VIETNAM NATIONAL UNIVERSITY HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Vo Van Hoang

ENHANCING INTRUSION DETECTION PERFORMANCE BY DATA AUGMENTATION, PARALLEL ENSEMBLE INFERENCE, AND FLOW SENSING STRATEGY

PHD DISSERTATION IN INFORMATION SYSTEMS

Ha Noi - 2025

VIETNAM NATIONAL UNIVERSITY HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Vo Van Hoang

ENHANCING INTRUSION DETECTION PERFORMANCE BY DATA AUGMENTATION, PARALLEL ENSEMBLE INFERENCE, AND FLOW SENSING STRATEGY

Major: Information Systems
Code: 9480104.01

PHD DISSERTATION OF INFORMATION SYSTEMS

PhD STUDENT **SUPERVISORS**

Vo Van Hoang Nguyen Ngoc Hoa Nguyen Ngoc Tu

Ha Noi - 2025

Declaration of Authorship

I, Vo Van Hoang, declare that this dissertation titled, "ENHANCING INTRUSION DETECTION PERFORMANCE BY DATA AUGMENTATION, PARALLEL ENSEMBLE INFERENCE, AND FLOW SENSING STRATEGY" and the work presented in it are my own. I confirm that:

- This work was done mainly while in candidature for the degree of Ph.D at VNU University of Engineering and Technology.
- This dissertation has not previously been submitted for any degree.
- The results in my dissertation are my independent work, except where works in the collaboration have been included. Other appropriate acknowledgments are given within this dissertation by explicit references.

Signed: _____

Date: _____

Acknowledgements

This dissertation is the culmination of a long and challenging journey, and it would not have been possible without the invaluable support, guidance, and encouragement I have received from many individuals.

First and foremost, I wish to express my heartfelt gratitude to my supervisors, Associate Professor Nguyen Ngoc Hoa and Associate Professor Nguyen Ngoc Tu. Their deep expertise, steadfast guidance, and unwavering patience have been essential to every stage of this research. Their insightful feedback and persistent encouragement have shaped not only the quality of this work but also my own academic growth. I am profoundly thankful for their mentorship and belief in my potential.

I am also deeply grateful to my colleagues and friends, whose support, camaraderie, and inspiring conversations have sustained my motivation and helped me overcome the inevitable challenges along the way. Their encouragement has made this demanding journey both enriching and rewarding.

Above all, I extend my deepest thanks to my family. Their unconditional love, understanding, and sacrifice have been my anchor and my greatest source of strength. Without their constant support and faith, this achievement would not have been possible.

To all who have contributed to this work and supported me in ways both big and small, I offer my sincere thanks. This accomplishment is as much yours as it is mine.

Abstract

Rapid proliferation and increasing sophistication of cyberattacks pose formidable challenges to traditional intrusion and malware detection methods, especially with regard to accuracy, scalability, interpretability, and real-time responsiveness. Addressing these critical issues, this dissertation proposes an integrated AI-powered threat detection framework that advances the state of large-scale cybersecurity defense across three tightly connected research threads.

First, to overcome the pervasive class imbalance and high-dimensional feature redundancy in cybersecurity datasets, an innovative data-centric pipeline is introduced. This pipeline proposes a method to augment the quality of the training dataset by compressing samples in the majority classes and generating more realistic samples in minority classes; determining the optimal feature set to enhance efficiency regarding detection rate. The result is a balanced and meaningful training set that significantly improves the detection of minority and emerging threats, laying a robust data foundation for subsequent modeling.

Second, leveraging this enhanced dataset, the dissertation develops a mutual deep+boosting ensemble approach that fuses the strengths of neural networks and advanced boosting models. We employ an ensemble of mutual deep learning and gradient-boosting inference, initially for voting among multiple AI-based classifiers, followed by stacking individual and voting probability predictions to improve malware detection and reduce vulnerability to model poisoning.

Third, to bridge the gap between research prototypes and real-world deployment, the dissertation presents NetIPS, a scalable, real-time intrusion prevention system. NetIPS integrates dynamic flow sensing with parallel ensemble inference and sandbox, enabling the system to focus computational resources on high-risk traffic and maintain wire-speed performance in large-scale operational environments.

Extensive experiments on multiple benchmark datasets, such as CSE-CIC-IDS2018, NSL-KDD, EMBER, and BODMAS, demonstrate the effectiveness of the proposed methods, with clear improvements in recall, overall accuracy, model transparency, and readiness for deployment. The dissertation contributions are supported by 03 SCI/E-indexed journal articles and 04 WoS-indexed conference articles. Together, this work delivers a coherent scientific foundation and a practical roadmap for developing next-generation, explainable, and deployable AI-powered cyber defense systems.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	**Full Term**
ANN	Artificial Neural Network
AES	Advanced Encryption Standard
APT	Advanced Persistent Threat
AutoGluon	AutoML Toolkit
AutoML	Automated Machine Learning
AWGAN	Augmented Wasserstein Generative Adversarial Network
Ax	Adaptive Experimentation Platform
BME	Bagging Model Ensemble
BODMAS	Benign and Malicious Portable Executable Dataset for Malware Detection
CBT	CatBoost
CICFlowMeter	Canadian Institute for Cybersecurity Flow Meter
CNN	Convolutional Neural Network
CRNN	Convolutional Recurrent Neural Network
CSE-CIC-IDS2018	Canadian Institute for Cybersecurity Intrusion Detection System 2018
DIC	Deep Inspection Cycle
DIW	Deep Inspection Window
DL	Deep Learning
DNN	Deep Neural Network
EMBER	Endgame Malware Benchmark for Research
ENN	Edited Nearest Neighbours
F1	F1-score
FAR	False Acceptance Rate
FastAI	Deep Learning Framework (Python)
FNR	False Negative Rate
FPR	False Positive Rate
GAN	Generative Adversarial Network
GBM	Gradient Boosting Machine
IDS	Intrusion Detection System

Abbreviation	**Full Term**
IPS	Intrusion Prevention System
IoC	Indicator of Compromise
KNN	K-Nearest Neighbors
LightGBM	Light Gradient Boosting Machine
LSTM	Long Short-Term Memory
MDOB	Mutual Deep+Boosting (Ensemble Learning)
ML	Machine Learning
NSL-KDD	NSL Knowledge Discovery in Databases
Optuna	Optimization Framework for Hyperparameter Tuning
PAID	Parallel AI-powered Intrusion Detection
PE	Portable Executable (file format)
PELID	Parallel Ensemble Learning-based Intrusion Detection
RF	Random Forest
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
SDAID	Signature and Deep Analysis-based Intrusion Detection
SHAP	SHapley Additive exPlanations
SMOTE	Synthetic Minority Over-sampling Technique
SOTA	State-Of-The-Art
SQLi	SQL-Injection
Suricata	Open-source IDS/IPS engine
SVM	Support Vector Machine
t-SNE	t-distributed Stochastic Neighbor Embedding
VM	Virtual Machine
XGB,XGBoost	Extreme Gradient Boosting
YARA	Tool for Identifying and Classifying Malware Samples

Introduction

Motivation

Recently, the increasing wave of cyberattacks has underscored the urgent need for more intelligent, adaptive and scalable threat detection systems. As digital transformation accelerates across critical domains ranging from government and healthcare to financial services and industrial control systems, cybersecurity has transitioned from a technical concern to a national security imperative [61, 16]. The increasing dependence on interconnected systems, cloud services, and ubiquitous computing platforms has significantly expanded the attack surface, providing adversaries with more entry points, vectors, and opportunities for disruption.

Although cybersecurity tools have traditionally relied on signature-based and rule-driven detection mechanisms, such approaches struggle to keep pace with modern threats [15, 16]. Static rule engines are inherently reactive: they are only as effective as the knowledge base that drives them. This limitation makes them vulnerable to zero-day attacks, polymorphic malware, and adversarial evasion techniques. In addition, traditional intrusion detection systems (IDS) often produce too many false alarms, struggle to handle new types of data, and do not adapt well to changing network conditions. These systemic shortcomings call for a paradigm shift towards AI-powered detection frameworks that can learn from data, recognize evolving attack patterns, and generalize beyond known threats.

The application of machine learning (ML) and deep learning (DL) in cybersecurity offers a promising pathway to enhance threat detection capabilities [50, 52, 105, 22, 67]. By learning from vast datasets of network traffic, system logs, and binary executables, these models can automatically identify patterns associated with malicious behavior, often with minimal human supervision. However, the practical deployment of ML/DL in cybersecurity is fraught with its set of challenges [75]. Unlike controlled environments in other ML domains, such as image classification or language modeling, cybersecurity data are inherently noisy, imbalanced, high-dimensional, and subject to adversarial interference. These characteristics pose unique difficulties for both learning and generalization.

One of the most pressing concerns is the issue of class imbalance in cybersecurity datasets [25]. Most publicly available intrusion and malware datasets contain an overwhelming proportion of benign samples and a relatively small number of malicious instances. Furthermore, within the malicious category, attack types are often unevenly distributed with a few dominant classes overshadowing rarer but equally dangerous threats. This imbalance biases

learning algorithms toward majority classes and severely impairs their ability to detect minority classes, which often represent zero-day or advanced persistent threats (APTs). The consequence is a high false-negative rate and an unacceptable failure mode for any intrusion detection system.

Equally problematic is the heterogeneity and complexity of feature spaces in cyber threat datasets. Network traffic data and executable files often contain hundreds or thousands of features, many of which are redundant, irrelevant, or weakly correlated with the target classes. Training ML models with such data increases computational complexity and introduces noise that dilutes the learning signal. Furthermore, complex deep learning models like convolutional neural networks (CNNs) and recurrent neural networks (RNNs) are very effective in learning patterns, but they work like "black boxes," meaning that we cannot easily understand how they make decisions, which is a big problem in security-sensitive areas where we need to explain, audit, and trust the results.

Another obstacle arises from the lack of real-time inference capability in many existing systems [1]. Although high detection accuracy is possible in controlled settings, using these models in real-world situations such as data centers or edge devices often shows serious problems with speed, capacity, and flexibility. The need for lightweight, scalable, and ensemble-capable solutions has never been more pronounced.

Motivated by these challenges, this dissertation proposes a unified research roadmap that seeks to improve learning performance, resilience, and explainability in AI-based malware and intrusion detection systems. By integrating methods for balancing dataset, feature refinement, model-specific optimization, and multimodel inference, the goal is to design systems that are not only accurate but also robust, interpretable, and deployable in realistic operational contexts. Through this lens, the dissertation aims to contribute both theoretically and practically to the next generation of intelligent cybersecurity systems that are not only technically sound but also operationally viable in the ongoing arms race against cyber adversaries.

The increasing size, variety, and complexity of today's cyberattacks, particularly those using clever and changing methods, have shown that traditional detection and prevention systems have serious weaknesses. Although recent advances in machine learning and deep learning have shown promise in addressing some of these challenges, many existing approaches continue to struggle with critical issues such as data imbalance, feature redundancy, model interpretability, and real-time operational constraints. Addressing these unresolved gaps and pushing the boundaries of intelligent, scalable, and explainable threat detection forms the core motivation for this dissertation. However, these studies still have certain limitations and there is much room for improvement. This is the main motivation for us

to carry out this dissertation.

Research Challenges

Despite remarkable advances in artificial intelligence (AI), deploying effective malware and intrusion detection systems in real-world environments remains difficult due to a combination of technical, operational, and structural challenges. This dissertation focuses on three core research areas, balancing dataset, robustness and generalization of ensemble learning models, and scalable real-time detection, and identifies the following major challenges:

1. Challenge 1: Cybersecurity datasets are heavily imbalanced, with the vast majority of samples belonging to benign traffic or a few common attack types, while rare but dangerous threats (e.g., infiltration, exfiltration, and zero-day attacks) are underrepresented. This leads to biased model learning and poor detection of minority attacks. Conventional oversampling methods, such as SMOTE, often focus on increasing the sample quantity without preserving semantic fidelity, risking the introduction of noisy or unrealistic samples. In cybersecurity, where context and behavior define threat patterns, careless augmentation can degrade model performance and even create artifacts. Thus, there is a critical need for sophisticated quality-driven data enrichment methods that leverage adversarial generation and clustering-based filtering to produce meaningful and semantically coherent samples for underrepresented classes, enhancing model learning without overfitting.

2. Challenge 2: Achieving high accuracy and low false positive rates in AI-powered intrusion detection systems, while maintaining overall system performance and interpretability, remains a persistent challenge. Boosting and deep learning models need to be carefully integrated and configured to balance accuracy, efficiency, and robustness. The challenge lies in selecting appropriate models for each dataset and effectively combining machine learning models to maximize performance and enhance system resilience in intrusion detection.

3. Challenge 3: For AI-powered intrusion detection systems to be operationally viable, they must process large volumes of traffic at wire speed with minimal delay. However, the computational complexity of machine learning models often hinders real-time deployment. The challenge lies in designing lightweight, scalable model architectures to maintain detection quality while ensuring high throughput and low latency. The inherent computational complexity of advanced AI models makes it difficult to maintain both high throughput and reliable performance in demanding real-world environments.

Together, these challenges underscore the need for a multifaceted detection framework that integrates data-centric augmentation, ensemble learning, explainability, and architectural optimization. The solutions proposed in this dissertation aim to bridge these gaps with validated improvements across detection accuracy, robustness, scalability, and operational interpretability.

Research Objectives

This dissertation focuses on the development of AI-powered threat detection systems that are not only accurate and robust, but also scalable and interpretable for real-world deployment. To achieve this overarching goal, the research is structured around the following three core objectives, each corresponding to a major contribution presented in Chapter 1, Chapter 2, Chapter 3, and Chapter 4:

- Objective 1: An overview of cyberattacks and the techniques used by hackers to carry out such attacks. Research intrusion and malware detection techniques and analyze the advantages and disadvantages of each method. Evaluate the results of the latest research related to the problem of intrusion detection.

- Objective 2: We propose a augmentation dataset method that aims to improve the quality of minority attack samples, select the most representative samples from the majority classes; minimize training noise by identifying important features within the dataset. This approach improves the quality of the training dataset and boosts the performance of machine learning models. This objective arises from the fact that cybersecurity datasets often suffer from severe class imbalance, where minority attack types are underrepresented, or the feature space is highly dimensional.

- Objective 3: Traditional intrusion detection methods often struggle with generalization and robustness against novel or adversarial attacks. This objective aims to integrate neural networks with boost models through soft voting and stacking strategies. The goal is to take advantage of the complementary strengths of each type of model to improve the accuracy of the classification in both host-based and network-based threat detection. Specifically, we employ an ensemble of mutual deep learning and gradient-boosting inference, initially using soft voting among multiple AI-based classifiers, followed by stacking the individual and voting probability predictions to enhance malware detection and reduce vulnerability to model poisoning.

- Objective 4: AI-based detection systems often suffer from inference latency and limited scalability. This objective aims to design a lightweight, high-throughput detection

architecture with support for flow-based sensing and parallel ensemble inference. The resulting system is implemented in the user space and is evaluated under simulated network conditions to validate low-latency operation and reliable threat detection.

These objectives form the foundation for the dissertation's contributions to practical and theoretical advancements in AI-driven cybersecurity detection systems.

Research Scope

To achieve the objectives of this dissertation, we focus on the following key areas:

1. Research data structures and class imbalance in intrusion detection datasets and study machine learning and deep learning models for their effectiveness.

2. Research focuses on building lightweight high-throughput detection architectures suitable for real-time deployment in large-scale networks.

Research Methodologies

This dissertation employs a systematic and layered research methodology, as outlined below:

- **Theoretical Methodology:** We conduct a comprehensive survey, synthesis, and evaluation of previous research relevant to intrusion detection and malware classification. The focus is on analyzing prior solutions to class imbalance, real-time detection bottlenecks, and the lack of explainability in AI-based security models. The literature is collected from highly regarded sources such as IEEE Xplore, ACM Digital Library, SpringerLink, ScienceDirect, and Wiley Online Library. From this review, we find research gaps and suggest ways to improve learning performance, detection strength, and understanding of models.

- **Experimental Methodology:** The proposed frameworks and algorithms are empirically validated through extensive experiments on multiple benchmark datasets, including public and custom-prepared corpora, specifically:

  – For balancing dataset, the methodology leverages an advanced adversarial augmentation dataset, feature optimization on datasets such as CSE-CIC-IDS2018, EMBER. The effectiveness in improving minority class detection and generalization is thoroughly evaluated.

– For robust mutual deep+boosting ensemble inference, the experimental setup integrates Neural Networks with gradient boosting models, employing soft voting and stacking. Performance is evaluated on both network-based intrusion and host-based malware classification tasks (e.g. NSL-KDD, BODMAS), with emphasis on accuracy, robustness, and model interpretability.

All algorithms are implemented in Python utilizing frameworks such as PyTorch, scikit-learn. Experimental results are rigorously compared with current state-of-the-art baselines, using standard metrics including accuracy, precision, recall, F1 score, AUC, and model explainability indicators.

Research Contributions

This dissertation presents a series of technical contributions that collectively advance the design, robustness, and deployability of AI-powered malware and intrusion detection systems. The key contributions are as follows:

1. We propose methods for augmentation dataset and feature set optimization. The approach integrates adversarial sample generation to enrich the minority class and employs filtering techniques to retain only semantically meaningful samples from the majority class. Additionally, for high-dimensional datasets, we perform rigorous feature selection to minimize redundancy while preserving the most discriminative attributes. This method significantly improves the ability to correctly identify the minority class and improves the overall performance of the model, as shown in tests on standard datasets such as CSE-CIC-IDS2018. *This contribution is derived from Chapter 2 and supported by VVH-J1, VVH-J2, VVH-J3, VVH-C2 and VVH-C4, as these works detail advanced augmentation dataset, feature optimization.*

2. We propose an integrated ensemble architecture that combines neural networks with boosting classifiers using both soft voting and stacking strategies. This hybrid framework leverages the complementary strengths of deep learning and tree-based models to enhance detection accuracy, robustness, and interpretability. Furthermore, by diversifying the modeling approaches, our method reduces vulnerability to model poisoning attacks and increases overall system resilience. Tests on well-known datasets show that the proposed combination of models consistently performs better than using only one model alone. *This contribution is developed in Chapter 3 and informed by VVH-J1, VVH-J3, VVH-C1, and VVH-C3, focusing on mutual ensemble inference soft voting and stacking.*

3. We design and implement NetIPS, a lightweight and real-time intrusion detection and prevention architecture optimized for large-scale network environments. Our system integrates signature-based detection, deep analysis, and behavioral analysis to achieve comprehensive intrusion detection. A dedicated analysis strategy is developed to coordinate these detection techniques optimally, allowing the system to take advantage of the unique strengths of each approach and ensure scalability for large network deployments. Evaluation on well-known benchmark datasets demonstrates that NetIPS achieves real-time performance without compromising detection quality. *This contribution corresponds to Chapter 4 and is supported by VVH-J1, VVH-J2, and VVH-C1, validating the effectiveness of flow sensing, parallel inference, and practical implementation of NetIPS in large-scale environments.*

Thesis Structure

This dissertation is structured into four chapters, each contributing to a cohesive research trajectory that spans data preparation, model development, and real-world deployment of AI-powered threat detection systems:

- Chapter 1 This chapter presents essential background knowledge in intrusion and malware detection, with an emphasis on machine learning, deep learning, and ensemble techniques. The chapter discusses common challenges such as class imbalance, high-dimensional data, and model interpretability. A comprehensive survey of existing work is conducted to identify research gaps and justify the proposed directions.

- Chapter 2 proposes augmentation dataset methods for machine learning, focusing on addressing the imbalance between minority and majority classes in the dataset. With the proposed approaches, the majority classes are sampled to select the most representative instances for training, while new high-quality samples are generated for the minority classes to ensure that both quantity and quality are on par with the majority classes, thereby balancing the class distribution for model training. In addition, the feature selection method helps to identify important and contributing features during training, eliminating low-value features to enhance model performance. This approach effectively addresses the problem of imbalance in the dataset for model training, ultimately improving the overall performance of the models.

- Chapter 3 focuses on improving machine learning models to enhance performance. The chapter proposes combining and mutually reinforcing different types of models to increase intrusion detection effectiveness and system robustness. Each model has

its own strengths, and their integration allows them to complement each other, thus improving detection capabilities and compensating for each other's limitations and reducing vulnerability to model poisoning.

- Chapter 4 proposes a practical deployment approach for intrusion detection systems in large-scale networks. A comprehensive process for intrusion detection is introduced that integrates both signature-based and behavior-based analysis, along with execution and sampling strategies. This forms a solution that enables the deployment of intrusion detection systems in large-scale network environments. With this approach, system performance is enhanced, analysis time is minimized, and the strengths of each component are maximized, enabling efficient and scalable intrusion detection for large networks.

Chapter 1

Preliminaries and Literature Reviews

This chapter provides the essential theoretical foundation and a comprehensive overview of the existing research landscape relevant to AI-powered intrusion and malware detection. First, it introduces fundamental concepts such as intrusion detection systems (IDS), malware detection, and the application of machine learning techniques in cybersecurity. Next, it discusses critical challenges, including class imbalance, high-dimensional dataset, and the need for model interpretability in real-world threat detection scenarios.

Furthermore, the chapter surveys state-of-the-art approaches and related work in the fields of augmentation dataset, machine learning model approach for both network-based and host-based intrusion detection. By synthesizing recent advances and highlighting unresolved issues, this chapter establishes the context and motivation for the novel methodologies and solutions proposed in subsequent chapters. In general, Chapter 1 aims to equip the reader with the foundational knowledge and critical perspective necessary to understand the research contributions and innovations presented in this dissertation.

1.1 Fundamental Concepts

1.1.1 Intrusion Detection System

Intrusion Detection Systems (IDS) are a fundamental component in modern cybersecurity infrastructure, designed to monitor system or network activity for signs of malicious behavior or policy violations [16, 83]. The core function of an IDS is to detect attempts at unauthorized access, exploit vulnerabilities, or anomalous behavior indicative of potential attacks. IDS solutions serve as a critical line of defense in preventing, identifying, and responding to cyber threats in real time or near real time. Depending on the deployment architecture, IDS can be broadly categorized into two types:

Network-based IDS (NIDS): These systems monitor incoming and outgoing traffic across network segments. They analyze packet data, protocol behavior, and traffic patterns to detect suspicious activity [35, 81]. NIDSs are commonly deployed at perimeter points (e.g., gateways, firewalls) and are effective in identifying threats such as denial-of-service (DoS) attacks, port scanning, or brute-force attempts.

Host-Based IDS (HIDS): These systems are installed on individual hosts or endpoints and

are responsible for monitoring system-level activities such as file changes, process execution, registry modifications, or user behavior [114, 56]. HIDS is particularly useful for detecting malware infections, privilege escalations, and unauthorized system access that may not be visible from the network layer. In terms of detection techniques, IDS are commonly classified as follows:

Signature-based detection: This approach relies on a database of known attack signatures or predefined [26, 28]. Although highly effective in detecting previously identified threats, it cannot detect novel or obfuscated attacks, including zero-day exploits.

Anomaly-based detection: These systems learn the normal behavior of users, systems, or networks and flag deviations from the learned patterns as potential intrusions. Anomaly-based methods offer improved generalization to unknown attacks, but often suffer from high false-positive rates due to the variability of legitimate behavior. Recent advances in artificial intelligence, particularly machine learning (ML) and deep learning (DL) [6, 40], have enabled a shift from static signature-based models to data-driven adaptive IDS. ML-based IDS can learn complex patterns from historical data and generalize to previously unseen threat variants, making them more suitable for evolving and dynamic attack environments. However, the practical deployment of such systems remains challenging due to issues such as data imbalance, adversarial evasion, and real-time performance constraints challenges, which are directly addressed in the subsequent chapters of this dissertation.

Malware, short for malicious software, refers to any software intentionally designed to cause damage, unauthorized access, data theft, or disruption to computer systems, networks, or users [107]. It encompasses a wide range of threat types, but is not limited to viruses, worms, trojans, ransomware, spyware, and rootkits. As malware continues to evolve in complexity and stealth, it poses a persistent threat to both individual users and enterprise infrastructures. Malware detection is broadly categorized into two approaches: dynamic analysis and static analysis.

