f:d:e4:b8 0806 42: arp reply 172.18.0.1 is-at e:62:4:b:ef:93
^CCleaning up and re-arping targets...
e:62:4:b:ef:93 4e:71:9f:d:e4:b8 0806 42: arp reply 172.18.0.1 is-at 22:9a:5d:4a:1b:d0
e:62:4:b:ef:93 4e:71:9f:d:e4:b8 0806 42: arp reply 172.18.0.1 is-at 22:9a:5d:4a:1b:d0
e:62:4:b:ef:93 4e:71:9f:d:e4:b8 0806 42: arp reply 172.18.0.1 is-at 22:9a:5d:4a:1b:d0
e:62:4:b:ef:93 4e:71:9f:d:e4:b8 0806 42: arp reply 172.18.0.1 is-at 22:9a:5d:4a:1b:d0
e:62:4:b:ef:93 4e:71:9f:d:e4:b8 0806 42: arp reply 172.18.0.1 is-at 22:9a:5d:4a:1b:d0
```

Figure A.37: ARP spoofing



```
┌─[root@97b315a6881d]─[~]
└──╼ #hping3 -A -f -p 80 172.18.0.2
HPING 172.18.0.2 (eth0 172.18.0.2): A set, 40 headers + 0 data bytes
len=40 ip=172.18.0.2 ttl=64 DF id=0 sport=80 flags=R seq=0 win=0 rtt=0.5 ms
len=40 ip=172.18.0.2 ttl=64 DF id=0 sport=80 flags=R seq=1 win=0 rtt=0.2 ms
len=40 ip=172.18.0.2 ttl=64 DF id=0 sport=80 flags=R seq=2 win=0 rtt=1.0 ms
len=40 ip=172.18.0.2 ttl=64 DF id=0 sport=80 flags=R seq=3 win=0 rtt=0.8 ms
len=40 ip=172.18.0.2 ttl=64 DF id=0 sport=80 flags=R seq=4 win=0 rtt=0.6 ms
len=40 ip=172.18.0.2 ttl=64 DF id=0 sport=80 flags=R seq=5 win=0 rtt=0.3 ms
len=40 ip=172.18.0.2 ttl=64 DF id=0 sport=80 flags=R seq=6 win=0 rtt=1.0 ms
len=40 ip=172.18.0.2 ttl=64 DF id=0 sport=80 flags=R seq=7 win=0 rtt=0.7 ms
len=40 ip=172.18.0.2 ttl=64 DF id=0 sport=80 flags=R seq=8 win=0 rtt=0.6 ms
len=40 ip=172.18.0.2 ttl=64 DF id=0 sport=80 flags=R seq=9 win=0 rtt=0.3 ms
len=40 ip=172.18.0.2 ttl=64 DF id=0 sport=80 flags=R seq=10 win=0 rtt=0.2 ms
len=40 ip=172.18.0.2 ttl=64 DF id=0 sport=80 flags=R seq=11 win=0 rtt=0.9 ms
len=40 ip=172.18.0.2 ttl=64 DF id=0 sport=80 flags=R seq=12 win=0 rtt=0.6 ms
len=40 ip=172.18.0.2 ttl=64 DF id=0 sport=80 flags=R seq=13 win=0 rtt=0.3 ms
len=40 ip=172.18.0.2 ttl=64 DF id=0 sport=80 flags=R seq=14 win=0 rtt=0.2 ms
len=40 ip=172.18.0.2 ttl=64 DF id=0 sport=80 flags=R seq=15 win=0 rtt=1.0 ms
len=40 ip=172.18.0.2 ttl=64 DF id=0 sport=80 flags=R seq=16 win=0 rtt=0.8 ms
len=40 ip=172.18.0.2 ttl=64 DF id=0 sport=80 flags=R seq=17 win=0 rtt=0.6 ms
^C
--- 172.18.0.2 hping statistic ---
18 packets transmitted, 18 packets received, 0% packet loss
round-trip min/avg/max = 0.2/0.6/1.0 ms
```

Figure A.38: TCP ACK flood with brocken packets