Dynamic malware detection involves executing a suspicious file in a controlled environment (e.g., sandbox) to observe its behavior in real time. This method is effective in uncovering run-time behavior, such as network communications, file modifications, or system calls [103, 69]. However, dynamic analysis is time-consuming, resource-intensive and susceptible to evasion by malware that uses sandbox detection or delayed execution techniques.

In contrast, static malware detection analyzes the structure and content of executable files without executing them [54]. This includes inspecting binary code, headers, metadata, imported libraries, and embedded resources. Static analysis is computationally efficient and safer to apply at scale, making it suitable for real-time scanning and host-based endpoint

protection.

However, it often struggles to detect obfuscated or polymorphic malware unless combined with robust feature engineering and learning mechanisms. Researchers also focus on the static analysis of Portable Executable Files (PE) which are the standard binary format used by Windows operating systems. PE files offer a rich source of structural and semantic information, including file headers, section tables, import/export functions, and entropy-based patterns. When extracted effectively, these features provide valuable input for machine learning models to classify files as benign or malicious [95]. Recent studies have shown that machine learning models, particularly those trained on large PE datasets, can achieve high accuracy in malware classification tasks [74, 29, 59]. However, challenges persist due to the high dimensionality of the extracted features, the imbalance between benign and malicious samples, and the need for interpretable decision making.

1.1.2 Common Types of Network Attacks

Network attacks represent any malicious activity aimed at violating the confidentiality, integrity, or availability of computer networks and their resources. A clear understanding of various types of network attacks is crucial for designing and evaluating intrusion detection and prevention systems. This section provides an overview of the most prevalent and impactful categories of network attacks in modern cybersecurity.

Network attacks can be categorized in various ways, such as by intent, technique, affected OSI layer, or exploited vulnerabilities. In the following, we classify attacks by the most relevant functional categories for academic research and practical defense.

1. **Denial-of-Service Attacks (DoS/DDoS):** DoS attacks aim to disrupt network service availability by overwhelming targets with excessive requests or malicious traffic, exhausting resources such as bandwidth or CPU [97]. DDoS attacks amplify this effect by leveraging multiple compromised machines (botnets). Typical techniques include SYN Flood, UDP Flood, ICMP Flood, HTTP Flood, and amplification attacks (e.g., DNS amplification). A famous example is the 2016 Dyn DNS DDoS, which disrupted major internet services.

2. **Scanning and Enumeration Attacks:** Attackers gather intelligence through port scanning, vulnerability scanning, and network mapping. Port scanning identifies open ports and services (e.g., using `nmap`), while vulnerability scanning seeks known weaknesses. Such reconnaissance is often a precursor to further exploitation and is detectable by intrusion detection systems [7].

3. **Spoofing Attacks:** Spoofing attacks involve the deliberate falsification or manip-

ulation of identity information in network communications, enabling adversaries to masquerade as trusted entities. These attacks undermine the integrity and trust of network protocols, and often serve as precursors to more sophisticated threats such as man-in-the-middle (MitM), session hijacking, or data theft.

4. **Man-in-the-Middle (MitM) Attacks:** occur when adversaries intercept and potentially alter communication between parties without their knowledge. Techniques include ARP poisoning, rogue access points, SSL stripping, and session hijacking, leading to credential theft or unauthorized data manipulation.

5. **Eavesdropping and Sniffing Attacks:** Eavesdropping involves unauthorized interception of network traffic to capture sensitive data. Passive sniffing targets unencrypted networks, while active sniffing may leverage ARP poisoning. Tools such as Wireshark and `tcpdump` are often misused for this purpose.

6. **Replay and Session Hijacking Attacks:** Replay and session hijacking attacks target the integrity and confidentiality of communications by exploiting weaknesses in session management and authentication mechanisms.

7. **Malware-Based Network Attacks:** Malware-based attacks involve the deployment and propagation of malicious software designed to infiltrate, disrupt, or gain control over systems and networks [79]. These attacks are highly diverse in technique and impact, often leveraging network vectors for both initial infection and command-and-control (C2) communications .

    - Worms (e.g., WannaCry): Worms are self-replicating programs that spread autonomously across networks by exploiting software vulnerabilities or weak configurations. The infamous WannaCry worm, for instance, exploited a Windows SMB vulnerability (EternalBlue) in 2017, infecting hundreds of thousands of systems globally within hours. Worms typically require no user interaction, can rapidly consume network bandwidth, and often deliver secondary payloads such as ransomware or rootkits.
    - Trojans: Trojans masquerade as legitimate software or files, tricking users into installing them. Once executed, trojans can open backdoors, allowing remote attackers persistent access, or serve as droppers for additional malware. Unlike worms, trojans do not self-replicate but rely on social engineering or software bundling for distribution.
    - Ransomware: Ransomware encrypts files or entire systems and demands payment (often in cryptocurrency) for decryption. Modern ransomware campaigns leverage

phishing, exploit kits, or exposed RDP services to gain a foothold, and frequently exfiltrate sensitive data before encryption (double extortion). Notable examples include CryptoLocker, WannaCry, and REvil.

Modern malware increasingly uses techniques such as encrypted network traffic (e.g., over HTTPS or custom protocols), fileless payloads, and multi-stage infections to evade detection by traditional antivirus and intrusion detection systems.

8. **Phishing, Spear Phishing, and Social Engineering Attacks:** These attacks primarily exploit human vulnerabilities to breach technical defenses, often serving as the initial stage of broader cyberattacks.

9. **SQL Injection and Web-Based Attacks:** Web applications are frequent targets for network-based attacks that exploit vulnerabilities in input validation, authentication, and session management [42].

10. **Advanced Persistent Threats (APT):** APTs are sophisticated, multi-stage attacks typically orchestrated by organized groups for long-term, stealthy access and data exfiltration [22]. These often combine social engineering, malware, and lateral movement, as seen in campaigns such as Stuxnet and SolarWinds.

11. **Supply Chain Attacks:** Attackers compromise trusted third-party providers (software, hardware, or services) to infiltrate target organizations. Notable example: SolarWinds breach in 2020.

12. **Insider Threats:** Insiders (employees, contractors) may intentionally or accidentally leak sensitive data, disable security controls, or aid external attackers. These attacks are hard to detect due to legitimate credentials and access.

    The network attack landscape is continually evolving, with emerging trends including:

    - Encrypted Traffic Attacks: Use of TLS/SSL for malicious command-and-control (C2) and data exfiltration.
    - IoT Attacks: Exploitation of insecure devices for botnets (e.g., Mirai).
    - Fileless Attacks: Leveraging legitimate tools (e.g., PowerShell) to avoid detection.
    - Cloud-Specific Attacks: Misconfiguration and privilege escalation in cloud environments.

    A comprehensive, multi-layered defense is essential to counter both traditional and novel threats. Understanding common types of network attacks is foundational for

Table 1.1: Summary of Common Network Attack Types

Attack Type	**Technique**	**Impact**	**Detection**
Denial-of-Service (DoS/DDoS)	Traffic floods, amplification	Service unavailability	Rate limiting, filtering
Scanning & Enumeration	Port/vulnerability scans	Reconnaissance	IDS, anomaly detection
Spoofing	IP/ARP/DNS falsification	Evasion, redirection	Authentication, ARP/DNS security
Man-in-the-Middle (MitM)	Interception, SSL stripping	Data theft, manipulation	Encryption, certificate pinning
Sniffing/ Eavesdropping	Passive/active traffic capture	Credential leakage	TLS, VPN
Replay/Session Hijacking	Packet replay, session ID theft	Unauthorized access	Token/session management, TLS
Malware Propagation	Worms, trojans, ransomware	Compromise, data loss	Antivirus, sandboxing
Phishing/Social Engineering	Deceptive messages, psychological tricks	Credential theft, initial access	User training, email filtering
SQLi/XSS/CSRF	Web input manipulation	Data theft, defacement	Input validation, WAF
APT	Multi-stage, stealthy infiltration	Espionage, long-term theft	Behavior analytics, EDR
Supply Chain	Third-party compromise	Widespread breach	Vendor management, code review
Insider Threat	Privileged misuse, data exfiltration	Confidentiality breach	Monitoring, least privilege, DLP

building and evaluating intrusion detection and prevention systems. By addressing both classic and modern attack vectors, researchers and practitioners can better design comprehensive, resilient cybersecurity solutions. The summary of common network attack types show as Table 1.1.

1.1.3 Machine Learning in Cybersecurity

Machine learning (ML) has emerged as a core enabler of intelligent cybersecurity systems due to its ability to learn complex patterns from data and generalize to previously unseen threats. In contrast to traditional rule-based or signature-based detection mechanisms [26, 28, 35, 81], which are based on predefined knowledge, ML-based approaches can adapt to evolving attack behaviors, making them particularly valuable for protecting against zero-day attacks and obfuscated malware [6, 40, 114, 56]. In the context of cybersecurity, machine learning techniques have been successfully applied to a wide range of tasks, including:

- Intrusion detection: Classifying network traffic as benign or malicious based on flow-level or packet-level features.
- Malware classification: Detecting and categorizing executable files as benign or malicious using static or dynamic features.
- Phishing and spam detection: Identify malicious URLs, emails, or messages.
- Behavioral analysis: Profiling user or system behavior to identify anomalies or insider threats.

Commonly used machine learning algorithms in these domains include:

- Tree-based models: Decision Trees, Random Forests, XGBoost, LightGBM, and CatBoost are widely used due to their robustness, interpretability, and ability to handle mixed data types. These models perform well on structured tabular data such as network flows or PE file attributes.
- Support Vector Machines (SVM): Effective for binary classification problems with well-separated classes, but less scalable for large datasets.
- K-Nearest Neighbors (KNN): Simple and intuitive but computationally expensive in high-dimensional spaces.

In recent years, deep learning (DL) models have gained traction for more complex cybersecurity tasks. Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Autoencoders have been applied to model patterns in raw byte sequences, logs, and network flows. For example:

- CNNs can learn spatial patterns in structured inputs or bytecode (e.g., PE headers) [74].
- RNNs are suitable for sequential data, such as network traffic over time or system calls.

Despite their power, ML and DL models in cybersecurity face several critical challenges:

- Imbalance dataset: Benign data often outnumber malicious instances in a large amount, which can cause classifiers to be biased toward the majority class [53].
- Generalization and Overfitting: Models trained on fixed datasets may not be generalized to real-world threats or novel attack types [2, 63].
- Interpretability: Many high-performance models act as black boxes, offering little insight into why a sample was classified as malicious, limiting the trust and auditability of the analyst.
- Real-time constraints: In operational environments, models must deliver predictions with low latency and high throughput, requiring careful trade-offs between complexity and efficiency [63].

Addressing these challenges requires not only algorithmic innovation, but also principled data processing, model ensemble design, and performance tuning. In this dissertation, we investigate multiple machine learning pipelines that span boost models, deep neural networks, and hybrid ensembles tailored specifically for intrusion and malware detection tasks. Particular emphasis is placed on handling imbalanced data, optimizing model performance, and ensuring deployment feasibility in real-time or resource-constrained environments.

1.1.4 Class Imbalance in Cybersecurity Dataset

Class imbalance is a widespread challenge in cybersecurity datasets, particularly in the domain of intrusion detection. Most real-world traffic traces and publicly available datasets exhibit a skewed distribution in which benign samples outnumber malicious ones [34]. Moreover, even within malicious categories, certain attack types such as port scanning or denial-of-service are vastly overrepresented, while advanced or stealthy attack vectors appear only sparsely.

This imbalance severely affects the training and performance of machine learning (ML) models [98, 111]. Standard classifiers tend to optimize for overall accuracy, which can lead to poor recall of minority (attack) classes, undermining the reliability of threat detection systems. In extreme cases, models may entirely ignore rare but dangerous attack categories

to reduce overall classification loss. There are two primary dimensions to the imbalance problem in cyber threat detection:

- Global imbalance: Refers to the disproportionate ratio between benign and malicious samples. For example, in the CSE-CIC-IDS2018 dataset, benign traffic often constitutes more than 80–90% of the total records.

- Intraclass imbalance: Occurs within the malicious subset, where attacks such as brute-force login attempts dominate, while more sophisticated threats such as infiltration, backdoors, or botnet activities are underrepresented.

Various techniques have been proposed to address class imbalance in ML workflows:

- Data-level methods: These include oversampling the minority class (e.g. random duplication, SMOTE), undersampling the majority class, or a combination of both, such as Sinha et al. [99] proposed a conventional oversampling procedure to balance the dataset. The experiment evaluated the CNN-BiLSTM model based on the NSL-KDD and UNSW-NB15 datasets with 10-fold cross-validation. It achieves an accuracy of 99.22% and a detection rate of 98. 88% for the NSL-KDD dataset. However, the experiment only demonstrates cross-validation and does not include independent test data after execution. However, naive oversampling can cause overfitting, while undersampling can discard valuable information.

- Algorithm-level methods: Modifications to loss functions (e.g., weighted loss, focal loss) or ensemble methods like boosting can mitigate imbalance by assigning a greater penalty to misclassified minority samples. Liu et al. [68] presented a technique to balance the dataset for network IDS, namely DSSTE. This method used techniques to balance the dataset for the minority and majority classes. The result of the experiment of this approach is achieved by 96.99% and 82.84% for the CSE-CIC-IDS2018 and NLS-KDD datasets, respectively.

- Hybrid approaches: Integrating data balancing with ensemble techniques or meta-learning strategies can yield better generalization and robustness. For example, Gupta et al. [43] presented a solution to balance the dataset. This method integrates DL and ensemble learning algorithms with data-level techniques based on Random Oversampling (ROS) and SVM-SMOTE. Before data oversampling, they used DNN to perform binary classification of benign and attack network traffic flow, followed by the XGB algorithm. They also distinguish between the majority and minority attack classes. The dataset is then resampled and the RF algorithm is applied to classify the various minority attack classes.

Despite these efforts, many solutions fall short in practical deployment due to lack of semantic quality control over generated samples, insufficient treatment of rare but critical classes, or poor integration with feature selection and model tuning. These shortcomings often lead to high false negative rates and an especially dangerous failure mode in intrusion detection systems (IDS).

1.1.5 Ensemble Learning in Intrusion Detection

Ensemble learning is a machine learning paradigm in which multiple models referred to as base learners are strategically combined to achieve better predictive performance compared to any individual constituent [38]. This approach is particularly advantageous in cybersecurity, where attack patterns are diverse, constantly evolving, and often subtle enough to elude single-model detection systems.

The core intuition behind the ensemble methods is that a group of diverse, moderately accurate models can collectively produce more robust and accurate predictions, especially when their errors are not correlated [9]. In threat detection contexts, ensemble learning contributes to:

- Higher accuracy and stability: Reducing overfitting and variance, particularly in small or imbalanced datasets.
- Improved generalization: Handling a wider range of attack types and data distributions.
- Resilience to evasion: Mitigating the weaknesses of individual models by relying on consensus.

There are several ensemble strategies relevant to intrusion and malware detection:

- Bagging (Bootstrap Aggregating): Techniques like Random Forest build multiple models on randomly resampled subsets of the data and average their predictions, reducing variance [72].
- Boosting: Methods such as XGBoost, LightGBM, and CatBoost sequentially train models in which each learner focuses on the errors of its predecessors. These models are powerful for structured cybersecurity data and have shown superior performance in malware and intrusion classification tasks [56].
- Voting ensembles: Combine predictions from heterogeneous models (e.g., a CNN and a decision tree) using hard (majority vote) or soft (probability averaging) voting to improve overall robustness [38].

- Stacking: A more sophisticated strategy where outputs of several base models are used as features for a meta-learner. This two-layer architecture can capture complex relationships between models, often leading to state-of-the-art results [91].

In cybersecurity, ensembles are especially beneficial due to the heterogeneity of feature types and attack vectors. For example, network intrusion detection may rely on time series or flow-level features, while static malware analysis may use structural metadata or byte-level patterns. No single-model architecture can effectively cover all these modalities, but ensemble strategies can integrate their strengths.

1.2 Approaches to Threat Detection

1.2.1 AI-powered Intrusion Detection

The model concept in ML/DL or AI refers to the mathematical processing of the prediction Y from the input X. The model's parameters are factors to learn from data to have a mode that can make a good prediction. The objective function is to maximize or minimize to find such parameters.

For intrusion detection based on an AI model, each network traffic flow is modeled by a vector of features $f = [a_1, ..., a_n]$. These vectors are analyzed to discover anomalies using a classification method. Current research in [2, 63] affirms that deep neural networks, gradient boosting machines (GBM), and gradient boosting are the best methods to predict intrusion attacks. Thus, we focus on these methods in our research to improve intrusion detection by deep network traffic analysis.

For boost learning methods, minimizing prediction errors is tactically performed by introducing a gradient term [40]. Thus, eXtreme Gradient Boosting (XGB) is considered a typical ML for intrusion detection. XGB uses decision trees as weak learners and combines their contributions to produce a strong learner. XGB uses an ensemble of K classification and regression trees, each of which has $K_E^i | i \in 1..K$ nodes. The final prediction is the sum of the prediction scores for each tree:

$$\hat{y_i} = \varphi(x_i) = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in F, \tag{1.1}$$

where x_i are members of the training set, y_i are the corresponding class labels, f_k is the leaf score for the tree k^{th} and F is the set of all K scores for all classification and regression trees. In XGB, the objective function is created using Taylor's theorem [18],

$$obj^{(t)} = \sum_{i=1}^{n} \left[m_i f_t(p_i) + \frac{1}{2} c_i f_t^2(p_i) \right] + \Omega(f_t) \tag{1.2}$$

where m_i and c_i are the inputs. The result of the objective function is a tree that adds itself to the model.

The GBM is also another typical ML for intrusion detection. It was the first version of a gradient-boosting ensemble that adopted a forward learning strategy. Trees are created sequentially, where subsequent trees rely on the outcomes of the preceding trees. Generally, GBM is accomplished through the iterative construction of a set of functions $f^0, f^1, ..., f^t$, given a loss function $\Omega(y_i, f^t)$. Suppose that function ft has been constructed; we can optimize our estimates of y_i by discovering another function $f^{t+1} = f^t + h^{t+1}(\mathbf{x})$ such that h^{t+1} diminishes the estimated value of the loss function.

On the other hand, DL is suitable for modeling complex nonlinear relationships by learning multiple data representations corresponding to different levels of abstraction. A DNN consists of several layers: input, hidden, and output. These layers are established for feature extraction and transformation. It is a promising method for detecting network attacks [39]. In DNN architecture, several activate functions have been proposed and used, including:

$$Sigmoid = \frac{1}{1+e^{-x}};\ Softmax = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{1.3}$$

$$Hyperbolic\ tangent = \frac{1-e^{-2x}}{1+e^{-2x}}; ReLU = max\{x, 0\} \tag{1.4}$$

The ReLU (Rectified Linear Unit) has the advantage of efficiently training large datasets among these functions [105].

Thus, each MD/DL or AI model has its advantage. The combination of several AI models allows us not only to improve the quality of intrusion detection but also to prevent spoofing attacks such as adversarial attacks.

Ensemble learning is a well-known technique for improving performance and reducing variance by training multiple models and combining their predictions to produce the optimal outcome. It is also an approach that combines multiple machine learning models into a more efficient solution than any single algorithm [18]. The ensemble techniques are fundamental or advanced. The fundamental ensemble methods consist of Max voting, Averaging, and Weighted Average, in which different algorithms are trained on the data, and after averaging, more powerful models are produced. Among the advanced techniques for ensembles, stacking, bagging, boosting, and combining [57].

1.2.2 AI-powered Malware Detection

In malware detection, current approaches focus on combining pattern matching and AI-powered analysis in PE files [89, 96, 107]. This study focuses on the detection of malware

embedded in PE files, the standard format for Windows executables. Detecting PE malware is a challenging task due to the complexity of PE structures, the prevalence of evasion techniques, and the vulnerability of AI models to adversarial attacks, as discussed in the Introduction.

Currently, LIEF [100] is one of the most common tools used for AI-powered malware detection [88, 37]. Here, the vector encapsulates various structural and metadata attributes extracted from the PE file (e.g., headers, sections, imports, and other characteristics).

For AI-powered malware detection, define D as the dataset composed of pairs (v, y), where v is the representation of a feature vector of a PE file and $y \in \{0, 1\}$ is the associated label (e.g. 0 for benign, 1 for malware), respectively. Thus, $D = (v_1, y_1), (v_2, y_2), ..., (v_M, y_M)$ with M representing the total number of samples. The problem is to train a generalized AI model $f : R^n \implies 0, 1$ on the dataset D such that, for any new PE file, its feature vector v is mapped to a predicted label $\hat{y} = f(v)$. The goal is to maximize the accuracy of f while generalizing well beyond the training dataset, thus enabling the reliable detection of malware in unseen PE files.

To achieve this, various ML techniques can be employed, including supervised learning algorithms such as decision trees, support vector machines, or neural networks. Additionally, feature set optimization and engineering play a crucial role in improving the model's performance by ensuring that the most relevant characteristics of the PE files are utilized for accurate predictions. Currently, CNN, XGB, CBT, and GBM are typical AI models used to predict intrusion attacks [40, 28, 56, 110]. Thus, we focus on these methods in our research to improve malware detection through PE file analysis.

1.2.3 Handling Imbalanced Datasets

The quality of the dataset impacts the performance when ML/DL is applied for intrusion detection. In general, most well-known datasets for intrusion detection have a much lower number of attacked flows than benign flows. A class with higher samples is called a 'majority' in ML classification issues. Moreover, the inverse, a class with small samples, is considered a 'minority class'. Using unbalanced datasets usually has terrible effects on both the training phase and subsequent prediction [93, 109].

Thus, we should balance datasets in ML to improve the quality of the training dataset. In imbalanced data handling, under-sampling the majority class is a conventional plan of operation. Then, oversampling techniques followed by undersampling can be used to balance the datasets. Some well-known and widely used techniques can be considered to tackle the imbalance problem, such as the synthetic minority oversampling technique (SMOTE) and edited closest neighbors (ENN) [71, 93]. Applying these methods can clean

or reduce the noise for the majority classes and generate more realistic samples for the minority classes in training datasets.

1.3 Related Work

1.3.1 Deep and Boosting Learning for Intrusion Detection

DL approaches effectively detect network attack associations within raw samples, feature learning, and classification tasks. Many DL techniques have implemented IDS in the last few years [50, 77, 33]. DL is used in real-time environments for several studies on network attack detection. For example, Bontemps et al. [19] propose a collective real-time anomaly detection model based on neural network learning and feature operation. Their method involves using typical time series data to train an LSTM RNN, followed by a live prediction for each time step. An approach for NIDS based on a hierarchical and dynamic feature extraction framework (HDFEF) wasproposed by Li et al. [65]. They defined a network activity as a series of packets using various network traffic flows. The distribution of the feature representations of several temporally associated network packets is then dynamically adjusted with an attention mechanism in a hierarchical network model. The final discriminant vectors are then obtained and utilized for classification after combining the vectors from the multi-space mapping. The precision of the HDFEF on the CSE-CIC-IDS2018 dataset is 99.05%.

Alrawashdeh et al. [11] proposed a DL method for anomaly detection using a Restricted Boltzmann Machine (RBM) and a deep belief network. Their method involved creating unsupervised feature reduction using a one-hidden-layer RBM and then passing the weights from this RBM to another RBM to create a deep belief network. With the NSL-KDD dataset, they were 97.91% accurate. In addition, Jayalaxmi et al. .[51] introduce the IDS framework known as PIGNUS, which combines an effective feature mapping method with a cascade model. The PIGNUS combines the cascade forward back propagation neural network for classification and attack detection with Auto Encoders to choose the best features. The cascade model creates an accurate categorization using related links from the input layer to the output layer to identify typical and aberrant behavior patterns. The experiment result from PIGNUS reaches 99.02% accuracy for the NSL-KDD dataset. Aldarwbi et al. [8] offer a system that transforms the netowrk traffic flow features into waves and leverages advanced audio/speech recognition DL-based methods such as LSTM, Deep Belief Networks (DBN) and CNN to detect intruders. It achieves the accuracy of 84.82% and 99.41% for the NSL-KDD and CIC-IDS2017 datasets, respectively. In other ways, the authors used the Firefly Optimization (FFO) technique to detect incursion and

the Probabilistic Neural Network (PNN) to categorize categories based on the NSL-KDD datasets [85]. The proposed approach achieves an accuracy of 98.99%.

Qazi et al. [87] presented a hybrid IDS framework using a convolutional recurrent neural network (CRNN) to detect network threats. This method merges an RNN with a CNN in which various RNN layers follow two convolutional layers. The output is then fed into fully connected, flattened, and SoftMax layers, which enable the model to detect and classify traffic. The experiment results on the CSE-CIC-IDS2018 dataset reach 98.90% accuracy. In addition, Ren et al. [92] employed CNN, and the Attention mechanism mix to construct a CA Block focused on local spatiotemporal feature extraction, using Equalization Loss v2 (EQL v2) to raise the minority class weight and balance the learning attention on minority classes. The accuracy of the experiment's results for the NSL-KDD and UNSW-NB15 datasets is 99.77% and 89.39%, respectively.

Moreover, Ghanbarzadeh et al. [36] proposed a method that uses the Multiobjective Quantum-inspired Binary Horse Herd Optimization Algorithm (MQBHOA) for IDS. This technique implements the Horse Herd Optimization Algorithm (HOA) metaheuristic optimization algorithm, a robust algorithm inspired by nature. The method achieved 99.0% and 99.78% of the accuracy of the NSL-KDD and CSE-CIC-IDS2018 datasets, respectively. In another method, Al et al. [7] offers a new classification-based NIDS on network flow traffic that generates huge amounts of data. The suggested system combines a hybrid DL (HDL) network composed of a CNN and an LSTM for a better IDS. In addition, data imbalance processing consisting of the SMOTE and Tomek-Links sampling methods termed STL was utilized to mitigate the effects of data imbalance on system performance. The accuracy of the proposed method in binary classification was 99.17% and 99.83% in multiclass classification.

Since the XGB outperforms other well-known algorithms in a single ML, its popularity is rising .[4]. Verma et al. .[104] recently suggested a technique for IDS that combined the XGB algorithm with K-Means clustering. For the NSL-KDD dataset, the experiment result has an accuracy of 81.2%, 82.38%, and 84.25%, and for the ANN, SVM, and XGB models, respectively. Furthermore, Devan et al. .[27] propose a strategy for enhancing NIDS that blends DNN with XGB. This approach uses the XGB technique for feature selection, and the experiment results are 97.60% accurate. Numerous dual-ensemble techniques involving fine-tuned CBT algorithms, such as XGB, CBT, LightGBM, and GBM, are fully assessed utilizing publicly available data sets, such as UNSW-NB15 and NSL-KDD. Louk et al. [72] presented a dual ensemble model by blending two current ensemble methods: bagging and CBT. The results of the experiment show that the presented technique achieves 94.66% accuracy.

Golchha et al. [38] present an attack detection framework for IIoT utilizing the voting-based ensemble learning method. This work includes an ensemble of current and classical ML approaches, including a histogram gradient booster, CBT, random forest (RF), and a hard-voting classifier. The result of the experiment reaches 99. 85%, 97. 90%, and 98. 83% precision for CBT, HGB, and RF, respectively. Moreover, Nazir et al. [82] suggested a wrapper-based feature selection approach called 'Tabu Search - Random Forest (TS-RF).' The Tabu search is used as a search technique, whereas the RF is used as a learning process for IDS. The suggested model achieved an accuracy of 83.12% for the UNSW-NB15 dataset.

In another approach, Hammad et al. [47] present a method to categorize network attacks called Multinomial Mixture Modeling with Median Absolute Deviation and Random Forest Algorithm (MMM-RF). This approach uses t-SNE to minimize data dimension, Correlation Feature Selection (CFS) to analyze the most important factors affecting network traffic, and SMOTE combined with Random Under-Sampling to control imbalance on the CSE-CIC-IDS2018 dataset. It has a 99.98% accuracy rate.

1.3.2 Deep and Boosting Learning for Malware Detection

Neural network (NN) approaches effectively detect malware and network attack associations within raw samples, feature learning, and classification tasks. Numerous NN techniques have been implemented in the last few years [50, 77, 33]. Some studies on attack detection employ DL in a real-time environment. For example, Divakarla et al. [29] presented a simple DNN-based Windows malware detection system that achieves a test accuracy of 96.76%. Moreover, the authors also performed an improved offensive generative model based on GAN to make the current DNN-based system accurate at 97.42%. This work demonstrates how the combination of DNN and rigorous static analysis aids in the development of a malware detection system, allowing it to learn complex features with a larger number of layers and more data.

Liu et al. [70] propose a generic ML-based visualization method for malware detection named Visual-AT. In addition, it uses the AT technique to detect and analyze malware that was initially difficult to identify, as well as potential variants, using transformed image data and two ML models. Visual-AT achieves up to 97.73% accuracy for the EMBER 2018. In [74], Marais et al. propose a malware detection model that transforms binary files into grayscale images, achieving 88% accuracy and 85% precision for EMBER 2018 in detecting packed or encrypted samples.

Moreover,Rigakia et al. [94] also came up with a way to train different kinds of surrogate model and sampling strategies to steal standalone ML models and four antivirus systems. This method presented a dual FFNN architecture, achieving the precision 98.02% for EM-

BER 2018. In [59], Lad et al. propose a model that focuses on the efficient extraction and classification of feature of PE files. They perform feature extraction, data standardization, and data cleaning techniques to address imbalances and impurities in the dataset. Using the EMBER dataset (2017 and 2018), they extracted 2,381 features and trained a deep learning (DL) model with dense and dropout layers. Their model achieved 97.53% and 94.09% accuracy, with 98.85% precision.

XGB often achieves faster training and inference times compared to other commonly used ML algorithms, such as Random Forest or SVM, when applied as standalone models [84]. Recently, Devan et al. [27] presented a method that uses the XGB technique for feature selection followed by a DNN and gets 97.60% accuracy. In addition, Mimura et al. [76] proposed a method for malware detection on PE files using printable characters using two language models for feature extraction and ML. The author uses the latest FFRI dataset between 2019 and 2021 to evaluate the method. According to the results, the XGB model achieves an accuracy of 99.0%, and the most suitable mix was Doc2Vec and multilayer perceptron, which achieved an F1 score of 98.10%. Each run time showed an almost linear increase with increasing dataset size.

Moreover, Alani et al. [76] introduced a lightweight obfuscated malware detector based on explainable ML techniques. The authors used the feature selection method (RFE) to reduce the number of features while effectively maintaining high accuracy. This method improved system efficiency when evaluated using the MalMem2022 dataset and achieved a remarkable accuracy of more than 99. 8%.

Despite the fact that many proposed ML/DL techniques have improved the development of IDS, they fail to achieve excellent performance, which consists of a low false alarm rate and a high detection rate. One of the explanations why the majority of these works disregard the imbalanced data in IDS datasets.

1.3.3 Data Augmentation

Researchers recently proposed several methods to improve the quality of datasets for training ML or DL models. For example, to balance the dataset in NIDS for industrial IoT, Zhang et al. [112] propose PWG-IDS based on WGAN with a gradient penalty to generate samples from minority class. The proposed reduces the number of iterations and generates more realistic sample data than GAN, using LightGBM for the classification algorithm. The experimental findings on the NSL-KDD and CSE-CIC-IDS2018 datasets demonstrate an accuracy of 99% and 96%, respectively.

Sinha et al. [99] proposed a conventional oversampling procedure to balance the dataset. The experiment evaluated the CNN-BiLSTM model based on the NSL-KDD dataset achiev-

Table 1.2: Summary of Related Works based Intrusion Detection

Method	Venue	Approach	Dataset	Acc(%)
RF+ miniVG-GNet [68]	IEEE Access 2020	Combination of K-Means and ENN to balance dataset then RF+ miniVGGNet to detect intrusions.	NSL-KDD, CIC-IDS2018	82.84, 96.99
WGAN+ LightGBM [112]	Computer Science 2021	Applying WGAN-GP for data generation on minority class samples and using LightGBM for the classification.	NSL-KDD, CIC-IDS2018	99.00, 96.00
MMM-RF [47]	Computer & Security 2022	Use CFS to analyze network traffic, T-SNE to minimize data dimension, and SMOTE to imbalance the CSE-CIC-IDS2018 dataset.	CIC-IDS2018	99.98
CNN, DBNs, LSTM [8]	Computers and Electrical Engineering 2022	Transforms the traffic flow features into waves and utilizes advanced audio/speech recognition deep-learning-based techniques to detect intruders.	CIC-IDS2017, NSL-KDD	99.21, 84.82
CNN+LSTM [101]	Digital Communications and Networks 2023	Used SMOTE to balance abnormal traffic, CNN to extract deep features, then CNN-LSTM to detect intrusions.	UNSW.NB15, CIC-IDS2017, NSL-KDD	99.21, 99.32, 98.45
FFO+PNN [85]	Alexandria Engineering Journal 2023	Used the FFO technique to extract features and PNN to classify categories.	NSL-KDD	98.99
CNN+EQL [92]	Computer Communications 2023	Used CNN and the Attention mechanism mingle to form a CA Block focusing on local spatiotemporal feature extraction and EQL v2 to increase the minority class weight and balance the learning attention on minority classes.	UNSW.NB15, NSL-KDD, CIC-IDS2017, CIC-DDoS2019	89.39, 99.77, 99.88, 99.58
PIGNUS [51]	Computer & Security 2023	Use Auto Encoders to select optimal features and Cascade Forward Back Propagation Neural Network for classification and attack detection.	NSL-KDD	99.02

ing an accuracy of 99.22% and a detection rate of 98.88% for the NSL-KDD dataset. However, the experiment only demonstrates cross-validation and does not include independent test data after execution. In another approach, Gupta et al. [43] presented a solution to balance the dataset. This method integrates DL and ensemble learning algorithms with data-level techniques based on Random Oversampling (ROS) and SVM-SMOTE. Before data oversampling, they used DNN to execute binary classification of benign and attack network traffic flow, followed by the XGB algorithm. They also distinguish between the majority and minority attack classes. Then the dataset is resampled, and the RF algorithm is applied to classify the various minority attack classes.

In addition, Liu et al. [68] proposed a method to balance the dataset for network IDS, namely DSSTE, to tackle the class imbalance issue. Authors affirm that the DSSTE improves the performance of intrusion detection with an experiment result of accuracy: 82.84% on the NLS-KDD dataset and 96.99% on CSE-CIC-IDS2018 compared with other methods. Some studies concentrate on splitting the training and testing data to enhance detection quality. In another example, Ullah et al. [101] proposed an IDS using transformer-based transfer learning for Imbalanced Network Traffic (IDS-INT). It uses SMOTE to balance abnormal traffic and detect minority attacks, uses CNN to extract features, and the CNN-LSTM model to detect different types of attacks with an accuracy of 99.21% for the UNSW-NB15 dataset. In addition, Liu et al. [68] presented a technique to balance the dataset for network IDS, namely DSSTE. This method used techniques to balance the dataset for the minority and majority classes. The experiment result of this approach achieves 96.99% and 82.84% for the CSE-CIC-IDS2018 and NLS-KDD datasets, respectively. Research concentrates on separating the training and testing datasets to boost detection quality; for instance, Ullah et al. [101] proposed an IDS employing transformer-based transfer learning for Imbalanced Network Traffic (IDS-INT). It employs SMOTE to balance unusual traffic and detect minority attacks, uses CNN to extract features, and the CNN-LSTM model to detect attacks with an accuracy of 99.21% on the UNSW-NB15 dataset.

Table 1.2 and Table 1.3 presents a comparative summary of state-of-the-art (SOTA) approaches, evaluated in terms of AI methodologies, approach, datasets used, and reported accuracy and precision. The analysis reveals that despite recent advancements, the quality of training datasets remains a limiting factor in achieving consistently high detection rates for intrusion detection and malware detection. Moreover, both false negative and false positive rates remain non-negligible across many approaches. These observations underscore two critical challenges in AI-powered IDS: improving the quality and representativeness of training datasets, and enhancing the performance and reliability of AI models in practical

deployment scenarios.

To address these challenges, this dissertation proposes a unified framework that integrates data-centric enhancement techniques, robust ensemble learning strategies, and scalable real-time deployment architectures, as elaborated in the following chapters.

1.4 Dataset Collection

Currently, in intrusion detection, there are several public datasets, such as DARPA (Lincoln Laboratory 1998-99), CAIDA (Center of Applied Internet Data Analysis, 2002-2016), ADFA (University of New South Wales, 2013), and NSL-KDD, CSE-CIC-IDS2017, and CSE-CIC-IDS2018 (Canadian Institute for Cyber Security). We use the following dataset for experiments to evaluate our methods:

- CSE-CIC-IDS2018 dataset is a large-scale modern intrusion detection dataset developed by the Canadian Institute of Cybersecurity. It contains realistic network traffic data with a wide range of both benign and malicious activities, making it widely used to evaluate intrusion detection methods.

- NSL-KDD dataset is an improved version of the original KDD'99 dataset, providing balanced samples of normal and attack traffic. It is commonly adopted as a standard benchmark in intrusion detection research, supporting direct comparison with related work.

- EMBER2017 dataset is a widely used benchmark dataset for Windows malware detection, containing labeled samples of malware and benign files collected in 2017 and earlier. It includes both labeled and unlabeled samples to facilitate various evaluation scenarios.

- EMBER2018 dataset, similar to EMBER2017, comprises labeled and unlabeled Windows binaries collected in 2018. It serves as a large-scale representative data set for training and testing malware detection models.

- BODMAS dataset is a recent dataset for malware detection, containing tens of thousands of labeled malware and benign samples in numerous malware families. Although comprehensive, it does not include benign binaries and lacks standardized feature definitions, presenting certain reproducibility limitations.

Table 1.3: Summary of Related Works based Malware Detection

Method	Venue	Approach	Dataset	Acc(%)
CNN [74]	Distributed Computing and Artificial Intelligence 2021	The method in this study converts binary files into grayscale images to detect malware. The model also integrates an attention mechanism to identify suspicious parts within the file.	EMBER 2018	94.00
DNN [29]	Procedia Computer Science 2022	This method builds an improved offensive generative model based on GANs to strengthen the current DNN-based system.	EMBER 2018	97.42
CNN [59]	International Journal of Computer Network and Information Security 2022	This method employs feature extraction, data standardization, and data cleaning techniques to address imbalances and impurities within the dataset.	EMBER 2017 & 2018	97.53, 94.09
EII-MBS [48]	Computers & Security 2022	This technique finds patterns in how instructions relate to each other and turns this information into vector representations to classify malware families.	BODMAS	99.29
XGB-CATB-EXT [80]	Computer, Material & Continua 2023	The technique in this study utilizes a model combining supervised and unsupervised learning to improve malware detection. Specifically, k-means clusters the data before a set of ML algorithms classifies it.	EMBER 2018	96.77
MD-ADA [17]	Computers & Security 2024	This approach combines CNN-based image embeddings and adversarial domain adaptation (using GANs) to classify malware.	BODMAS	99.29
FCG-MFD [45]	Journal of Network and Computer Applications 2025	This method uses function call graphs and node2vec along with ideas from NLP to help classify malware families.	BODMAS	99.28

1.5 Evaluation Metrics

We use standard metrics computed from the confusion matrix to evaluate the network attacks detection method, such as accuracy (noted as Acc), Precision ($Prec$), Sensitivity (Sens, or Recall/True Positive Rate), False Alarm Rate (FAR, or False Positive Rate), Recall (Rec), F1 score ($F1$), etc. The ML models built in this paper are all multilabel classification models [30]. Therefore, the metrics to evaluate model performance need to be made based on the overall assessment of all the n results of predicted labels $\hat{y}_i$ and real labels y_i, where $i \in [1...n]$. The following overall formulas compute these metrics:

$$Acc = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i \cap \hat{y}_i|}{|y_i \cup \hat{y}_i|} \quad Prec = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i \cap \hat{y}_i|}{|y_i|}$$

$$Rec = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i \cap \hat{y}_i|}{|\hat{y}_i|} \quad F1 = \frac{1}{n}\sum_{i=1}^{n}\frac{2|y_i \cap \hat{y}_i|}{|y_i| + |\hat{y}_i|}$$

To measure the overall performance of intrusion detection, we also use the false positive rate (FPR) and false negative rate (FNR), computed by the following formulas:

$$FPR = \frac{False\_Intrusion\_Number}{Total\_Number\_of\_Benigns}$$

$$FNR = \frac{False\_Benign\_Number}{Total\_Number\_of\_Intrusions}$$

To evaluate the efficacy of multiclass classification, we also use the Receiver Operating Characteristic Curve Area (AUC) [60]. It is worth noting that with numerous intrusion classes, the One-vs-Rest (OvR) strategy is more suitable than One-vs-One to be used to calculate the AUC. OvR can also handle class imbalances since it treats each class independently and does not require balanced class distributions. Thus, we compute the AUC for each class individually against the rest of the classes and then average the individual AUC scores to obtain the multiclass AUC.

$$AUC = \frac{1}{n}\sum_{i=1}^{n} AUC_i$$

1.6 Research Gaps and Approach Direction

Despite growing interest in AI-powered security solutions, several unresolved challenges hinder the development of scalable, interpretable, and resilient threat detection systems. Based on an in-depth review of current research and practical system evaluations, this dissertation identifies and addresses three key research gaps:

- Research Gap 1: Most real-world intrusion detection datasets suffer from severe class imbalance, where minority attack classes are underrepresented and difficult to learn. Conventional oversampling techniques, such as SMOTE, focus on increasing the sample volume without guaranteeing semantic quality, often introducing noise. Additionally, network flow/PE files exhibit high-dimensional feature spaces with redundant or weakly relevant attributes, further complicating model training and inference efficiency.

  Approach Direction (Chapter 2): To address these limitations, we propose augmentation dataset methods that enhances both the quantity and quality of training dataset, optimizing feature space. Our method includes:

  - Using WGAN to generate new and realistic for minority-class samples;
  - KMeans-based filtering to select semantically meaningful samples from training datasets;
  - SHAP-based feature selection to reduce dimensionality while preserving important features;
  - Evaluation of ensemble-based AI models on optimized data to validate gains in accuracy, F1-score, recall.

  The methods are validated on benchmark datasets such as CSE-CIC-IDS2018, confirming its effectiveness in improving detection performance under unbalanced datasets.

- Research Gap 2: Although recent studies have proposed various approaches to optimize machine learning models for intrusion detection, model optimization remains a persistent challenge in machine learning applications. This can be achieved through different methods, such as parameter tuning or combining multiple models to minimize misclassification and improve the predictive performance of the models.

  Approach Direction (Chapter 3): We design a mutual deep+boosting ensemble inference pipeline that leverages the complementary strengths of diverse models to enhance overall performance and reduce vulnerability to model poisoning. This chapter contributes:

  - Development of base models (CNN, XGBoost, LightGBM, CatBoost) trained on augmentation training set/ optimized feature sets;
  - Mutual of predictions via soft voting and stacking to improve the perfomance and reduce poisoning model attacks.
  - Extensive empirical evaluation across malware detection and intrusion classification tasks, measuring both performance metrics and inference consistency; This

ensemble structure improves classification robustness and offers a modular foundation for deployment in dynamic cyber environments.

- Research Gap 3: Despite recent advances, most IDS models remain unsuitable for high-throughput environments due to computational bottlenecks, static detection logic, and lack of adaptive flow control. Traditional detection frameworks are unable to meet real-time latency constraints or scale to modern enterprise or ISP-level networks.

  Approach Direction (Chapter 4): We propose a scalable and low-latency intrusion prevention system called NetIPS, built upon parallelized deep and boosting models integrated with flow-sensing optimization and sandbox analysis. Key features include:

  – An AI-powered detection core using parallel ensemble learning;

  – A flow-sensing strategy that selectively triggers inference based on dynamic traffic characteristics;

  – A hunting malware method that detect malware files transfer between network.

  – A user-space deployment model that minimizes kernel overhead and supports real-time decision-making;

  – Evaluation on emulated network environments to assess performance under strict latency constraints.

  This architecture closes the gap between academic detection models and operational requirements, supporting a proactive and scalable threat response.

Collectively, these three research directions form a cohesive strategy to advance the state of AI-powered cyber threat detection. By addressing dataset, model, and system-level limitations, the dissertation contributes to the design of next-generation detection frameworks that are not only accurate and robust, but also explainable and deployable in real-world scenarios.

1.7 Summary

This chapter has established the foundational context for this dissertation by highlighting the motivation to develop intelligent and resilient AI-based cyber threat detection systems. It began by reviewing the increasing scale and complexity of modern attacks and underscored the shortcomings of traditional intrusion and malware detection approaches, especially their inability to cope with class imbalance, high-dimensional feature spaces, lack of interpretability, and challenges in real-time deployment.

In addition, we also introduced fundamental concepts in intrusion detection, malware classification, feature extraction, and the application of machine learning and deep learning in cybersecurity. Through an extensive survey of recent literature, it highlighted both the progress and persistent gaps in current research, particularly regarding data imbalance, the limitations of single-model approaches, and the absence of scalable real-time AI solutions.

By synthesizing these issues, Chapter 1 identified the key research challenges and framed them as a roadmap for future contributions to the dissertation. Specifically, it defined three interrelated research directions: (i) addressing the challenge of learning from unbalanced and redundant datasets (Chapter 2); (ii) advancing machine leaning model robustness and detection accuracy via mutual ensemble learning (Chapter 3); and (iii) bridging the gap to practical deployment by designing scalable, real-time intrusion prevention architectures (Chapter 4).

In summary, this chapter has identified the key research challenges and objectives in intrusion and malware detection and outlined the main scientific contributions and research roadmap of the dissertation. The mapping between these contributions and the corresponding technical chapters has also been presented, providing a clear structure for the remainder of this work. The next chapter will focus on advanced balancing dataset techniques, addressing the challenges of class imbalance and feature redundancy through intelligent augmentation dataset, adversarial sample generation, and feature selection methods that lay the groundwork for subsequent model development and evaluation.

Chapter 2

Enhancing AI-powered Intrusion Detection with Data Augmentation and Feature Optimization

This chapter focuses on a key challenge for AI-powered intrusion detection systems: the imbalance present in machine learning datasets and the difficulty in handling a large number of features simultaneously. This chapter introduces a novel approach to improving the quality of the dataset for machine learning, starting with advanced GAN-based techniques to increase the number of samples for the minority class and selecting the best representative samples for the majority class. In addition, a SHAP-based feature optimization method is proposed to identify the most important features for training machine learning models. However, in the evaluation section, this chapter primarily assesses the quality of the augmented data, while comprehensive comparisons will be conducted in subsequent chapters. These enhancements not only aim to "clean" and enrich the training dataset but also lay a solid foundation for the following chapters, where machine learning and ensemble models will fully realize their potential on this optimized data platform.

2.1 Problem Statement

In the domain of AI-based cybersecurity analytics, the performance and reliability of detection systems are highly dependent on the quality and structure of the training dataset. Two major challenges arise consistently across intrusion detection and malware classification tasks: (1) severe class imbalance in security datasets and (2) the presence of redundant or non-informative features in high-dimensional representations.

First, currently, many datasets are used in machine learning and deep learning training, such as KDD99, DARPA, CAIDA, ADFA, CSE-CIC-IDS2018, etc. [58]. However, the datasets still have limitations in terms of data quality, such as the difference in the number of classes, duplication, etc. [2]. Therefore, handling the imbalanced dataset is a challenge that should be addressed in designing the intrusion detection system using machine learning/deep learning. In particular, most public datasets in this domain, such as CSE-CIC-IDS2018 [32] for network traffic and EMBER2018 [13] for malware classification, suffer from highly imbalanced label distributions. Minority attack types such as SQL injection, U2R, or

heartbleed often account for less than 0. 1% of the total instances. Traditional oversampling methods, including SMOTE [23] or ADASYN, often fail to capture the intrinsic structure of minority samples and risk introducing synthetic noise that degrades classifier performance. As a result, trained models are prone to high false-negative rates, especially when evaluated on rare or adversarial samples.

Second, a frequent challenge in AI-powered malware detection is the time required to learn the features of the program when the number or size of the features is large, or the number of programs is enormous [53]. Although fewer features may speed up learning, the detection accuracy is likely to decrease [46]. However, selecting a mix of well-chosen characteristics makes it possible to achieve reasonable accuracy, a quick learning curve, and small datasets, although this approach is more complex. The feature spaces extracted from raw binary files tend to be high-dimensional and noisy. For example, EMBER2018 contains more than 2,300 static features, many of which are correlated or irrelevant to malware behavior. Feeding such unrefined data into learning models not only increases the risk of overfitting, but also inflates training time and hinders real-time inference. Furthermore, the lack of explainability in deep or ensemble models has become a bottleneck for practical deployment in security-critical environments.

To address these limitations, this dissertation introduces two complementary contributions:

- We propose augmentation dataset methods that integrates clustering-based compression for majority classes and sample generation for minority classes. The augmentation dataset process is adaptive to feature type and class distribution, with the objective of producing a balanced and diverse training set while maintaining data fidelity.

- We also develop a SHAP-based feature pruning technique termed the Optimized Feature Set (OFS), which employs model-agnostic feature importance to reduce dimensionality while preserving classifier performance. This method enhances model interpretability and reduces training/inference overhead across multiple downstream detection models.

These contributions serve as the foundational input processing components for the detection systems presented in later chapters. They are experimentally validated in this chapter using diverse datasets and classifiers and have been shown to significantly improve the effectiveness and efficiency of AI-based intrusion and malware detection frameworks.

2.2 Approach Direction

To address the two major challenges of class imbalance and feature redundancy in intrusion detection datasets, we introduce augmentation dataset methods aimed at enhancing the learning capacity of AI models in practical cybersecurity contexts. The proposed approach is designed to simultaneously address the problem of insufficient dataset in minority classes, select high-quality samples from majority classes, and identify valuable features in datasets with large numbers of features, issues that are commonly encountered in real-world datasets.

We begin by formally defining the training set RT, partitioned into RT_{maj} (the n samples of the majority class) and RT_{min} (the m samples of the minority class), where the class distributions are typically highly imbalanced. To improve data quality and class balance, we first apply K-Means clustering to compress redundant samples in the majority class, as shown in the following equation:

$$S_{maj} = \sum_{i=1}^{n} Compress(RT_{maj}) \tag{2.1}$$

We generate the minority class using the oversampling technique, then verify and remove noise ($Sample_{noise}$) from the new class samples to increase the number of samples of minority classes by the following formula:

$$S_{min} = \sum_{i=1}^{m} Generate(RT_{min}) \setminus Sample_{noise} \tag{2.2}$$

Finally, we obtain the augmented training set as ATS with the same number as τ of every class label by the following formula:

$$ATS = S_{maj} + S_{min} \tag{2.3}$$

In particular, to further improve sample quality, we incorporate a KMeans-based filtering mechanism that clusters most samples and selectively retains those that lie near the decision boundary, which are more likely to contribute to improved classifier performance. Moreover, we propose a novel augmentation dataset strategy that is both adversarially informed and semantically guided. Unlike conventional oversampling techniques such as SMOTE, which often produce generic and potentially noisy synthetic data, our method uses Wasserstein Generative Adversarial Networks (WGAN) [14] to generate more realistic and context-sensitive attack samples. These GAN-based augmentation methods learn from the underlying structure of minority class instances to synthesize high-fidelity samples that reflect genuine threat behavior patterns. This ensures that the training set is enriched not just quantitatively but also qualitatively with examples from minority classes that are diverse and informative.

On the other hand, we address the issue of feature redundancy and computational inefficiency through feature engineering and SHAP-based feature selection. Given the high dimensionality of network flow data, many features may be irrelevant or even misleading for model learning. By computing SHAP (SHapley Additive exPlanations) [73] values across multiple models, we identify the most influential features that consistently contribute to accurate predictions. This results in a reduced feature set that maintains semantic richness while lowering model complexity and inference latency. Given an ML model m_i to predict the output y of a data sample x. The Shapley value $\phi_j^{(i)}$ for the j-th feature in the data sample x is calculated by determining the difference between the model prediction with feature f_j included and its prediction without it by Equation 2.4:

$$\phi_j^{(i)} = \sum_{S \subseteq \{1,2,\ldots,N\} \setminus \{j\}} \frac{|S|!(N - |S| - 1)!}{N!} \Big(m_i(x_{S \cup \{j\}}) - m_i(x_S) \Big) \tag{2.4}$$

where N is the number of features; S is a subset of the set of features not containing the j-th feature; x_S is the data sample x containing only the features in set S; $x_{S \cup \{j\}}$ is the data sample x containing the j-th feature; $m_i(x_S)$ is the prediction value of the model when considering only the features in set S; $m_i(x_{S \cup \{j\}})$ is the prediction value of the model when including the j-th feature.

After computing the SHAP value for each data sample, Equation 2.5 is used to determine the importance value of feature j over the entire dataset as follows:

$$I_j = \frac{1}{n} \sum_{i=1}^{n} |\phi_j^{(i)}| \tag{2.5}$$

where n denotes the number of data samples.

We use k ML models, $m_1, m_2, m_3, ...m_k$, and apply the formula in Equation 2.5 to calculate the importance score I_j^k for each feature j as determined by model m_k, where k ranges from 1 to k. We then identify all features that satisfy the condition $I_j^k \geq \tau$, with τ being a predefined threshold. Next, we select the features chosen by the individual model. These features are considered essential for training the ML models. Finally, we reconstruct the dataset using only the important features. This method creates a new dataset with a reduced number of features, and we use it to train and test the ML models.

The enhanced datasets both augmented and dimensionally reduced are used to train a variety of AI models, including deep neural networks and ensemble-based classifiers. These models are evaluated under multiple performance metrics, such as accuracy, precision, recall, and F1 score, with a specific focus on improvements in minority class detection. The results demonstrate that the proposed approach not only mitigates the impact of class imbalance but also enhances the generalization and stability of the model.

Algorithm 2.1 AugDS: Build the Augmented Dataset

Input: F - Raw Dataset, represented by a list of feature vectors; K - scaling factor

Output: T - Augmented Dataset;

$L \leftarrow ComputeLabels(F)$ ▷ Get all labels of dataset F

$F \leftarrow Normalize(F)$ ▷ normalizing all feature vectors

$ES = EditedNearestNeighbours(RT, |L|)$ ▷ determining the easy sets ES by finding L nearest neighbours samples

$DS = RT \setminus ES$ ▷ difficult set DS is the rest of RT

$Majors, Minors \leftarrow ComputeMajMin(DS)$

$S_{maj} \leftarrow \emptyset, S_{min} \leftarrow \emptyset$

for each $M \in Majors$ **do** ▷ Compression Step

  $C \leftarrow Clustering(M, |L|)$ ▷ computing the centroids C of $|L|$ clusters by using KMeans algorithm

  $M \leftarrow Compress(M, C, \tau)$ ▷ compressing majority samples using centroids C of L clusters

  $S_{maj} \leftarrow S_{maj} \cup M$

end for

for each $M \in Minors$ **do** ▷ Zooming Step

  for each $m \in range(K, K + \frac{number}{N_{S_{min}}})$ **do** ▷ Zooming Step, $N_{S_{min}}$ is number sample in S_{min}.

    $M \leftarrow Zoom(m)$ ▷ zoom range is $[1 - \frac{1}{K}, 1 + \frac{1}{K}]$ on both continuous and categorical features.

    $S_{min} \leftarrow S_{min} \cup M$

  end for

end for

$T = ES \cup S_{maj} \cup S_{min}$ ▷ synthese of new dataset T

return (T)

2.3 Training Dataset Augmentation

2.3.1 Difficulty-Aware-based Data Augmentation

With vast datasets about network traffic, the number of network attacks is typically extremely low, resulting in a significant disparity between the labeled dataset about cyberattack types and the benign network traffic. Insufficient network attack samples significantly hinder the development of predictive models. In addition, the amount of benign network traffic is excessive. It makes the model too difficult to predict, leading to a high rate of false positives.

To improve the performance of ML models, we must minimize the amount of benign

network traffic and increase the number of network attack patterns used in the training phase. It helps the ML model improve accuracy and avoid overlearning about a specific label. Consequently, we propose a method based on the concept of the DSSTE algorithm proposed by [68] to augment the training dataset. Our algorithm is named AugDS and is shown in Algorithm 2.1.

In particular, the algorithm is specifically designed to address two core challenges commonly found in network intrusion datasets: severe class imbalance with an overwhelming number of benign samples compared to rare attack instances and the presence of redundant data in the majority class. The algorithm proceeds in several main phases, employing a partitioning strategy based on sample difficulty, and subsequently applying tailored compression or augmentation techniques for each group.

Initially, the entire raw dataset was normalized to ensure feature compatibility and facilitate subsequent operations such as clustering and distance calculations. The algorithm then uses the Edited Nearest Neighbors (ENN) method to partition the dataset into two subsets: the 'easy set', which contains samples whose labels are consistent with those of their nearest neighbors (and are thus easy for machine learning models to classify), and the 'difficult set', which consists of samples lying close to class boundaries or frequently misclassified by conventional learners.

Within the difficult set, Algorithm 2.1 further distinguishes between majority and minority classes based on their sample counts. For the majority classes, a data compression procedure is applied. This involves using the KMeans clustering algorithm to group similar samples and represent each group by its centroid. In this way, redundant benign samples are reduced to a manageable, representative subset, which helps mitigate overfitting and excessive bias toward the majority class.

In contrast, for minority classes, the algorithm adopts a "zooming" strategy. Here, new synthetic samples are generated for each original minority-class sample by introducing controlled perturbations to their feature values within a specified range, determined by the scaling factor K. This ensures that the minority classes are enriched with realistic and diverse examples, enhancing the model's ability to recognize rare or emerging attack patterns. Importantly, this augmentation is designed to maintain the original data distribution and avoid the introduction of artificial noise.

Finally, Algorithm 2.1 synthesizes the augmented dataset by combining the easy set retained in its original form to preserve representative characteristics of all classes, compressed majority samples and zoomed minority samples. The resulting training dataset is therefore more balanced, rich in information, and tailored to support robust machine learning models for intrusion detection. By focusing on hard-to-classify regions and adaptively enhancing

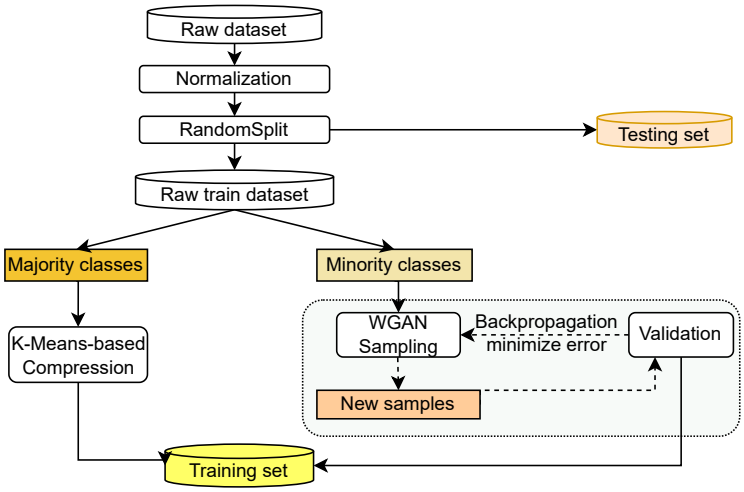


Figure 2.1: Architecture of AWGAN-based Data Augmentation

class balance, AugDS substantially improves detection performance, especially for rare but critical attack types, while also reducing the risk of overfitting on the dominant benign traffic.

2.3.2 AWGAN-based Data Augmentation

In another approach, to solve the issue of the unbalanced dataset in IDS, our augmented WGAN method, AWGAN, generates realistic samples for minority classes using WGAN. In the meantime, the majority classes in a large number of samples, such as benign flows, can occasionally hinder the performance of ML models. Therefore, it is essential to maintain significant samples in the majority classes. We also use the K-Means algorithm in conjunction with WGAN to eradicate ineffective samples. Consequently, the AWGAN is depicted in Figure 2.1 and is described formally in Algorithm 2.2. In this algorithm, to augment the training set, we perform the following steps:

Step 1 - Dataset Reprocessing: carry out dataset normalizing; eliminate noise and duplicated raw data on the dataset; and split it into the training and testing sets, preset by a ratio of 7:3, respectively. We use τ as a constant to determine the maximum samples of the label class in the training set. The testing set is used for the evaluation in the DL models of our project.

Step 2 - Finding Majority and Minority Classes: The training set is separated into the majority class and minority class from the initial/original train dataset. We compressed the majority class and utilized the oversampling approach to create data for the minority class. Consequently, the total number of classes in the training set is equal.

Step 3 - Compressing Majority Classes: we reduce the number of the majority class by proposing a method inspired by the idea of the DSSTE [68] algorithm to augment the

Algorithm 2.2 AWGAN: Create the Training & Testing Sets by Augmented WGAN

Input: F - Raw Dataset, represented by a list of feature vectors.
r - ratio between training and testing sets; default is 7:3.
τ - maximum samples in a label.
Output: T - Training Set; V - Testing Set.

1: $L \leftarrow GetLabels(F)$ ▷ Get all labels of dataset F.
2: $F \leftarrow Normalize(F)$ ▷ Normalize all feature vectors.
3: $(RT, V) \leftarrow SplitTrainTest(F, r)$ ▷ Split F randomly into the raw training set RT and testing set V with ratio of r.
4: $(S_{maj}, S_{min}) \leftarrow GetClasses(RT)$ ▷ Determine majority classes (S_{maj}) and minority classes (S_{min}) from RT
5: $T \leftarrow \emptyset$
6: **for each** $M \in S_{maj}$ **do** ▷ Compression each majority class
7: $C \leftarrow Clustering(M, |L|)$ ▷ Compute the centroids C of $|L|$ clusters by using ENN
8: $M \leftarrow Select(M, C, \tau)$ ▷ Compress majority samples using C of L clusters
9: $T \leftarrow T \cup M$
10: **end for**
11: **for each** $M \in S_{min}$ **do** ▷ Generate samples for minority classes by WGAN
12: **while** $|M| < \tau$ **do**
13: $S \leftarrow WGAN\_Sampling(M)$ ▷ Generate new samples
14: $M = Denoise(M, S)$ ▷ Eliminate noise samples
15: **end while** ▷ Repeat until get enough samples τ.
16: $T \leftarrow T \cup M$ ▷ Add realistic samples to T
17: **end for**
18: **return** (T, V)

dataset. We use the Edited Nearest-Neighbor (ENN) algorithm to obtain the majority of labels that are frequently difficult to classify due to their proximity. Using a clustering algorithm, we then compressed every label class in the majority class to reduce the number of label classes. We eventually obtain τ for each label class in the majority class and append the majority class samples to the training set.

Step 4 - Augmenting Realistic Data for Minority Classes: We balance the minority class using the oversampling model based on WGAN[14], which uses attack data to generate simulated attack samples. Then we validate these new samples using the train & test model built from the actual dataset to test the output of the sampling model. Depending on the result of the testing phase for the new attack samples as a testing set, the oversampling model will be backpropagation to minimize error. It also removes the noise, which is new attack samples that failed to be classified. This step will repeat until the train & test model

can't define actual and simulated attack data, and the number of every label class in the minority class is equal to τ. Moreover, this step generates more realistic attack samples, and these new attack samples may be similar to those for other attacks.

Step 5 - Results: Finally, we obtain a new training set, which contains the number of every label class that is the same as τ, and use this training set to train and the testing set in Step 1 to test our models. Thus, it helps us to obtain a better AI model.

2.4 Feature set Optimization

2.4.1 Feature Extraction and Cleaning

In ML, feature extraction and cleaning are crucial steps to choose and improve a subset of its features. This step can drastically reduce computational costs and eliminate pointless data processing during training. Generally, datasets contain raw, unclean, or imbalanced data. There are also duplicate records and categorical data. Therefore, we need to preprocess all datasets by cleaning, removing records, and transforming categorical data. These records can negatively impact the training process, often leading to model overfitting. We adopt simple methods to handle missing values of the attributes, such as removing rows containing null values or duplicating entries.

In our method, assuming that the input D_o is a list of records, where each record r is a list or a tuple of values by columns, we initialize an empty list $S_r \leftarrow \{\}$ to store the unique records. Then iterates through each record in the input data, and if the record is not already in the S_r list, it adds the unique record, r, to that list. Finally, it returns the list $D_o\,[S_r]$, which contains only the unique records from the input data.

In addition, the set of attributes in the training samples also influences the quality of the AI models. Therefore, eliminating redundant attributes that do not affect malware detection is one of the essential tasks. We used an AutoML toolkit called AutoGluon [31] to analyze, evaluate, and remove redundant attributes.

2.4.2 Feature Vectorizing

We must transform raw data, often in JSON format, into numerical vectors to prepare dataset features for AI model training. Many famous models, such as the GBM family model, only accept input as a number vector, making the latter unsuitable for training AI models. Therefore, we perform vectorization to obtain binary format features and store them in CSV for future use. Since most of the features of the dataset have unique values and cannot be easily categorized, the hashing technique is suitable to retain the data

characteristics [5]. Consequently, we employ the feature hash technique in this work to vectorize the features into a feature vector. First, we will need to define two hash functions:

- The *kernel hash* $h : T \to \{1, 2, ..., n\}$. The kernel hash function h maps tokens from the set T to a set of indices from 1 to n. It is defined as follows:

$$h(t_i) = index \quad \text{if } t_i \in T$$

  where t_i is the i^{th} token in set T; $index$ is the index of the token t_i in the set of indices.

- The *sign hash* $\zeta : T \to \{-1, +1\}$. The sign hash function ζ maps tokens from the set T to the set $\{-1, +1\}$. It is typically used to represent positive and negative values of tokens with different sign values. Specifically:

$$\zeta(t_i) = \begin{cases} -1 & \text{if } t_i \text{ has a negative or zero value} \\ +1 & \text{if } t_i \text{ has a positive value} \end{cases}$$

  where t_i is the i^{th} token in the set T.

Then we define the feature hashing function. The Feature Hashing function ϕ transforms sequences of tokens from set T^* into a feature vector in the n-dimensional real space $\mathbb{R}^n$. It is defined in Equation 2.6 as follows:

$$\phi : T^* \to \mathbb{R}^n, \phi(t_1, ..., t_k) = \sum_{j=1}^{k} \phi(t_j) \tag{2.6}$$

where T^* is the set of all finite strings that contain tokens in T; $t_1, ..., t_k$ are tokens in the sequence; and k is the number of tokens in the sequence.

The Feature Hashing function can be equivalently represented by Equation 2.7 as follows:

$$\phi(t_1, ..., t_k) = \sum_{i=1}^{n} (\sum_{j:h(t_j)=i} \zeta(t_j))e_i \tag{2.7}$$

where $\zeta(t_j)$: Sign value of j^{th} token; e_i: Unit vector with value 1 at the index i and 0 elsewhere; $h(t_j)$: Index of the j^{th} token mapped by the kernel hash function h; and n: Number of indices.

We label and encode all category characteristics, except the last characteristic, before applying the normalization procedure to make ML readable. The label encoding transforms the categorical features into numerical values (the integer values begin at '0'). We use the Keras library to encode categorical data into numerical data. Categorical features are converted to binary features that are "one-hot" encoded, which means that a feature represented by that column receives a 1 if it is one of the characteristics converted to binary features. Otherwise, it gets 0.

2.4.3 Feature Normalization

Suppose the feature values are more similar in the ML algorithms. If the data points or feature values are highly dissimilar, it will take longer for the algorithm to understand the data, resulting in lower accuracy. To address this issue, we used StandardScaler from the Scikit-Learn library to normalize the feature range. It is particularly effective for normalizing datasets with numerous features, outperforming other methods such as Min-Max, Z-Score, and Robust Scaler. Normalization is used in the output obtained after the label encoding is applied to ensure that each independent feature has a mean of 0 and a standard deviation of 1, expressed by Equation 2.8 as follows:

$$X_{\text{new}} = \frac{X - \mu}{\sigma} \tag{2.8}$$

where X is the original feature sample, X_{new} is the standardized feature sample, μ is the mean of the feature values, and σ is the standard deviation of the feature values.

Normalizing the features centers them around zero with a unit standard deviation, facilitating the ML algorithm's learning process. This normalization technique helps speed up convergence and improve the model's overall performance.

2.4.4 SHAP-based Feature Set Optimization

Our method takes as input the training data X, a matrix of size $n \times m$, where n is the number of samples, m is the number of input features, and l is the number of labels. The method seeks to produce a chosen subset of features, OFS (Optimizing Features using SHAP), containing the most essential and practical features of the original datasets. The process starts by initializing an empty list $FS \leftarrow \{\}$ to store the important features. We train ML models $M_1, M_2, ...M_k$ in the training dataset. We then chose the best models based on their results. Next, we use SHAP to explain and calculate the importance of each feature, X_i, in the test data. We determine the importance coefficient, p_i, for each feature of the data. The feature X_i is added to the list FS if its importance coefficient, p_i, is greater than or equal to a predefined threshold τ. Once all features have been evaluated, the list FS is sorted in descending order based on the importance coefficients τ. Next, we identify the most significant features of the sorted list that meet the threshold. We then return the important feature subset FS as the final output OFS for training and testing the ML model. Consequently, the overall pseudocode for optimizing the set of features using SHAP is presented in Algorithm 2.3.

In summary, the suggested method uses SHAP and ML models to figure out the importance weight of each feature in the dataset. Then it uses a threshold, τ, to pick the most

Algorithm 2.3 OFS: **O**ptimizing **F**eature **S**et Using **S**HAP

Input: DS - dataset with the feature set F; M - m AI models; τ - threshold to drop features.

$X, y \leftarrow DS$ ▷ Get dataframes for features and labels
$X \leftarrow \text{Normalize}(X)$ ▷ Normalize all features to [0,1]
$FS \leftarrow \emptyset$ ▷ Init the feature set list.
for each $m \in M$ **do** ▷ Determine the feature importance for each AI model m.
    $AI \leftarrow m.fit(X, y)$ ▷ Train m using the dataset.
    if m is a boosting model **then**
        $shap\_values_m \leftarrow SHAP.TreeExplainer(m)$ ▷ Compute the SHAP values of all features based on decision tree model.
    else
        $shap\_values_m \leftarrow SHAP.DeepExplainer(m)$ ▷ Compute the SHAP values of all features based on DL model.
    end if
    $FS.push(shap\_values_M)$ ▷ Push the Shapley values of the model M into the list FS.
end for
$OFS \leftarrow \emptyset$
for each $f \in F$ **do**
    $shap\_values \leftarrow FS[f]$ ▷ Get SHAP values of feature f on all models M.
    if $shap\_values \geq \tau$ **then**
        $OFS \leftarrow OFS \cup f$ ▷ Consider f being important and add to OFS in the case of all its SHAP values $\geq \tau$.
    end if
end for

Output: OFS - Optimized Feature Set.

important and influential features based on their importance coefficients.

2.5 Experiments and Evaluation

The appliance server used in our experiments has a configuration of 2 x Intel Xeon-Platinum 8160 (24-cores); 384GB DDR4 RAM; NVIDIA Tesla T4 16GB; SmartNIC Napatech NT40E3-4-PTP to validate our proposed method in Chapter 2, Chapter 3 and Chapter 4. Suricata v6.0.3 is tuned, and new components are added to control traffic flows and implement the deep inspection strategy described in Chapter 4 . We use Python version 3.8 as a programming language with the following libraries and frameworks: Fastai V2.3.0, Scikit-learn V0.24.1, Matplotlib V3.4.1, Pandas V1.2.3, Numpy V1.20.2.

2.5.1 Dataset Preparation

In this section, we prepare the datasets for experiments to demonstrate the contributions of our proposed augmentation dataset algorithms. These datasets will also be used for experiments in Chapter 3 and Chapter 4. In our experiments, the data preparation process is conducted according to the following steps:

1. Define redundant attributes that are not needed in AI models based on the Autogluon framework [57].

2. Remove redundant attribute columns from the original data set (still keeping the original number of samples).

3. Delete the NaN and duplicate samples in the dataset after cleaning redundant features.

4. Perform a random selection of the testing samples according to the stated strategy. The output of this step is the testing set and the raw training set.

5. Generate and augment the raw training set based on our proposed augmented dataset algorithms. The output of this step is the augmented training set.

The datasets were prepared following the five basic steps described above. To evaluate our proposed methods, we constructed the corresponding datasets for experimental assessment, as detailed below:

- DS1: We selected CSE-CIC-IDS2018 and NSL-KDD, two well-known benchmark datasets, to evaluate the effectiveness of Algorithm 2.1, the output constitute DS1. In particular, based on the distribution of classes in CSE-CIC-IDS20- 18, we notice that there are six classes with more than 20,000 samples and six classes with fewer than 20,000 samples. With NSL-KDD, two classes have more than 20,000, and three classes have less than 20,000. Therefore, we decided to choose a threshold of 20,000 samples to augment the training datasets with Algorithm 2.1.

  For testing datasets, to ensure objectivity, they are built by randomly selecting samples from the original datasets. With a maximum of 20,000 samples for each label, we selected a testing/training sample ratio of 3:10 and then had a maximum of 6,000 samples for each class.

  To augment SQL-injection detection, we also built a testbed system, as shown in Figure 2.2, to add more detection ability. In this testbed, we deploy all the necessary standard equipment on the DMZ network, including routers, firewalls, switches, and web servers. We use Kali Linux to perform SQL-Injection attacks on the attack

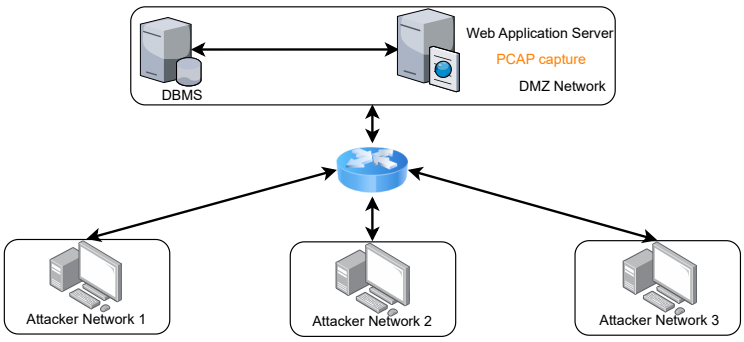


Figure 2.2: Testbed Architecture for SQL-Injection Attack Generation

network. We also use the Wireshark tool to capture network traffic and then use CICFlowMeter to extract traffic flow features such as CSE-CIC-IDS2018.

Finally, based on our dataset preparation process, we obtain two augmented datasets DS1, illustrated in Table 2.1. Note that DS1 datasets will be comprehensively evaluated in Chapter 3.

- DS2: We also selected CSE-CIC-IDS2018 and NSL-KDD to experimentally evaluate the effectiveness of Algorithm 2.2, the output constitute DS2. The CSE-CIC-IDS2018 has a total of 12 classes labeled by [Benign, Infiltration, Bot, DDos-HOIC, DoS-GoldenEye, DoS-Hulk, DoS-Slowloris, DDoS-LOIC-UDP, BruteForce-Web, BruceForce-XSS, SQL-Injection]. The *Bengin* samples in this dataset are much larger than the attack samples. It has enough samples for *Bot* and $DDOS - HOIC$, while $SQL - Injection$ and $BruteForce - Web$ have very small attack samples.

  Related to NSL-KDD, it has four classes: [DoS, Probe, U2R, R2L]. Like CSE-CIC-IDS2018, we also observe that *Benign* samples dominate the attack samples. However, *DoS* has larger samples, while the other attacks, i.e. $R2L$ and $U2R$, suffer from very low samples.

  For both datasets, the samples of each label (in each dataset) are largely imbalanced. To enhance these datasets, the AWGAN algorithm is applied to each class, where the parameter r (ratio between training and testing sets) is set to 7:3, and τ (maximum samples in a label) set to 20,000. Note that, with AWGAN setting with these parameters, the class having more than 20,000 samples will be "compressed," selecting only 20,000 samples. Meanwhile, the class with less than 20,000 samples will be "zoomed" and "generated" with more realistic samples up to 20,000. A point that should also be emphasized here is that for classes with less than 20,000 samples, the training/test set division must be performed before applying WGAN. For example, with the "*BruteForce-Web*" class of CSE-CIC-IDS2018 with 261 samples, the test set

Table 2.1: Dificulty-Aware-based Data Augmentation

Class	Original	Train	Test
CSE-CIC-IDS2018			
Benign	4,360,029	20,000	6,000
Bot	282,310	20,000	6,000
DDoS-HOIC	668,461	20,000	6,000
DoS-GoldenEye	41,455	20,000	6,000
DoS-Hulk	434,873	20,000	6,000
Infiltration	160,604	20,000	6,000
SQL-Injection	26,797	20,000	6,000
DoS-SlowHTTPTest	19,462	13,623	4,491
DoS-Slowloris	10,285	14,826	2,373
DDoS-LOIC-UDP	1,211	1,588	279
BruteForce-Web	253	978	58
BruteForce-XSS	151	106	35
NSL-KDD			
Benign	61,343	20,000	6,000
DoS	39,927	20,000	6,000
Probe	8,153	20,000	1,881
R2L	697	4,467	161
U2R	36	36	8

will be randomly selected at $30\% * 261 \simeq 78$ samples. The remaining 183 samples will be fed into AWGAN to generate up to 14,000 samples.

Similarly, after utilizing our AWGAN, we obtained the augmented training sets for training AI models and test sets for evaluating them. Finally, Table 2.2 summarizes the number of samples for each class of both datasets. In this table, the Original column represents the number of original samples after removing NaN and duplicate values, the Train column is the number of samples augmented by AWGAN, and the Test column shows the original samples.

- DS3: We selected EMBER2017, EMBER2018, and BODMAS, which are widely recognized datasets for malware detection, to experimentally evaluate the effectiveness of Algorithm 2.3, the output constitute DS3. In our work, we extract and represent

Table 2.2: AWGAN-based Data Augmentation

Label	Original	Train	Test
CSE-CIC-IDS2018			
Benign	4,360,029	14,000	6,000
Infiltration	160,604	14,000	6,000
Bot	282,310	14,000	6,000
DDoS-HOIC	668,461	14,000	6,000
DoS-GoldenEye	41,455	14,000	6,000
DoS-Hulk	434,873	14,000	6,000
DoS-SlowHTTPTest	13,067	14,000	4,082
DoS-Slowloris	6,977	14,000	2,093
DDoS-LOIC-UDP	1,120	14,000	336
BruteForce-Web	261	14,000	78
BruteForce-XSS	97	14,000	29
SQL-Injection	53	14,000	17
NSL-KDD			
Benign	61,343	14,000	6,000
DoS	39,927	14,000	6,000
Probe	8,333	14,000	2,500
R2L	637	14,000	191
U2R	40	14,000	12

a PE file using the Library to Instrument Executable Formats (LIEF). We utilize this library for both the EMBER 2017 and 2018 datasets. The library contains nine groups of raw static features, totaling 2381 features. These groups include a byte histogram with 256 features, a byte-entropy histogram with 256 features, and string information with 104 features. The generic file includes ten features and header data with 62 features. The import functions category has 1280 features, while the section information has 255. Data directories have 30 features, while export information has 128 features.

Moreover, the EMBER datasets contain both training and testing sets. The training dataset provides benign, malicious, and unlabeled data in three categories as $0, 1$, and -1. However, the testing set within the dataset contains no unlabeled data. To strike a compromise, we excluded unlabeled samples from further processing and rebuilt the

training set using only labeled data. This step will equalize the training and testing sets and improve the performance of the ML model.

We also used the BODMAS dataset [108] for evaluation. The BODMAS dataset was curated from a large corpus of PE files collected between August 2019 and September 2020. Each sample in BODMAS is labeled malicious 1 or benign 0. Feature extraction for BODMAS was performed using the LIEF, consistent with the EMBER dataset. Each sample is represented by a 2381 dimensional feature vector, encompassing various static characteristics of PE files. Three datasets for our experiment evaluation:

- The EMBER2017 dataset contains malware and benign samples collected from 2017 and older, including 300K of each label (*0* for benign/*1* for malware) in the training dataset, 100K of each label in the test dataset, and 300K of unlabeled samples, which are marked as label *-1*. The total EMBER 2017 contains 1,100K samples.
- The EMBER2018 dataset, collected in 2018, includes 300K of each label (*0*/*1*) in the training dataset, 100K of each label in the test dataset, and 200K of unlabeled samples, which are marked as label *-1*. EMBER2018 totally contains 1,000K samples.
- The BODMAS dataset includes 57,293 malware samples from 67 malware families and 77,142 benign samples. All samples in the BODMAS dataset are labeled as either *0* (benign) or *1* (malicious). The total BODMAS dataset contains 134,435 labeled samples. Moreover, BODMAS, although recent, lacks standardized feature definitions and includes only feature vectors without benign binaries due to copyright issues, limiting full reproducibility.

After releasing the original dataset in 2017 (we will call this EMBER2017), Anderson et al. also released a new updated version in 2018 (often called EMBER2018). Compared to the former version, EMBER2018 has many improvements, including:

- *New data*: EMBER2018 only contains new data collected in 2018. The EMBER2017 dataset only includes all pre-2018 samples.
- *Duplicates and Outliers cleaned*: The authors identified and eliminated abnormally low- and high-density data using a fast cover tree. The authors classified the samples as outliers if they did not geometrically contribute to the data, as determined by a weighted fractal dimension. Duplicates were nodes in the tree whose radius was less than the L2 norm implementation error rate.
- *New features included*: For every sample in EMBER2018, MD5 was present and also provided the AVClass label for malicious samples.

– *Harder Dataset*: This new dataset prioritized the addition of NET-packed software and 32- and 64-bit samples along with benign and malicious samples. This dataset also relied on the trusted tag in VirusTotal to prove benign origins. The trusted tag on VirusTotal helped the authors incorporate samples from various vendors that had false positives. The dataset prioritizes samples of significant malware families from 2018. Some example families include Wannacry, Emotet, Qbot, Gh0st, Brambul, Zbot, Kovter, Samsam, Mirai, Coinminer, Nanocore, Cerber, Ursnif, Redyms, Ramdo, Tinyloader, and Trickbot.

As a result, based on technique in Section 2.4. Experiments using the EMBER2017 and EMBER2018 datasets identified 90 redundant characteristics from the 2381 attributes representing each PE sample. The result of the feature cleaning step produced a set of 2291 important attributes to form the feature vectors. We removed them; their column IDs include: "627, 636, 638, 648, 650, 652, 659, 663, 666, 667, 669, 670, 672, 673, 674, 675, 676, 849, 859, 862, 867, 883, 891, 894, 896, 898, 899, 900, 904, 905, 908, 909, 910, 912,917, 919, 920, 922, 923, 924, 926, 931, 932, 934, 936, 937, 629, 630, 635, 651, 653, 671, 861, 864, 871, 878, 884, 889, 890, 907, 911, 913, 914, 918, 925, 933, 935, 938, 939, 942, 962, 974, 989, 997, 1022, 1042, 1044, 1052, 1053, 1056, 1057, 1069, 1086, 1120, 1125, 1151, 1173, 1179, 1184, 2367".

Furthermore, we performed the optimization feature method based on Algorithm 2.3 for EMBER2017 and EMBER2018. Our method chooses a set of thresholds to compute the optimal number of features for the classification task. We use six thresholds: 0.1, 0.075, 0.05, 0.25, 0.01, and 0.001. For each threshold, features with SHAP values $\geq$ the chosen threshold are selected, shown as Figure 2.3. We found that the threshold of 0.025 gives the best result, shown as Figure 2.4. This threshold yields 170 (all the features selected by the four models) and 565 (the union of all the features selected by the four models) important features out of a total of $2,381$ features in the original dataset. We then save this set of features as a new dataset for use in the next steps.

Moreover, following the method in Section 2.4, we applied the BODMAS dataset. Using six thresholds (0.1, 0.075, 0.05, 0.025, 0.01, and 0.001). Our analysis revealed that a threshold of 0.01 provides an optimal balance, resulting in two refined feature sets, shown in Figure 2.5. The intersection set selected by the four models (4 features). The union set was selected by at least one model (165 features). Figure 2.5 illustrates the variation in the accuracy of four models under different SHAP-based feature selection thresholds. The results indicate that tree-based models maintain relatively stable performance, with only a slight drop at the 0.075 threshold compared to lower thresholds. In contrast, CNN is highly sensitive: at the 0.075 threshold, its accuracy decreases dramatically to the lowest level

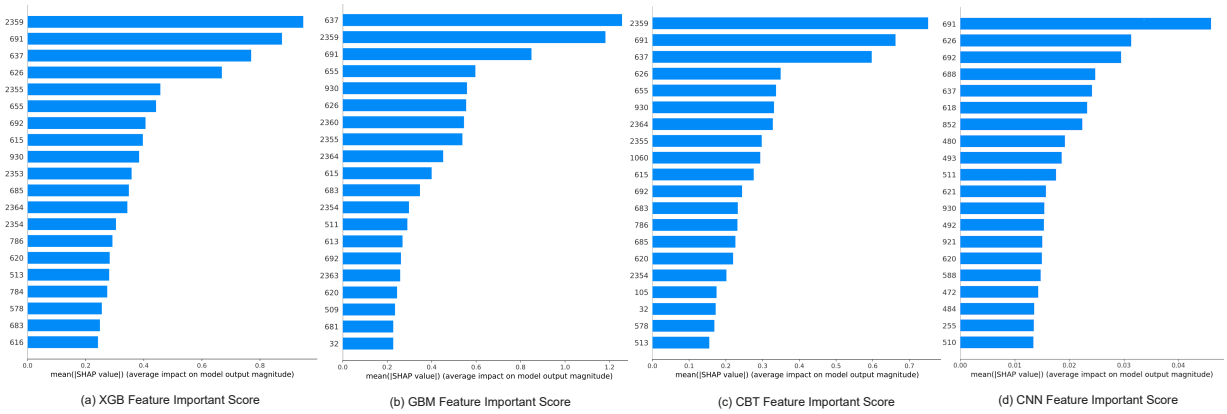


Figure 2.3: SHAP-based Feature Important Scores on EMBER2018 Dataset

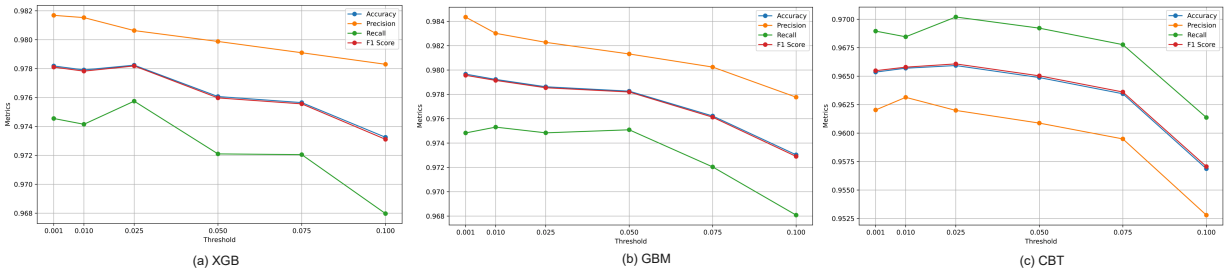


Figure 2.4: Threshold-based Performances on EMBER2018 Dataset

(95.5%) but quickly recovers as the threshold decreases to 0.05 and below. This observation indicates that the 0.075 threshold is unsuitable, especially for CNN, as it removes too many important features. The thresholds 0.05, on the other hand, produce a higher and more stable accuracy, providing a better balance between retaining the relevant features and maintaining the classification performance.

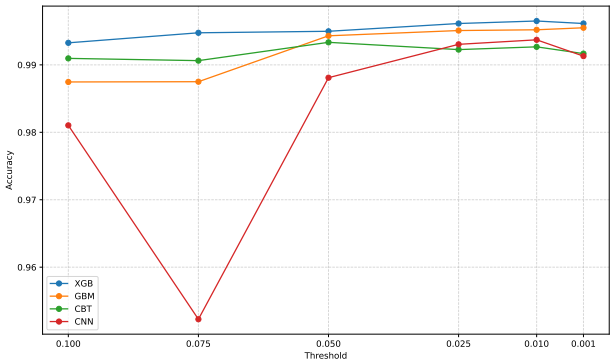


Figure 2.5: Threshold-based Performances on BODMAS Dataset

2.5.2 Results and Evaluation

In this section, we present the results obtained from the experiments to prove the performance of our proposed method. To evaluate the effectiveness of the proposed algorithms, we use the following scenarios:

- S1: We thoroughly evaluate Algorithm 2.1 on the DS1 to investigate its effectiveness in addressing class imbalance. Our goal is to ensure a more equitable representation of all types of attack in the training set, thereby improving the model's ability to accurately detect frequent and rare intrusions.
- S2: The AWGAN Algorithm 2.2 is evaluated on DS2 to rigorously assess its ability to generate realistic and diverse synthetic samples for minority classes. By directly addressing the data imbalance at the class level, this scenario enables us to verify whether adversarial augmentation via AWGAN can significantly boost detection rates.
- S3: The OFS Algorithm 2.3 is examined using the DS3, with a focus on static malware detection tasks. The evaluation comprises three phases: (1) establishing a baseline with the original dataset, (2) measuring improvements with the augmented dataset, and (3) performing a comparative performance analysis. This scenario demonstrates the practical impact of advanced feature selection on malware detection.

2.5.2.1 S1 Results

Originally, CSE-CIC-IDS2018 were heavily imbalanced; for example, the *Benign* class in CSE-CIC-IDS2018 had more than 4 million samples, while classes such as *DDoS-LOIC-UDP* and *BruteForce-Web* had only 1,211 and 253 samples, respectively. In NSL-KDD, *Benign* began with 61,343 samples, while *U2R* had just 36.

After applying our balancing method, most classes were increased or compressed to up to 20,000 training samples. For instance, in DS1, *DDoS-LOIC-UDP* rose to 1,588, *BruteForce-Web* to 978, and *Benign* was reduced to 20,000. In NSL-KDD, *Probe* was augmented to 20,000, *R2L* to 4,467, while *U2R* remained at 36 due to data scarcity.

These changes, summarized in Table 2.1, led to a much fairer distribution between classes and provided a solid foundation for robust model training. The effect is also visible in the t-SNE visualizations : before the balancing shown in Figure 2.6a and Figure 2.7a, majority classes dominate the feature space and the minority classes are barely visible. After balancing shown in Figure 2.6b and Figure 2.7b, the clusters for all classes are more evenly distributed, with the minority classes forming clearer and more distinct groups.

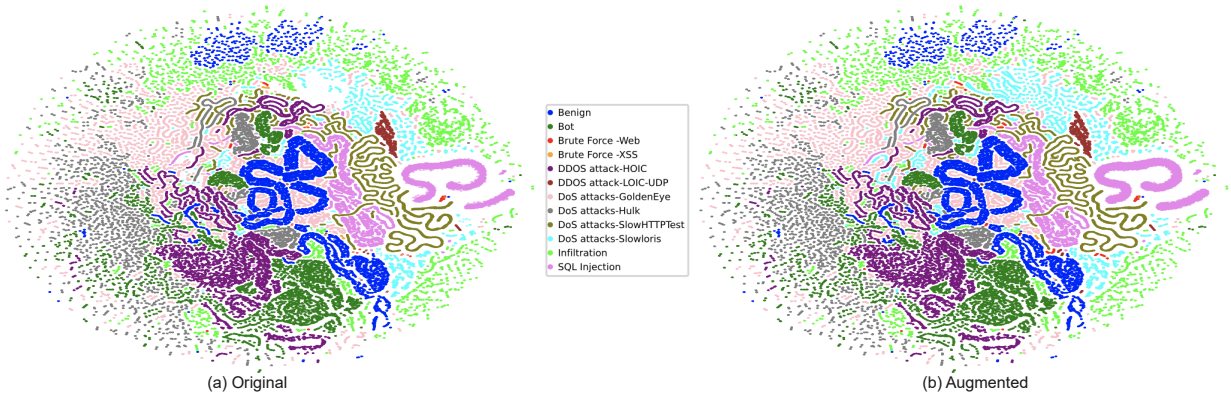


Figure 2.6: Difficulty-Aware-based Visualization of CSE-CIC-IDS2018 Training Set

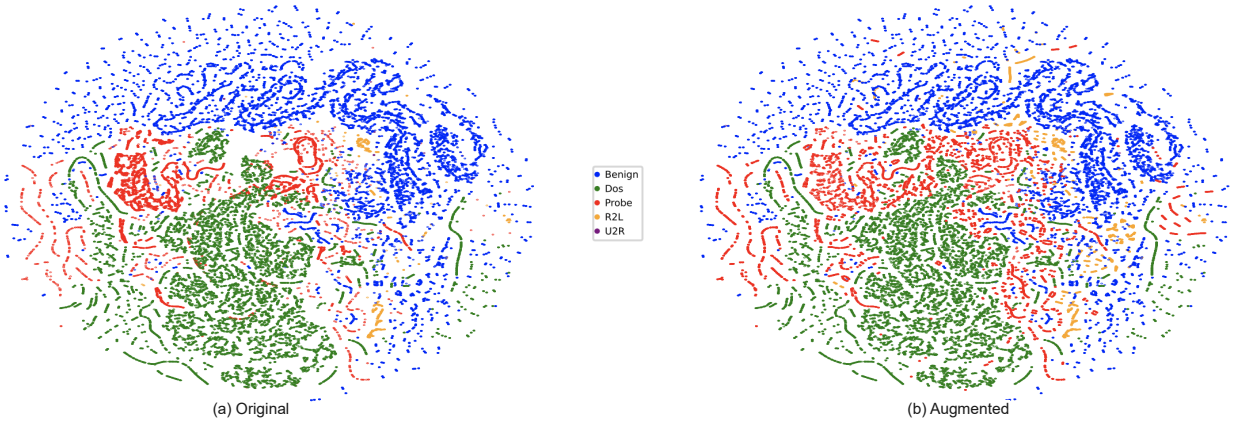


Figure 2.7: Difficulty-Aware-based Visualization of NSL-KDD Training Set

This improvement in class distribution and separation supports our observed performance gains shown in Table 2.3, where DNN, XGB, and GBM achieved high precision, precision, recall, and F1, all above 98. 8% in all classes. AUC values also reached nearly 100%, demonstrating the effectiveness of the method, especially for minority attack types.

However, a limitation remains: for some extremely minority classes (e.g., *BruteForce-Web*, *BruteForce-XSS*, *U2R*), it was not feasible to increase their size to 20,000 due to the lack of original data, as shown in Table 2.1. This is also reflected in the t-SNE plots, where these classes still appear as smaller, less compact clusters. Thus, while overall balance and detection improved significantly, detecting the rarest attacks continues to be a challenge and warrants further research.

2.5.2.2 S2 Results

After applying AWGAN, we obtain balanced, augmented training sets for model training and separate test sets for evaluation. Table 2.2 summarizes the number of samples per class in both datasets. In this table:(1) The "Original" column shows the sample count after cleaning (removal of NaN and duplicates); (2) The "Train" column gives the number of

Table 2.3: Evaluation of AI models on Dificulty-Aware-based Data Augmentation (%)

Metric	*CSE-CIC-IDS2018*			*NSL-KDD*		
	DNN	XGB	GBM	DNN	XGB	GBM
Acc	99.73	99.58	**99.74**	98.80	**99.66**	99.43
Prec	**99.80**	99.59	99.59	98.84	**99.66**	99.44
F1	**99.66**	99.58	99.58	98.80	**99.66**	99.43
Rec	**99.73**	99.58	99.58	**99.80**	99.66	99.43
AUC	99.96	**100**	**100**	99.84	**100**	99.92

samples after AWGAN augmentation; (3) The "Test" column shows the number of samples used for model evaluation.

The CSE-CIC-IDS2018 dataset has a total of 12 classes. The *Bengin* samples in this dataset are much larger than the attack samples. It has enough samples for *Bot* and $DDOS - HOIC$, while $SQL - Injection$ and $BruteForce - Web$ have very small attack samples.

Related to NSL-KDD, it has four classes. Like CSE-CIC-IDS2018, we also observe that *Benign* samples dominate the attack samples. However, *DoS* has larger samples, while the other attacks, i.e. $R2L$ and $U2R$, suffer from very low samples.

The individual models evaluated in this study achieved an F1 score of 99. 77% or higher except for the DNN model, which indicates excellent performance, shown as Table 2.4. We also can see that the AWGAN algorithm has greatly improved the quality of the training data set by confirming that all AUC measurements are greater than 99. 85%. This demonstrates the excellent efficiency gains in intrusion detection made possible by data augmentation using the AWGAN algorithm.

We also use the distributed stochastic neighbor embedding method (t-SNE) [102] in order to visualize high-dimensional training sets. Figure 2.8a and Figure 2.9a show the original data before performing AWGAN-based augmentation, while Figure 2.8b and Figure 2.9b illustrate the augmented training sets.

As illustrated in Figure 2.8 and Figure 2.9, the visualization confirms that the training set after using the AWGAN-based augmentation has solved the challenges of sparse and unbalanced data. The DS2 datasets had more distinct clusters corresponding to their classes than before the augmentation. In addition, with very high intrusion detection results, as illustrated in Table 2.4 for both datasets, AWGAN clearly qualifies to improve training set quality.

Table 2.4: Evaluation of AI models on WGAN-based Data Augmentation (%)

Metric	CSE-CIC-IDS2018					NSL-KDD				
	XGB	CBT	GBM	BME	DNN	XGB	CBT	GBM	BME	DNN
F1	99.77	99.92	**99.95**	99.77	97.75	**99.48**	99.21	**99.48**	**99.48**	98.00
Acc	99.76	99.92	99.96	**99.98**	97.54	99.49	99.22	**99.56**	99.43	98.07
Prec	99.83	99.93	99.96	**99.98**	98.20	**99.49**	99.21	**99.49**	99.41	98.03
Rec	99.76	99.92	99.96	**99.98**	97.54	**99.49**	99.22	**99.49**	99.43	98.07
FPR	**0**	**0**	0.03	**0**	0.13	0.67	1.27	**0.63**	0.77	1.22
FNR	**0**	0.01	**0**	**0**	1.37	0.37	0.39	**0.30**	0.32	2.26
AUC	**100**	**100**	99.99	99.99	98.69	**99.99**	99.98	**99.99**	99.89	99.85

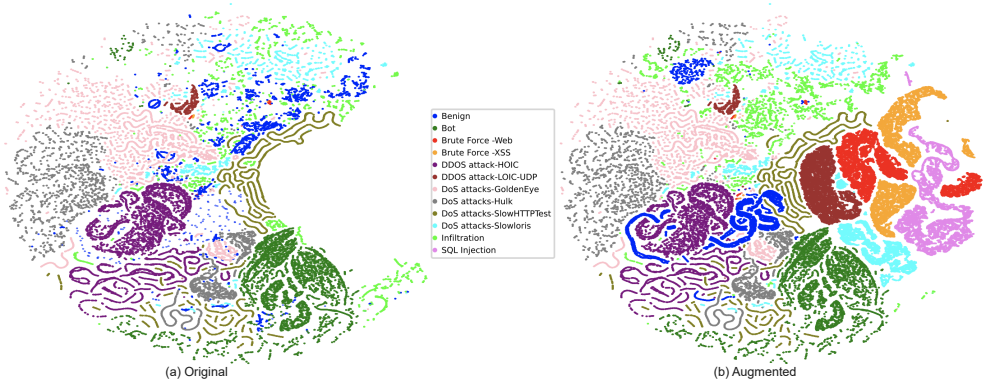


Figure 2.8: AWGAN-based Visualization of CSE-CIC-IDS2018 Training Set

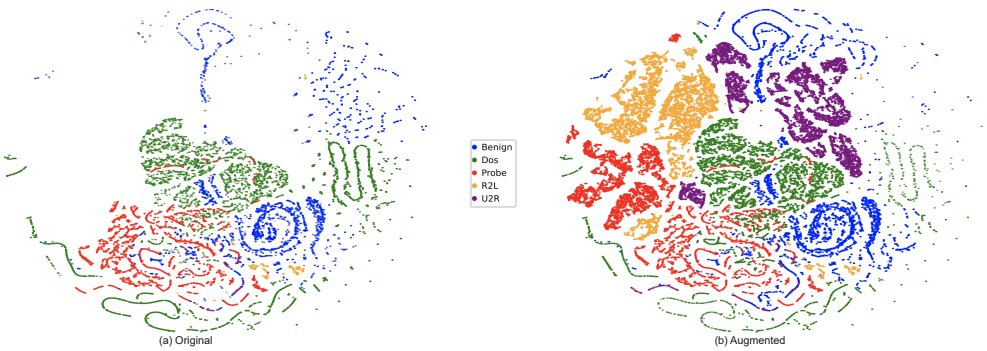


Figure 2.9: AWGAN-based Visualization of NSL-KDD Training Set

2.5.2.3 S3 Results

In this scenario, the models begin with no specified parameters, as the aforementioned algorithm automatically generates hyperparameters based on the dataset. Based on running all models on the EMBER2017 without tuning hyperparameters, the experiment result showed that with XGB and CBT, the accuracy is already above 99%, while GBM and CNN have an accuracy below 99%, shown as Table 2.5. This is not surprising, given that numerous researchers, including the original authors of the dataset, have commented on the relative simplicity of the EMBER dataset. Therefore, we focused only on fine-tuning the EMBER2018 and BODMAS dataset.

Furthermore, we performed the optimization feature method based on Algorithm 2.3 for EMBER2018 and BODMAS. Table 2.6 displays the results of the F1 score when using the 170 and 565 features of EMBER2018 dataset, indicating a superior performance when choosing the 565 features. The accuracy for the XGB model is 97.68%, the CBT model is 97.52%, the GBM model is 97.89% and the CNN model is 95.90%.

Table 2.5: Evaluation of AI models on Original Datasets(%)

Method	F1	Acc	Prec	Sens	FAR	FNR
EMBER2017 Evaluation						
XGB	99.16	99.16	99.16	99.16	0.84	0.84
CBT	**99.27**	**99.27**	**99.27**	**99.27**	**0.73**	**0.73**
GBM	98.67	98.67	98.67	98.67	1.33	1.33
CNN	95.95	96.04	93.72	95.95	3.35	4.05
EMBER2018 Evaluation						
XGB	97.63	97.63	97.63	97.63	2.37	2.37
CBT	97.19	97.19	97.19	97.19	2.81	2.81
GBM	**97.80**	**97.80**	**97.80**	**97.80**	**2.20**	**2.20**
CNN	94.03	94.02	94.16	94.02	5.97	5.98
BODMAS Evaluation						
XGB	98.71	98.69	99.68	97.75	0.32	2.25
CBT	**98.94**	**98.93**	99.88	**98.02**	0.12	**1.98**
GBM	98.90	98.89	**99.94**	97.88	**0.06**	2.12
CNN	98.90	98.89	99.87	97.96	0.13	2.04

To evaluate the effectiveness of our feature optimization and data balancing strategies,

Table 2.6: Evaluation of AI models based Features set Optimization (%)

Method	F1	Acc	Prec	Sens	FAR	FNR	F1	Acc	Prec	Sens	FAR	FNR
	BODMAS (4 features)						BODMAS (165 features)					
XGB	**89.76**	**90.78**	**85.19**	94.85	4.82	5.15	**99.28**	**99.39**	99.28	99.28	**0.61**	0.72
CBT	**89.76**	90.77	85.17	94.86	4.83	5.14	99.26	99.37	**99.29**	99.23	0.71	0.77
GBM	**89.76**	**90.78**	85.16	94.89	4.80	5.11	99.13	99.26	99.09	99.16	0.74	0.84
CNN	88.03	88.95	81.77	**95.34**	**4.66**	**4.66**	99.13	99.26	99.02	**99.24**	0.76	**0.74**
	EMBER2018 (170 features)						EMBER2018 (565 features)					
XGB	97.59	97.59	**97.84**	97.34	2.16	2.66	97.67	97.68	97.97	97.37	2.17	2.63
CBT	97.45	97.45	97.52	97.37	2.25	**2.53**	97.52	97.52	97.58	**97.46**	2.26	**2.54**
GBM	**97.85**	**97.86**	97.23	**97.47**	**2.13**	**2.53**	**97.88**	**97.89**	**98.34**	97.42	**2.16**	2.58
CNN	95.72	95.72	95.71	95.73	4.03	4.27	95.72	95.90	95.64	95.19	4.08	4.81

we compare the model performance in the original datasets shown in Table 2.5 and in the optimized feature sets shown in Table 2.6 for EMBER2018 and BODMAS dataset.

In EMBER2018 dataset, the original data and optimized datasets (170 and 565 features) show minimal differences in the results. For example, XGB has F1-scores of 97.63% (original), 97.59% (170 features), and 97.67% (565 features). All metrics, including FAR and FNR, remain stable across configurations, indicating that feature optimization preserves baseline performance on this well-structured dataset.

In the original BODMAS dataset, all models achieve F1-scores and accuracy below 98.9%, with particularly high precision for tree-based models (e.g., CBT: 99.88%). However, when models are trained on the optimized feature set with only 4 selected features, F1 and accuracy drop to around 89–91%. When using 165 optimized features, all models regain their high performance, with F1 and accuracy returning to 99.1–99.4%, and precision, recall (sensitivity), FAR, and FNR are all comparable to or slightly better than the original baseline.

This demonstrates that excessive feature reduction (e.g., to 4 features) can degrade performance, but a carefully selected subset (165 features) maintains or even improves the model's effectiveness compared to training on all original features. In particular, the XGB model achieves an F1-score of 99.28% and accuracy of 99.39% on the 165 feature dataset, slightly outperforming its original F1 of 98.71%.

For BODMAS dataset, optimizing and balancing features not only reduces model complexity, but also maintains or improves detection quality, particularly for the most important metrics (F1, Acc, Prec, Sens, FAR, FNR). For EMBER2018 dataset, the model is robust to both original and optimized feature sets, with performance differences less than 0.1

percentage points.

Feature optimization and data balancing, when carefully applied, achieves equal or superior results to the original baseline, while reducing input dimensionality and potential overfitting. This validates the practical utility and robustness of our approach across datasets and model types.

2.6 Summary

This chapter addresses one of the most significant challenges in intrusion detection: data imbalance, specifically, the redundancy of samples in the majority class and the scarcity of samples in the minority class. These issues not only lead to biased machine learning models that struggle to detect rare attacks but also increase the false positive rate. To overcome this, the chapter proposes, develops, and rigorously evaluates the dataset augmentation strategy and the optimization of the feature set.

The beginning of the chapter presents a detailed analysis of the practical challenges posed by network datasets such as CSE-CIC-IDS2018 and EMBER, where benign traffic dominates overwhelmingly, while attack samples, especially novel or sophisticated ones (e.g., infiltration, exfiltration, APT), are extremely limited. Relying solely on conventional or random oversampling methods often results in synthetic data that is noisy, less realistic, and can even degrade model performance.

To address these issues, the chapter introduces novel augmentation dataset techniques for machine learning. Specifically, the proposed method uses the Edited Nearest Neighbors (ENN) algorithm to partition the dataset into 'easy' and 'difficult' subsets. For the majority class within the difficult subset, a clustering-based compression technique using KMeans is applied to reduce redundancy while still preserving the most representative features of the model. For the minority class, in addition to a zooming technique that generates new samples around the original points, the method also uses WGAN to generate new attack samples. WGAN not only learns the true distribution of attack data, but also produces high-quality samples while assessing the quality of the generated data to enrich the minority class. Furthermore, we also propose the feature set optimization method, which reduces the dimensionality of features, thus increasing diversity while maintaining model interpretability.

The entire approach is empirically evaluated on both public and real-world datasets. The results show that the proposed method significantly improves the detection of rare attack classes, reduces false positives, and improves model stability. However, comprehensive comparisons will be made in the following chapters.

These research results have been partially presented in published works, including three

articles in respected journals (VVH-J2, VVH-J1, VVH-j3) and two conference paper (VVH-C2, VVH-C4), highlighting the novel and important contributions discussed in this chapter. Specifically, VVH-J2 presents an algorithm that addresses the challenge of class imbalance in network intrusion datasets through data compression and zooming techniques. VVH-J1 and VVH-C4 propose GAN-based methods capable of generating new samples to augment the minority class, thus mitigating data imbalance. VVH-j3 introduces a feature optimization approach to improve the quality of the dataset. In general, the content of these publications demonstrates the originality and scientific significance of the research contributions introduced in this chapter.

Chapter 3

Enhancing AI-powered Intrusion Detection with Mutual Deep and Boosting Inference

Building upon the enhanced data foundation established in Chapter 2, Chapter 3 focuses on harnessing the synergistic power of deep learning methods and modern boosting algorithms. This chapter changes the focus from the "quality" of input data to the "quality" of detection models, explaining how to effectively combine models like CNN, XGBoost, LightGBM, and CatBoost using methods like soft voting and stacking, which allows each model to work together well while being understandable and handling uncertainty effectively. Furthermore, mutual support and integration of these models increases the overall resilience of the proposed system; when one model fails to detect an attack, another may succeed and vice versa. The selection and integration of these approaches in this chapter also serve as an essential preparatory step toward building a practical, large-scale intrusion detection and defense system, an objective that will be further developed in the next chapter.

3.1 Problem Statement

In the evolving threat landscape, traditional machine learning techniques often fail to recognize sophisticated cyberattacks due to their limited representation capacity. As AI-based intrusion and malware detection systems are increasingly deployed in practice, one of the key challenges is optimizing the performance of machine learning models to achieve high accuracy, robustness, and generalization in real-world scenarios. Traditional machine learning methods, such as boost models or deep neural networks, each have their own advantages and limitations.

Boosting models such as XGBoost, LightGBM, and CatBoost excel in capturing non-linear features and often perform well in structured datasets [18]. However, they may lack the ability to detect subtle anomalies or abstract patterns, which are strengths of deep neural networks (CNNs, DNNs). In contrast, deep learning models, especially convolutional neural networks, have shown great promise in detecting sophisticated attack patterns, but are often susceptible to noisy data, are harder to interpret, and typically require substantial computational resources [1].

Another pressing issue is that most current approaches rely on single-model architectures

or only simple combinations of different models, resulting in unstable performance and limited resilience against adversarial attacks or complex real-world data [38]. Furthermore, selecting the optimal model for each dataset and scenario is a challenging task, requiring a delicate balance between accuracy, robustness, inference speed, and deployment feasibility.

These challenges highlight the need for a promising approach based on ensemble learning, which takes advantage of the complementary strengths of strengthening models and deep learning architectures. Integrating diverse models can reduce the risk of individual model bias or vulnerability while improving the detection of common, rare, or sophisticated attack patterns.

In this chapter, our aim is to answer the following problem statement: **How can we design a flexible ensemble learning framework that optimizes discrimination, enhances robustness, and maintains operational efficiency for intrusion and malware detection?**

To address this question, the proposed solution includes the following:

- Designing a mutual ensemble inference strategy through soft voting and stacking, allowing for complementary and mutually reinforcing detection capabilities across models.
- Explicitly improving system-level resilience by taking advantage of the compensatory nature of the ensemble. When one model fails to detect a threat, others in the ensemble may still succeed, thus increasing the overall robustness and reducing the risk of missed detections.
- Assess performance across metrics such as accuracy, F1 score, robustness to adversarial variation, system-level resilience to missed detections, and computational overhead.

This approach aims to harness the complementary strengths of both deep and boost models, leading to a unified detection mechanism that is accurate, robust to model weaknesses, and practical for deployment in real-world scenarios.

3.2 Network Intrusion Detection via AI-Powered Deep Analysis

3.2.1 Direction Approach

Suppose $P_1(f)$, $P_2(f)$,...$P_n(f)$ are the probability outputs of the n AI models for a feature vector f; ω_1, ω_2,..., ω_n are the weight ratio that represents the importance of model where

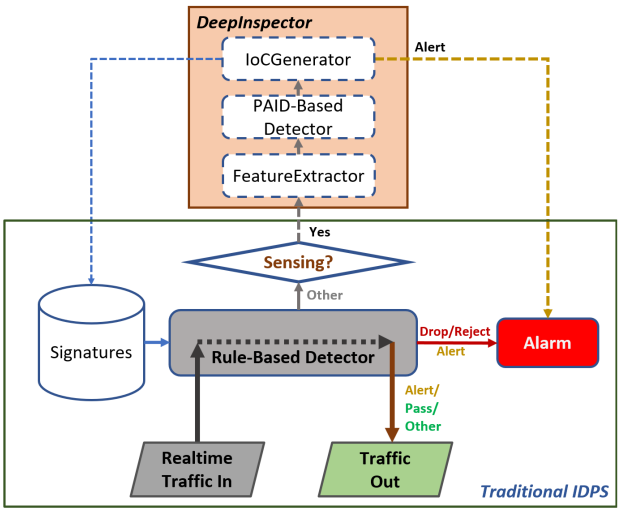


Figure 3.1: Network Intrusion Detection by Using AI-powered Deep Analysis

each $\omega_i \in (0,1)$ and $\sum_{i=1}^{n} \omega_i = 1$. The ensemble learning is involved to combine them into regular and attacks network flow by the following formula:

$$ELPred(f) = \sum_{i=1}^{n} P_i(f) * \omega_i \tag{3.1}$$

We developed the SDAID solution, a comprehensive network intrusion detection approach that uses deep AI-powered analysis to identify anomalous behavior, as illustrated in Figure 3.1. However, within the scope of this chapter, we focus solely on AI-powered deep analysis for network intrusion detection, specifically the DeepInspector component; other techniques will be presented in the following chapter. In DeepInspector, once it receives the flow data, it will perform AI-powered deep inspection by performing three core tasks as follows:

1. FeatureExtractor: this component takes on the role of extracting features from each network flow. CICFlowMeter can be used to perform this task by transforming a network flow into a vector of 83 features [44].

2. PAID-BasedDetector: this detector assumes AI- powered deep analysis. Its role is to analyze a feature vector as input and determine whether it is benign or an intrusion attack as output. Deep-sequence analysis powered by AI is described in more detail in Section 3.2.

3. IoCGenerator: this module takes the result of the PAID-BasedDetector as input, processes it, and generates an alert message (msg) for the Alarm component of IDPS. It also generates an indicator of compromise (IoC) from flow features such as Destination

Algorithm 3.1 PAID: Perform an Ensemble Learning for AI-powered Intrusion Detection

Model: XGB - XGB trained model; DNN - DNN trained model; GBM - GBM trained model

Input: f - traffic flow.

Output: (msg, IoC) - (alert message; generated new IoC)

$R \leftarrow \emptyset$

$F \leftarrow CICFlowMeter(f)$ ▷ extract 83 features of traffic flow f

$Fin \leftarrow F \backslash [FlowID, SrcIP, SrcPort, Label]$ ▷ remove 4 unused features

$Cats \leftarrow [DstPort, Protocol]$ ▷ Categorical Variables

$Conts \leftarrow Fin \setminus Cats$ ▷ 77 Continuous Variables

Perform three processes P1,P2,P3:

P1: $dnn\_preds \leftarrow DNN.predict(Cats, Conts)$ ▷ perform the prediction using DNN model

P2: $xgb\_preds \leftarrow XGB.predict(Cats, Conts)$ ▷ perform the prediction using XGB model

P3: $gbm\_preds \leftarrow GBM.predict(Cats, Conts)$ ▷ perform the prediction using GBM model

Wait P1, P2, P3 finished.

$avgs \leftarrow (xgb\_preds + dnn\_preds + gbm\_preds)/3)$

$FC \leftarrow avgs.argmax(axis = 1)$ ▷ get the flow labels from 0 to 11

if $FC! = 0$ **then** ▷ classified as network attacks

    $msg \leftarrow Alert(FC)$ ▷ constitute an alert by using metadata from the flow f; set alert category being as label

    $R \leftarrow IoCGenerator(FC)$ ▷ generate a new IoC to handle the next similar flows

end if

return $msg; IoC$

IP, Port, Protocol in the case of PAID-BasedDetector recognized as an attacked intrusion. This IoC is then sent to the Signatures component of traditional IDPS in order to update his IoC set. Note that when detecting a benign flow, the IoC and a null message are returned from the IoCGenerator.

The extraction of network traffic flows described above does not affect the processing of network traffic flows corresponding to the *Other* case. It indicates that the deep analysis process does not obstruct or discard network traffic. However, AI-powered deep analysis concentrates on anomaly detection and updating the IDPS ruleset with new signatures to prevent network attacks. In addition, the alert provides information for the administrator to have a plan for early network attack mitigation.

The DeepInspector component within our SAID method employs PAID, an AI-driven

deep analysis model. Based on the study in [63, 57], we acknowledge that DNN, GBM, and XGB are presently providing the most accurate network intrusion prediction results. Consequently, our in-depth analysis examines and employs these three methodologies. In addition, to take advantage of the strengths of both approaches, we propose combining DNN, GBM, and XGB in deep analysis for intrusion detection using ensemble learning.

In Algorithm 3.1, network traffic is first captured, extracted and modeled by 79 feature vectors. Next, these vectors are put into the XGB, GBM, and DNN models through three concurrent processes $P1$ and $P2$ $P3$. Once these processes are completed, the prediction results are combined with the function $argmax(x)$ to obtain a better result following the formula: $avgs = (xgb\_preds + dnn\_preds + gbm\_preds)/3$ and perform: $avgs.argmax(axis = 1)$ to get the final result. Lastly, depending on the outcome of the above step, if the network attack is detected, the IDPS will generate a new rule to drop/reject the network attacks and send an alert to the administrators.

3.2.2 Network Traffic Flow Modeling

As mentioned above, one of the core tasks in DeepInspector is to model a network traffic flow using a feature vector. There are also several ways to extract features from a network flow, such as CICFlowMeter, KDD99Extractor [64]. For CICFlowMeter, it allows to model a traffic flow in PCAP format into a vector of 83 features. Meanwhile, KDD99Extractor extracts 41 features for each flow.

In the SAID method, we propose using CICFlowMeter to perform the feature extraction task. From the output vector of 83 features, we recognize that "FlowID, SrcIP, SrcPort, and Label" are redundant features. We ultimately retained 79 features as input vector for the AI-powered analysis. In this vector, 'Dst Port, Protocol' are considered the categorical variables for classification. Thus, the 77 remaining features are considered continuous variables. DeepInspector will analyze and determine the intrusion attacks based AI models described in the following subsections.

3.2.3 DNN-based Intrusion Detection Algorithm

The network intrusion detection with the DNN model using the tabular learner technique of the FastAI development framework achieves the highest precision while minimizing the detection rate compared to other ML / DL models [57]. Therefore, we represent the DNN model using FastAI. However, in a DNN model, it is difficult to identify the optimal hyperparameters for efficiency and performance. In order to design the best DNN model, we use an Adaptive Experimentation Platform that employs Bayesian Optimization to turn the hyperparameters to obtain the optimal model.

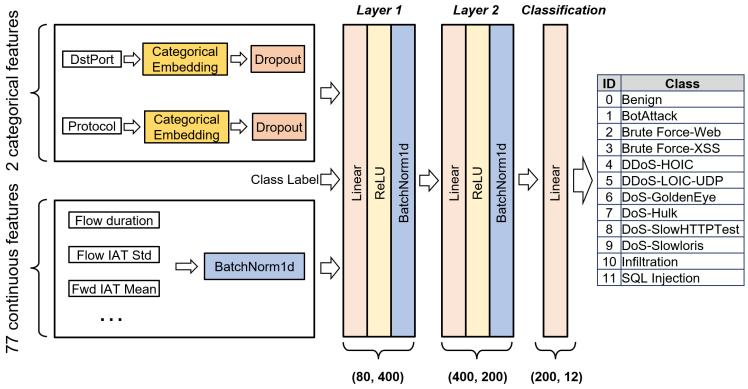


Figure 3.2: DNN-based Intrusion Detection.

Our DNN model is depicted in Figure 3.2. This architecture consists of four essential components: categorical variables, continuous variables, hidden layers, and output layers. Variables of type have discrete non-numeric values, such as IP address, protocol, etc. Continuous variables, on the contrary, encompass a range of values. Based on the feature vector generated from CICFlowMeter mentioned above, our DNN architecture is established as follows:

- The two features, DstPort and Protocol, are treated as categorical variables. Each feature will go through categorical embedding and dropout. The 77 features related to network traffic flows (specifically FlowDuration, TotFwdPkts, TotBwdPkts, TotLenFwdPkts, TotLenBwdPkts, FwdPktLenMax, FwdPktLenMin, FwdPktLenMean, FwdPktLenStd, BwdPktLenMax, BwdPktLenMin, BwdPktLenMean, BwdPktLenStd, FlowByts/s, FlowPkts/s, FlowIATMean, FlowIATStd, FlowIATMax, FlowIATMin, FwdIATTot, FwdIATMean, FwdIATStd, FwdIATMax, FwdIATMin, BwdIATTot, BwdIATMean, BwdIATStd, BwdIATMax, BwdIATMin, FwdPSHFlags, BwdPSHFlags, FwdURGFlags, BwdURGFlags, FwdHeaderLen, BwdHeaderLen, FwdPkts/s, BwdPkts/s, PktLenMin, PktLenMax, PktLenMean, PktLenStd, PktLenVar, FINFlagCnt, SYNFlagCnt, RSTFlagCnt, PSHFlagCnt, ACKFlagCnt, URGFlagCnt, CWEFlagCount, ECEFlagCnt, Down/UpRatio, PktSizeAvg, FwdSegSize- Avg, BwdSegSizeAvg, FwdByts/bAvg, FwdPkts/b- Avg, FwdBlkRateAvg, BwdByts/bAvg, BwdPkts/b-Avg, BwdBlkRateAvg, SubflowFwdPkts, SubflowFw- dByts, SubflowBwdPkts, SubflowBwdByts, InitFwdWinByts, InitBwdWinByts, FwdActDataPkts, FwdSegSizeMin, ActiveMean, ActiveStd, ActiveMax, ActiveMin, IdleMean, IdleStd, IdleMax, IdleMin) are used as continuous variable. These features are normalized by BatchNorm1D.

- Layer 1: The categorical class label of each network data flow is also combined with the input features into a vector of 80 features in the first hidden layer for training. It is composed of three standard blocks: “Linear”, “ReLU” and “BatchNorm1D”. Using

a hyperparameter optimization framework, the output of the first hidden layer is set to 400 features.

- Layer 2: For the second hidden layer, its structure is the same as that for the first hidden layer. However, from 400 input features, its output is normalized to 200 using the hyperparameter optimization framework.

- Classification: The final output layer assumes the role of classification through the linear filter. This set maps from 200 input features to 1 unique value representing a type of intrusion attack. In our work, except for 'Benign' flow, we are also interested in 11 intrusion types: *Bot, BruteForce-Web, BruteForce-XSS, DDOS-LOIC-UDP, DDoS-HOIC, DoS-Hulk, DoS-GoldenEye, DoS-SlowHTTPTest, DoS-Slowlor- is, Infiltration, SQLInjection.*

Note that Figure 3.2 illustrates our DNN architecture for AI models trained from datasets with feature sets like CICFlowMeter. Thus, it has to be modified in the DNN settings in the case of using another set of features.

3.2.4 Boosting-based Intrusion Detection Algorithm

The Boosting algorithms that provide superior classification performance in terms of accuracy and speed. In boosting learning, multiple models are constructed sequentially. The first model was constructed using arbitrary guesswork, whereas the second was based on residuals. The updated model was created by combining the two models. XGB is also an ensemble technique: It combines numerous decision trees to create a model composed of a decision tree forest. An individual decision tree is constructed by sorting all features and evaluating each conceivable split for each feature. The rule for a node is determined by the division that yields the highest score for the objective function [90]. Due to its performance and scalability, its prominence is increasing.

We use boosting models with several hyperparameters to detect network intrusions. First, we employ regularization to combat overfitting and avoid overly restrictive modeling of the training data. We also configure additional effective parameters, including Maximum Depth, Minimum Child Weight, and Gamma. The tree may split without correct regularization until it can precisely predict the training set, resulting in overfitting. The documentation for these hyperparameters can be found in Subsection 3.2.5. Moreover, we use our balanced datasets for training with boosting learning models. The experiments in Subsection 3.2.6 prove that the increase in learning models is among the best learning models to achieve an excellent precision detection rate compared to other DL models.

Table 3.1: Hyperparameter Optimization

Model	Hyperparameter	Value	Optimal
DNN	Learning rate	[0.001, 1.0]	0.003
	Batch size	[16, 32, 48, 64, 96, 128]	64
	Epochs	[1, 2, ..., 15, 16]	5
	Layers	[[200, 100], ..., [1000, 500]]	[400, 200]
XGB	Learning rate	[0,1]	0.01
	n_estimators	[1,∞]	30
	max_depth	[0,∞]	6
GBM	Learning rate	[0,1]	0.02
	min_samples_leaf	[1,∞]	30
	max_depth	[0,∞]	9

3.2.5 Hyperparameter Optimization

We select the model parameters based on a technique called Hyperparameter Optimization [78]. We use Ax for optimal parameters. Ax is a platform for optimizing any experiment, including ML experiments, A/B tests, and simulations. We use a technique called Bayesian Optimization. Bayesian optimization starts by building a smooth surrogate model of the results using Gaussian processes based on observations from previous rounds of experimentation [61].

In the DNN model, four hyperparameters can be tuned and take two types of values: range and choice, as shown in Table 3.1. From that, we obtained the learning rate being 0.003; batch size 64; the number of epochs 5; and $[400, 200]$ for the features of layers. The XGB and GBM are also configured for acting as a tree booster with six hyperparameters explicitly defined in the training phase. The optimal values are also illustrated in Table 3.1.

3.2.6 Experiments and Evaluation

For the experimental environment, we use the setup presented in Section 2.5. In this section, we present the results obtained from the deep experiments to prove the performance of our proposed method, PAID-based intrusion detection. We use DS1, prepared in Subsection 2.5.1, to experimentally evaluate the effectiveness of our approach. DS1 consists of two datasets: CSE-CIC-IDS2018 and NSL-KDD, both of which have been augmented using the method described in Subsection 2.3.1.

Table 3.2: Confusion Matrix of S1 Evaluation

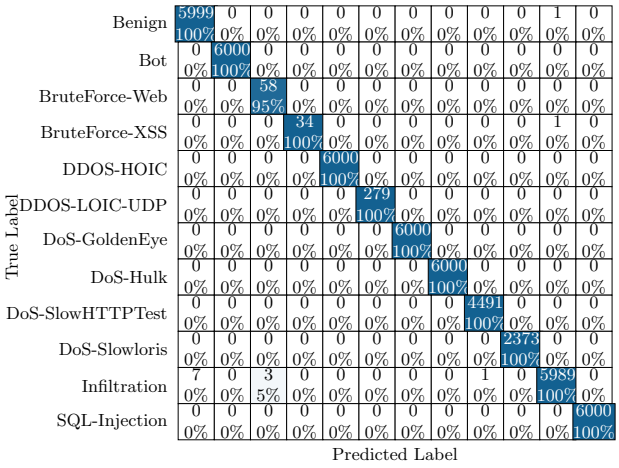
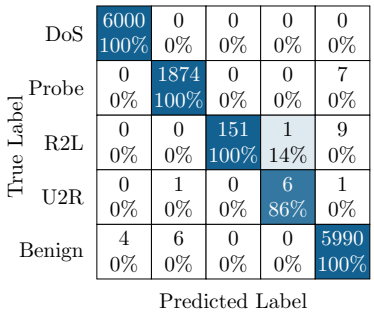


Table 3.3: Confusion Matrix of S2 Evaluation



We built the PAID-based DeepInspector tool from the proposed method to evaluate our method with the augmented datasets mentioned above. To evaluate PAID Algorithm 3.1, we perform the two scenarios described as follows:

- Scenario $S1$: Use the augmented CSE-CIC-IDS2018 dataset to train PAID. Four main performance metrics are used to evaluate the performance of PAID. Moreover, we also measure the performance of each model constituted to PAID.

- Scenario $S2$: In this scenario, we evaluate the PAID based on NSL-KDD, augmented from the NSL-KDD dataset, with the same method. However, NSL-KDD has only 41 features and four classes. Among those attributes, 'duration' is considered redundant and removed in this dataset. Two features 'protocol_type', 'service' are used as categorical variables and the 38 others features (flag, src_bytes, dst_bytes, land, wrong_- fragment, urgent, hot, num_failed_logins, log- ged_in, num_compromised, root_shell, su_attempted, num_root, num_file_creations, num_shells, num_acce- ss_files, num_out- bound_cmds, is_host _login, is_guest_login, count, srv_count, serror_rate, srv_serror_- rate, rerror_rate, srv_rerror_rate, same_srv_rate, diff_srv_rate, srv_diff_host_rate, dst_- host_count, dst_host_srv_count, dst_host _same_srv_rate, dst_host_diff_srv_rate, dst_-

Table 3.4: Performance Evaluation based Network Intrusion Detection

Metric	*S1 (CSE-CIC-IDS2018)*				*S2 (NSL-KDD)*			
	DNN	XGB	GBM	**PAID**	DNN	XGB	GBM	**PAID**
Acc	99.73	99.58	99.74	**99.97**	98.80	99.66	99.43	**99.69**
Prec	99.80	99.59	99.59	**99.97**	98.84	99.66	99.44	**99.69**
F1	99.66	99.58	99.58	**99.97**	98.80	99.66	99.43	**99.69**
Rec	99.73	99.58	99.58	**99.97**	99.80	99.66	99.43	**99.69**
AUC	99.96	**100**	**100**	**100**	99.84	**100**	99.92	99.99

host_same_src_port_rate, dst_host_srv_diff_host_rate, dst_host _serror_rate, dst_host_srv_serror_rate, d- st_host_rerror_rate, dst_host_srv_rerror_rate) considered as continuous variables of DNN model. Thus, the DNN settings in this scenario have to change the input of Layer 1 to 41 features and the output of the Classification layer to 4.

3.2.6.1 S1 Results

As mentioned above, the purpose of using the CSE-CIC-IDS2018 dataset is to objectively compare the effectiveness of PAID with methods in other studies that also use the same CSE-CIC-IDS2018 dataset. The confusion matrix illustrates the results of our experiment performed with the PAID method, shown in Table 3.2. Moreover, we also implement the DNN, GBM, and XGB methods and compute all their performance metrics, such as accuracy, precision, and F1-score, etc. Consequently, we present the evaluation results in the first part of Table 3.4.

In this scenario, 3/6000 $Infiltration$ attacks are denoted as $BruteForce-Web$; 1/6000 $Infiltration$ attacks are defined as $DoS-SlowHTTPTest$; 1/35 $Brute-Force-XSS$ attacks are defined as $Infiltration$ and 1 $Benign$ flows are considered intrusion. The false negative rate is low: only 7 $Infiltration$ attacks (in total $43,236$ attacks) are not detected by PAID. Meanwhile, the false-positive rate is remarkable: only one benign flow is considered an intrusion. In general, the accuracy of the DNN, XGB, GBM and PAID models is 99.73%, 99.58%, 99.46% and 99.97%, respectively. These results confirm that our proposed PAID method is the best.

3.2.6.2 S2 Results

Based on NSL-KDD, we use the augmented training set to train the DNN, XGB, GBM, and PAID models and perform the prediction on the testing set. Note that in this case, the PAID algorithm is set to four class labels corresponding to NSL-KDD. We consequently

Table 3.5: Comparison of PAID with other SOTA methods

Method	Acc	Prec	F1	Rec
CSE-CIC-IDS2018-based Evaluation				
PAID (our)	**99.97**	**99.97**	**99.97**	**99.97**
WGAN+IDR [22]	–	99	98	97
RANet [113]	96.73	–	96.59	96.73
Adaboost [55]	99.69	99.70	99.70	99.69
Autoencoder [21]	99.20	95.00	-	98.90
AUE [114]	97.90	98.00	98.00	98.00
DSSTE + miniVGGNet [68]	96.99	97.46	97.04	96.97
LSTM + AM + SMOTE [67]	96.20	96.00	93.00	96.00
NSL-KDD-based Evaluation				
PAID (our)	**99.69**	**99.69**	**99.69**	**99.69**
Autoencoder [3]	99.20	-	-	99.27
Multiple LSTM [52]	98.94	-	-	99.23
SMO [49]	96.20	-	-	-
RANet [113]	83.23	–	82.57	83.23
DNN [105]	78.50	81.00	76.50	78.50

indicate this experiment results for the PAID as the confusion matrix shown in Table 3.3. In more detail, we have the results of S2 with 1/161 $R2L$ attacks labeled as $U2R$ and 1/8 $U2R$ attacks labeled as the *Probe*. The total attacks on the 17 network (including 7 *Probe*, 9 $R2L$ and 1 $U2R$) are not detected by PAID, and 10 *Benign* flows are considered intrusion. For the performance metrics for the three trained models, the accuracy for the DNN model is 99.36%, for the GBM model it is 99.45% and for the XGB model it is 99.53%. Finally, the *PAID* gets 99.69% of accuracy. All metrics allowing for evaluation results are shown in the second part of Table 3.4. It also indicates that PAID has the greatest performance compared to the other models.

3.2.7 Comparison with SOTAs

In favor of well-known datasets, such as NSL-KDD and CSE-CIC-IDS2018, our experiment results can be compared with other SOTA methods. The comparison of intrusion detection performance between PAID and SOTA is summarized in Table 3.5. It is clear that the F1 score and the accuracy of our PAID method reach the same 99.69% with the NSL-KDD dataset, higher than the accuracy of all compared models, such as Adaboost Autoencoder + Softmax, which is 99.20% accuracy, or Multiple LSTM, which is 98.94% accuracy. With

the CSE-CIC-IDS2018 dataset, the accuracy of PAID is 99.97%; it is also higher than the accuracy of other compared models; for example, the Adaboost has 99.69% of accuracy, the DSSTE-niniVGGNet has an accuracy of 96.99%, or AUE has an accuracy of 97.90%.

Our experiments also indicate that PAID-based prediction has the same complexity and analysis speed as DNN-based prediction with respect to deep analysis speed. This benefit resulted from the concept of concurrent ensemble learning and performance. Consequently, the accuracy, precision, and speed results demonstrate that PAID is currently the most efficient learning model.

Based on experimental findings, the XGB and GBM models have the shortest execution time compared to the DNN and PAID models. Moreover, the accuracy of the PAID ensemble learning model is not significantly greater than that of the XGB model. Administrators can select the ensemble learning PAID, XGB, or GBM model to investigate in-depth traffic flows based on the network size, scope, and security level.

3.3 Malware Detection via Mutual Deep and Boosting Ensemble Learning

3.3.1 Approach Direction

We represent executable files as binary feature vectors to construct a classifier for malware detection. For this purpose, datasets often provide comprehensive features extracted from real-world executables. With these datasets and features 1...n, we can construct a vector X for each input sample such that $X \in \{0,1\}^n$. $X_i = 1$ indicates the presence of feature X and $X_i = 0$ indicates its absence.

Let's assume that the probability outputs of n AI models for a feature vector f are $P_1(f)$, $P_2(f)$,..., and $P_n(f)$. In our method, we perform soft voting ensemble learning for these above models following the formula: $P_{voting} = \frac{1}{n}\sum_{i=1}^{n} P_i(f)$. Finally, we use soft voting (P_{voting}) as the base model and individual models (P_i) to perform stacked ensemble learning.

We apply ensemble learning, including soft voting and stacking, to build binary classification models for malware detection. This approach also improves resistance against evasive attacks that attempt to alter, obfuscate, or compromise the system. The method we propose in this study is called MDOB, an acronym for *"Enhancing Resilient and Explainable AI-Powered **M**alware **D**etection using Feature **O**ptimization and **M**utual **D**eep+**B**oosting Ensemble Learning."* Figure 3.3 illustrates the comprehensive architecture of our MDOB method.

In MDOB, the first stage presented in Subsection 2.4.4, is feature optimization, with the aim of reducing the number of features and identifying the most significant features to improve the performance of AI models. Then, we combine ensemble voting and stacking

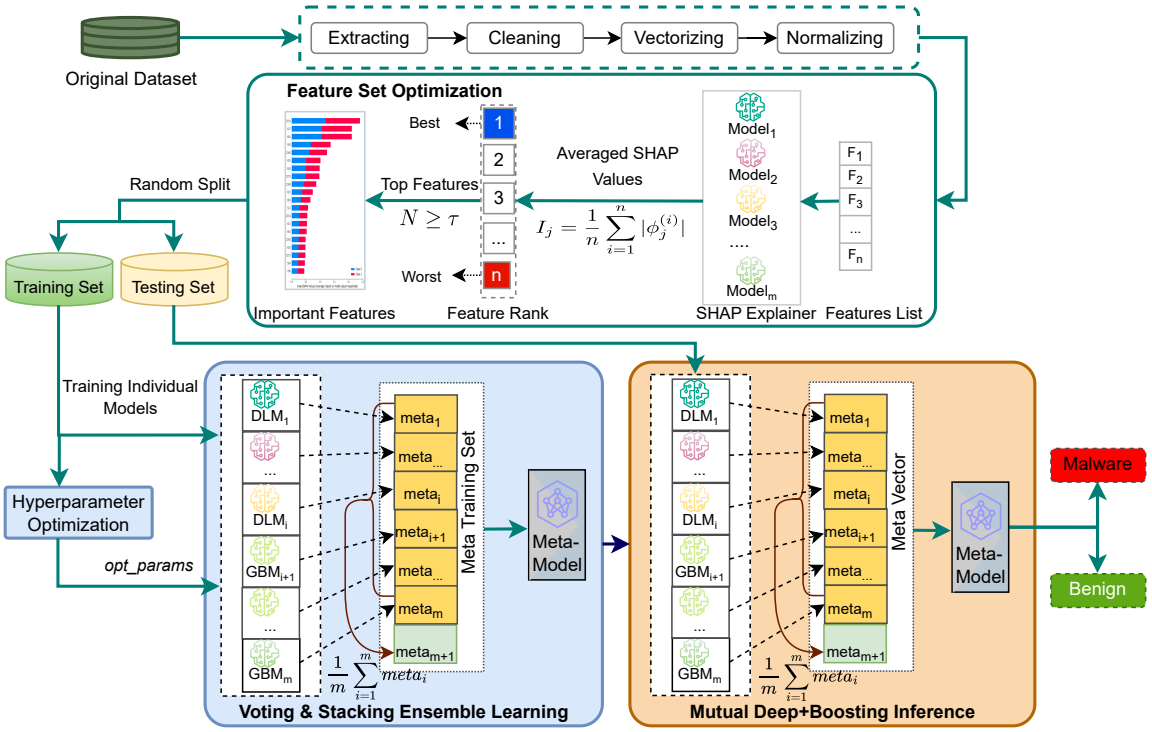


Figure 3.3: Architecture of MDOB-based Malware Detection

learning methods to enhance the accuracy of the ML model. Thus, MDOB aims to detect malware through the following main steps:

Step 1: We begin by utilizing the techniques described in Subsection 2.4.1 to extract, clean and vectorize the dataset. For example, the data are scaled using Equation 2.8. This process yields a cleaned and normalized dataset ready for the next steps.

Step 2: We optimize the feature set by applying Algorithm 2.3 to analyze and choose the most significant features, thus reducing the data dimension for the training and detection phase. By removing irrelevant features, the output improves the training and testing sets, enabling more efficient training and testing in the subsequent steps.

Step 3: Using the optimized feature from Step 2, we build the AI models using three main ideas:

1. Combining multiple AI models, including deep models such as DNN, CNN, etc., and gradient-boosting models to enhance resistance against alter, obfuscate, and improve the accuracy of malware detection.
2. Developing an AI model that utilizes both ensemble voting and ensemble stacking to enhance malware detection performance.
3. Performing inference-based malware detection to improve reasoning speed utilizing multiple complementary AI models through a combination of DL and GBM.

Although MDOB is a comprehensive approach, within the scope of Chapter 3, we focus mainly on methods to improve the performance of machine learning models, which is *Step*

3. The Subsection 3.3.2 will detail our mutual deep + boosting ensemble learning approach.

3.3.2 Mutual Deep and Boosting Learning

As analyzed and evaluated in Section 1.3, malware detection based on executable file behavior analysis with AI models can currently be categorized into two main approaches: using DL models or GBM models. However, relying solely on a single model or focusing only on combining multiple models of the same type does not effectively detect different types of malware. Therefore, in this study, we propose a mutual learning model that integrates both the DL and the GBM models. The selection of DL and GBM models for our AI framework will be determined using AutoML frameworks such as AutoGluon. AutoGluon easily supports many AI models, including deep learning models like CNN and DNN, as well as gradient boosting methods such as XGB, CBT, and GBM, all of which fit well with the MDOB teamwork approach.

Moreover, AutoGluon offers automated hyperparameter tuning, model selection, and performance optimization, which are essential for efficiently handling high-dimensional datasets like EMBER. However, through empirical evaluation of the DL and GBM models, we have observed that DL models generally do not achieve as high performance as GBM models (as demonstrated in the experimental results in Subsection 3.2.6). Consequently, our mutual learning approach will leverage multiple GBMs while selecting only the most optimal DL model to maximize the strengths of each AI model type.

In particular, DL models are effective in identifying complex patterns, and GBM algorithms are excellent at making firm decisions. Running them simultaneously speeds up analysis and improves detection accuracy, leading to a stronger system that can quickly adapt to new threats. By combining these methods, we can build a strong defense system that adapts to cyber threats, enabling us to make real-time updates and improvements. This framework strengthens our cybersecurity measures and helps organizations react proactively to possible vulnerabilities.

Deep learning-based detection methods have recently gained popularity due to their ability to automatically extract features from large datasets. These techniques, which employ neural networks to increase the accuracy and efficacy of identifying patterns in complex data, are essential for several applications, such as malware or intrusion detection. We designed our CNN model for binary classification tasks, shown in Figure 3.4. The CNN model architecture is defined as follows:

- Input Layer: The input shape is $(\text{columns}, 1)$, where *columns* represents the number of features.
- Convolutional Layer: Consists of 128 filters with a kernel size of 3 and ReLU activation.
- MaxPooling Layer: Downsamples the convolutional layer output with a pool size of 2.
- Flatten Layer: Flattens the previous layer's output to a 1D tensor.

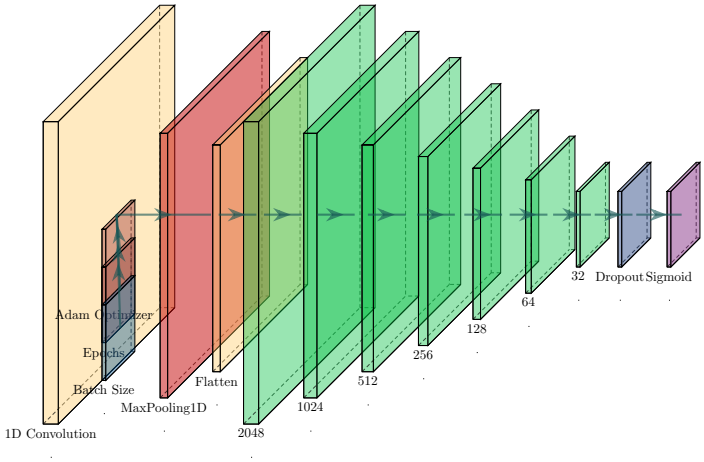


Figure 3.4: Architecture of CNN Model

- Fully Connected Layers: This consists of several dense layers with decreasing units: 2048, 1024, 512, 256, 128, 64, and 32, all using ReLU activation.
- Dropout Layer: Apply dropout regularization with a dropout rate of 0.5 to prevent overfitting.
- Output Layer: Consists of a single neuron with sigmoid activation, outputting the probability of the positive class.

The Adam optimizer compiles the model with a learning rate and a binary cross-entropy loss function. Accuracy is monitored during training. During training, we select the best model based on validation accuracy. The CNN model aims to classify input data into one of two classes based on the features provided.

This step aims to enhance the accuracy and effectiveness of detection systems by using machine learning techniques. It consistently improves the performance of detection tasks in a wide range of settings by combining several ML models into a single strong model.

3.3.3 Combination of Voting and Stacking Ensemble Learning

Our approach integrates voting and stacking learning to construct a more robust model using multiple AI-based classifiers. The method first applies soft voting to combine probability predictions from different AI models and then uses stacking to train a new model. This two-layered ensemble strategy improves predictive performance and generalization by taking advantage of the complementary strengths of the individual base learners.

In particular, the algorithm takes as input a training dataset TD with an optimized feature set OFS. It first trains a set of AI models $MS = \{M_1, M_2, \ldots, M_m\}$, then uses the soft voting method to aggregate the predictions and generate a new meta-training set MTD, which is subsequently used to train a meta-model MM. During the training phase

Algorithm 3.2 VSEL: Combination of Voting and Stacking Ensemble Learning

Input: $TD = \{(X^i, y^i)\}_{i=1}^{N}$ - training dataset with optimized features; $MS = \{M_1, M_2, ..., M_m\}$ - set of m base models; $model\_params$ - optimized hyperparameters of m AI models; K - number of folds for building meta training dataset (MTD).

$MTD \leftarrow \emptyset$ ▷ Init MTD
$\{TD_1, TD_2, ..., TD_K\} \leftarrow Split(TD, K)$ ▷ Split the training dataset into K folds
for each fold $k \in 1..K$ **do**
    $TD_{\text{train}} \leftarrow TD \setminus TD_k$; $TD_{\text{val}} \leftarrow TD_k$ ▷ Use $K-1$ folds for training and 1 fold for validation
    for each $M_i \in MS$ **do**
        $M_i \leftarrow \text{Train}(M_i, TD_{\text{train}}, model\_params[M_i])$
    end for
    for each $(X, y) \in TD_{\text{val}}$ **do**
        $meta \leftarrow \emptyset$; $vote\_sum \leftarrow 0$ ▷ Create the meta-feature vector
        for each $M_i \in MS$ **do**
            $p_i \leftarrow M_i(X)$ ▷ Predict the probability for X using the trained base model M_i
            meta.push(p_i)
            $vote\_sum \leftarrow vote\_sum + p_i$
        end for
        $p_{\text{vote}} \leftarrow vote\_sum/m$ ▷ Calculate soft voting prediction from all base models
        meta.push(p_{vote}) ▷ Add soft voting result as an additional feature m+1 in the meta-layer
        $MTD.push(meta, y)$ ▷ Add the meta-feature vector and corresponding label to MTD
    end for
end for
$MM \leftarrow \text{AutoML.SelectBestModel}(MTD)$ ▷ Perform AutoML on MTD to select the best as the meta model
for each $M_i \in MS$ **do**
    $M_i \leftarrow \text{Train}(M_i, TD, model\_params[M_i])$ ▷ Retrain all base models on the whole training dataset to be used in final prediction
end for

Output: MS - n trained AI models; MM - trained meta model.

for the base model, we employ techniques such as cross-validation to improve robustness and mitigate overfitting. Each model in the set MS is trained with its own optimized hyperparameters. This process is illustrated in Algorithm 3.2.

Furthermore, adversary attacks often employ obfuscation and deformation techniques

to generate new types that can evade malware detection methods. However, our method combines multiple ML models and ensemble learning techniques to detect malware; this allows us to test an input PE file that embeds malware in these models. If this model fails to detect malware, other models may be able to detect it. This work creates the ability to detect and prevent adversary attacks [24].

3.3.4 Hyperparameter Optimization

To optimize ML models in our approach, such as training individual models, we use Optuna [6] to find the best parameters for each model, ensuring that their performance is maximized. This work is done through Algorithm 3.3.

Algorithm 3.3 Hyperparameter Optimization using Optuna

Input: $model$ - AI model; $D_{train} = (X_{train}, y_{train})$ - training set; $D_{test} = (X_{test}, y_{test})$ - testing set; N_{trials} - number of trials; $T_{timeout}$ - optimization timeout; $params$ - list of hyperparameters.

function OBJECTIVE($trial$)
    $model\_params \leftarrow \{p_1, p_2, p_3, \ldots, p_n\}$ ▷ Initialize dictionary of hyperparameters for the model
    for $p \in params$ **do** ▷ Use Optuna to suggest hyperparameter values for each parameter p
        $model\_params[p] \leftarrow$ trial.suggest_⟨parameter_type⟩(“p”, ⟨min_value⟩, ⟨max_value⟩)
    end for
    $clf \leftarrow$ model(**model_params) ▷ Instantiate model with current parameters
    $clf.fit(X_{\text{train}}, y_{\text{train}})$ ▷ Train model on training data
    $preds \leftarrow clf.predict(X_{\text{test}})$ ▷ Make predictions on testing data
    $metric \leftarrow$ performance_metric($y_{\text{test}}, preds$) ▷ Compute evaluation metric
    return metric
end function
Initialize an empty dictionary $opt\_params = \emptyset$
Optimize the objective function using Optuna:
$study \leftarrow optuna.create\_study(direction = \text{“}maximize\text{”})$
study.optimize($objective$, n_trials=N_{trials}, timeout=T_{timeout})
$trial \leftarrow$ study.best_trial
$opt\_params \leftarrow trial.params$ ▷ Get optimized model parameters from the best trial
return opt_params

Output: opt_params - optimized hyperparameters.

Our method initializes an empty list to store the optimized model parameters (opt_params). Subsequently, it defines the objective function $objective(trial)$, which assesses the perfor-

mance of the model with a specified set of hyperparameters. We define the parameters of the ML model (*params*), which represent the hyperparameters for optimization. The algorithm suggests hyperparameters for each parameter in *params*; then the model is instantiated with the suggested hyperparameters and trained on the training data ($X_{\text{train}}, y_{\text{train}}$). Predictions are made on the testing data (X_{test}), and a performance metric is calculated based on actual labels (y_{test}) and predicted labels (*preds*). The algorithm returns the performance metric as an objective value. We create a study object (*study*) and set the optimization direction to"maximize". The objective function (*objective*) is optimized by invoking the *study.optimize* function with the specified number of trials (N_{trials}) and the timeout (T_{timeout}), resulting in the best trial of the study. We then retrieve the optimized model parameters (*opt_params*) from the best trial.

Finally, the algorithm returns the optimized model parameters as an output. In summary, the algorithm quickly finds the best hyperparameters for a ML model by suggesting that it use Optuna repeatedly and check how well it works on a validation set. Afterward, it provides the set of hyperparameters that yield the best performance metric.

3.3.5 Experiments and Evaluation

For the experimental environment, we use the setup presented in Section 2.5. We use DS3, prepared in Subsection 2.5.1, to experimentally evaluate the effectiveness of our approach. DS3 consists of three datasets: EMBER2017, EMBER2018 and BODMAS datasets, which have been augmented using the method described in Section 2.4. Based on the MDOB method described in Section 3.3, we successfully built the AI-powered malware detection tool. We conducted two scenarios to evaluate MDOB, as detailed below.

- Scenario $S1$: The focus is on using the EMBER2018 dataset to evaluate our proposed MDOB method. We consider this dataset to be more challenging than its previous iteration. We use this scenario to assess the effectiveness of our models and compare them with other recent methods.
- Scenario $S2$: We evaluated our proposed MDOB method using the BODMAS dataset. We consider the BODMAS dataset as an additional benchmark that complements the EMBER datasets by offering malware samples from a more recent time. BODMAS dataset also brings in different types of malware and gives us samples with timestamps, which helps us analyze changes over time and check how well our models handle new malware threats. This scenario allows us to examine the generalization capability of our models when applied to newer datasets beyond the EMBER dataset series.

Based on the hyperparameters tuned above, we get a prediction from all models simultaneously. After that, we calculate the prediction of the averaging probability from the child model, and we use the *argmax* function to identify the label with the highest probability, thus achieving optimal final results.

3.3.5.1 S1 Results

Similarly to EMBER2017, we ran the model without hyperparameters and tuned it around the initial parameters for EMBER2018. Unlike EMBER2017, the initial accuracy could not be better this time. The increased complexity of EMBER2018 compared to its previous version partly explains this. We focused on fine-tuning for all models.

For example, Figure 3.5 shows the fine-tuning of the CNN model, using 512 epochs, a batch size of 64, and a validation split of 10%. The model quickly fits the training set, with the loss approaching zero and the accuracy nearly reaching 100%. However, the validation loss fluctuates heavily and increases with more epochs, while the validation accuracy stabilizes around 97–97.5% without further improvement. This indicates a clear overfitting, where the model memorizes the training data but does not generalize to unseen samples. This observation explains why CNN in Figure 3.6 underperforms compared to boosting models, which are more stable on tabular features. The results highlight that CNN is not optimal when applied directly to EMBER2018 and requires additional strategies such as regularization, early stopping, or integration of the ensemble to improve generalization. The accuracy for the XGB model is 97.68%, the CBT model is 97.52%, the GBM model is 97.89%, the CNN model is 95.90%, and the voting of these models is 97.89%. Finally, MDOB obtains the accuracy of 98.14%.

Figure 3.6 compares the F1-score of different models on the EMBER2018 dataset using 565 features. The results indicate that boosting-based models achieve high and stable performance, ranging from 97.5% to 97.9%, with LightGBM slightly outperforming the others. In contrast, CNN yields the lowest result (95.7%), reflecting the limitations of deep learning architectures when applied directly to tabular features extracted from PE files. In particular, the ensemble methods further enhance performance: Soft voting reaches about 98.0%, while stacking achieves the highest score at 98.1%. These findings confirm that the integration of multiple models can take advantage of the complementary strengths of individual algorithms, mitigate their weaknesses, and ultimately deliver superior accuracy and robustness for malware detection.

The results also indicate that CNN is less effective in malware detection than other approaches, which exhibit the miss rate FNR at 4.81%. In contrast, MDOB achieves the lowest FNR at 2.32%, highlighting the effectiveness of ensemble models in improving detection accuracy, making it the most reliable method to minimize undetected malware. Meanwhile, GBM, CBT, and XGB have moderate FNRs ranging from 2.5% to 2.6%, indicating that they are slightly less effective than MDOB in reducing false negatives.

3.3.5.2 S2 Results

We ran the models on BODMAS without extensive hyperparameter tuning, similar to what we did with EMBER2018. Compared to EMBER datasets, BODMAS has newer types of malware and different time patterns, which shows that malware detection is changing.

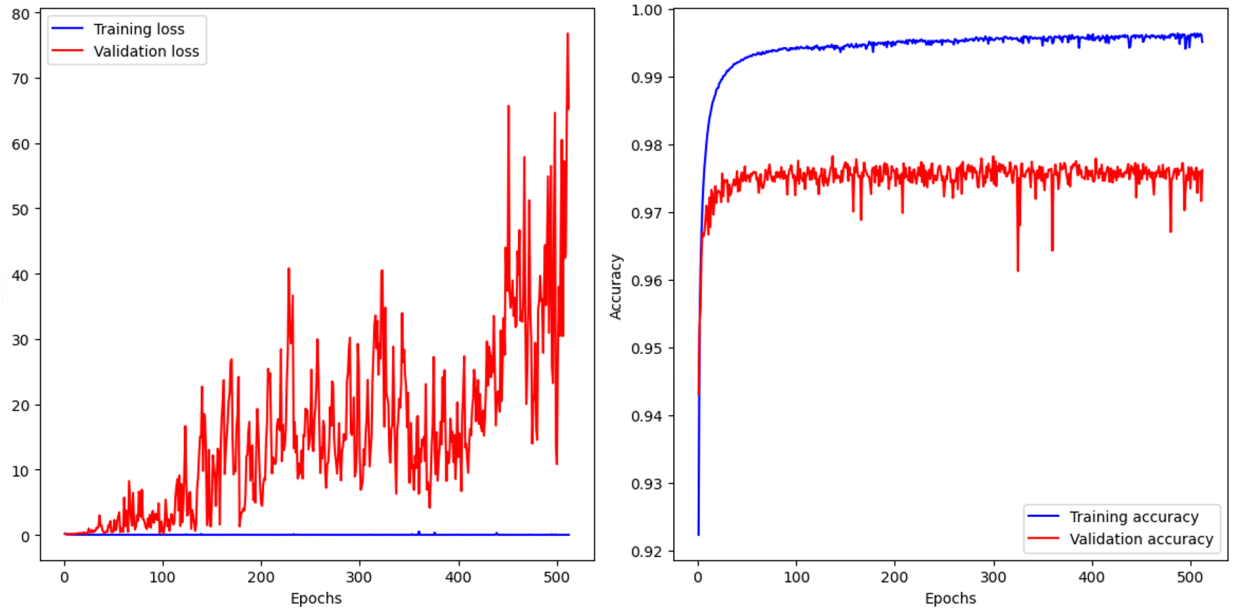


Figure 3.5: CNN Training Performance based EMBER2018 (565 features)

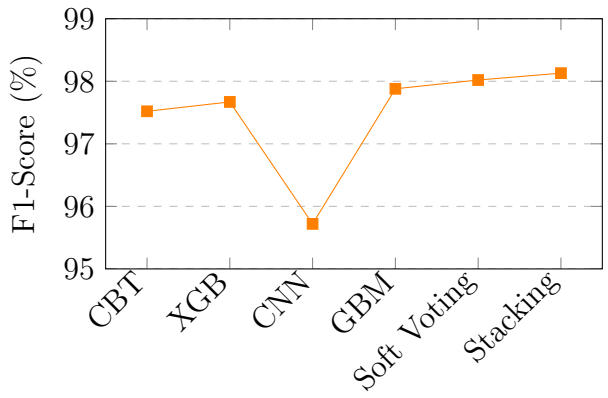


Figure 3.6: EMBER2018-based Performance on 565 Features

However, it does not have standard feature definitions and only includes feature vectors without harmless files because of copyright problems, making it hard to fully reproduce the results. For this experiment, we used the feature sets selected by SHAP at threshold 0.01, specifically the 4-feature (intersection) and 165-feature (union) subsets.

The results, summarized in Table 3.6, show a noticeable gap in the model performance between the minimal feature set (4 features) and the enriched feature set (165 features). Specifically, with 165 features, the XGB model achieves an accuracy of 99.39%, the CBT model reaches 99.37%, the GBM model achieves 99.26%, and the CNN model obtains 99.26%. The soft voting ensemble further improves performance to 99.42%, while MDOB achieves the highest accuracy at 99.46%.

Figure 3.7 presents the F1-score performance of XGBoost, CatBoost, LightGBM, CNN, and two ensemble methods on the BODMAS dataset using 165 features. The results reveal that all models achieve consistently high performance, ranging narrowly from 99.1%

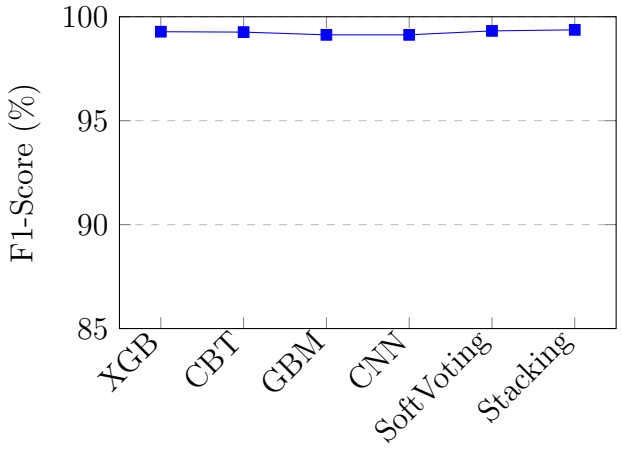


Figure 3.7: BODMAS-based Performance on 165 Features

Table 3.6: Evaluation of AI models based Malware Detection (%)

Learning	Method	F1	Acc	Prec	Sens	FAR	FNR	F1	Acc	Prec	Sens	FAR	FNR
		BODMAS (165 features)						EMBER 2018 (565 features)					
Baseline	XGB	99.28	99.39	99.28	99.28	0.61	0.72	97.67	97.68	97.97	97.37	2.17	2.63
	CBT	99.26	99.37	99.29	99.23	0.71	0.77	97.52	97.52	97.58	97.46	2.26	2.54
	GBM	99.13	99.26	99.09	99.16	0.74	0.84	97.88	97.89	98.34	97.42	2.16	2.58
	CNN	99.13	99.26	99.02	99.24	0.76	0.74	95.72	95.90	95.64	95.19	4.08	4.81
Mutual DLM+GBM	Voting	99.32	99.42	99.34	99.30	0.66	**0.70**	98.02	97.89	98.38	97.65	2.03	2.35
Mutual Voting+Stacking	**MDOB**	**99.37**	**99.46**	**99.48**	**99.26**	**0.54**	0.74	**98.13**	**98.14**	**98.58**	**97.68**	**1.93**	**2.32**

to 99.3%. Among the base models, XGBoost, CatBoost, and CNN achieve nearly identical scores, with only marginal differences at the decimal level. LightGBM records a slightly lower score, though the gap is negligible. Ensemble methods continue to provide incremental benefits: soft voting outperforms individual models, while stacking achieves the highest result, approaching 99.4%. These findings suggest that when the feature set is already well-refined (165 features), the performance differences among individual models diminish. However, ensemble approaches remain valuable as they exploit the complementary strengths of different models, yielding an optimal and more stable detection performance.

F1 scores exhibit a similar trend, and MDOB achieves the best F1 scores of 99.37% (165 features). MDOB greatly reduced the false negative rate (FNR) to 0.74% when using 165 features, showing that the ensemble methods are very effective in improving detection abilities.

3.3.6 Comparison with SOTAs

The comparison of malware detection implementations between MDOB and SOTA is summarized in Table 3.7. The accuracy of our MDOB method reaches 98.14%, higher than the accuracy of all the compared methods, such as dualFFNN k-medoids [94], which is 98.02%

Table 3.7: Comparison of MDOB with SOTA Methods (%)

Method	Venue	Acc	Prec	F1	Sens
	EMBER2018				
MDOB (our)	-	**98.14**	**98.58**	**98.13**	**97.68**
AutoML [20]	Computers & Security 2024	95.80	–	95.80	–
dualFFNN k-medoids [94]	Computers & Security 2023	98.02	–	–	–
Consensus [80]	CMC 2023	96.77	–	96.77	–
DL [10]	Telecom 2023	95.57	–	–	–
MLMD [86]	CAI 2023	97.42	–	–	–
DNN [59]	IJNIS 2022	94.09	90.14	88.66	88.85
	BODMAS				
MDOB (our)	-	**99.46**	**99.48**	**99.37**	**99.26**
EII-MBS [48]	Computers & Security 2022	99.29	98.26	94.23	98.07
MD-ADA [17]	Computers & Security 2024	99.29	–	99.13	–
FCG-MFD [45]	JNCA 2025	99.28	–	99.14	–

of accuracy, or AutoML [20], 95.80% of accuracy; it is also higher than the F1-score of other compared models; for example, DNN [59] has 88.66% accuracy, and the consensus [80] has an accuracy of 96.77%. On the BODMAS dataset, our MDOB method achieves an accuracy of 99.46%, which is higher than all other compared methods using the same dataset, such as MD-ADA [17] with 99.29%, EII-MBS [48] and FCG-MFD [45] both with 99.28%. Moreover, MDOB maintains the lowest false alarm rate (0.54%) and achieves the highest F1-score (99.37%) among all models evaluated. Consequently, the accuracy, precision, and speed results demonstrate that MDOB is currently the most efficient learning model.

3.4 Summary

In this chapter, we focus on improving the performance and robustness of intrusion and malware detection systems through ensemble learning and mutual interaction among machine learning models. Building on the enhanced datasets developed in Chapter 2, this chapter addresses the limitations of individual models and proposes a unified framework that takes advantage of the complementary strengths of both deep learning and modern boosting algorithms.

The chapter begins by analyzing the challenges faced by conventional classifiers such as convolutional neural networks (CNNs) and standalone boosting algorithms (e.g., XGBoost, LightGBM, CatBoost). Although deep learning models are highly effective at extracting

complex features from raw or structured data, they often suffer from instability and lack of interpretability, especially under imbalanced or adversarial data conditions. In contrast, boosting models are generally more stable and interpretable, but are limited in their ability to recognize highly complex and nonlinear attack patterns.

To overcome these limitations, we introduce a mutual ensemble inference framework that combines deep learning and boosting via two strategies: soft voting and stacking meta-learning. In this framework, deep and boost models are trained on the augmented and balanced datasets from Chapter 2. The soft voting approach aggregates predictions from all base models to enhance stability and consistency, while stacking employs a meta-classifier to learn from the outputs of the base models, thereby improving accuracy and adaptability.

We conduct comprehensive experimental evaluations on multiple benchmark datasets for both network intrusion detection and static malware detection. The results demonstrate that the hybrid ensemble approach significantly outperforms both individual model baselines and the most advanced current methods, achieving superior accuracy, adaptability to rare classes, generalization to unseen attacks, and robustness to noise or adversarial data. The experiments also show that this method increases stability and enhances system resilience; when one model fails, others may succeed; this is a key strength of the proposed approach.

These research results have been partially presented in published works, including two articles in respected journals (VVH-J1, VVH-j3) and two conference papers (VVH-C1, VVH-C3), highlighting the novel and significant contributions discussed in this chapter. Specifically, VVH-J1 and VVH-C1 introduce and evaluate the mutual ensemble inference framework that combines deep and boosting learning to enhance the effectiveness of network intrusion detection. Meanwhile, VVH-j3 and VVH-C3 provide a detailed presentation of the integration of these ensemble methods, as well as performance analysis in malware detection tasks. In general, the content of these publications demonstrates the originality and scientific significance of the research, laying the foundation for the deployment of large-scale network systems discussed in the next chapter.

Chapter 4

Holistic Large-Scale AI-powered Intrusion Prevention with Flow Sensing Strategy and Parallel Ensemble Inference

After proposing methods to balance the dataset and develop detection models in previous chapters, Chapter 4 shifts its focus to implementing these achievements in practice by designing and deploying an effective intrusion detection and prevention system for large-scale network environments. At this stage, the emphasis moves from "models and algorithms" to "system design and real-world application," where critical strategies such as flow sensing, high-speed processing methods, rapid response capabilities and sandbox-based malware analysis play a decisive role in transforming research results into practical value. This chapter evaluates the overall effectiveness of the entire process, while also providing a comprehensive perspective on scalability, quick adaptability, and efficient deployment in today's real-world scenarios.

4.1 Problem Statement

Although previous chapters have focused on improving the quality of data and models, real-world deployment of AI-based IDS systems introduces a new dimension of challenges. In operational environments, timeliness and scalability are critical constraints. Detection models must not only be accurate, but also be able to make decisions in real time and under resource constraints [25].

In the context of increasingly complex network environments, the demand for proactive defense against cyber threats has driven the development of AI-based intrusion detection and prevention systems (IDS/IPS). However, deploying such systems in large-scale real-world networks presents unique challenges in terms of performance, latency, scalability, and resilience against sophisticated attack techniques.

Traditional intrusion detection approaches including signature-based, rule-based, and even standalone deep learning models have inherent limitations. Signature-based methods typically only detect known threats and struggle to adapt to novel or previously unseen attacks. In contrast, modern AI models, especially deep learning, show promise for anomaly and zero-day attack detection but require significant computational resources, making real-time processing difficult in large-bandwidth networks [17]. Furthermore, many current systems assume a static data pipeline and lack the ability to adapt in real time, leading to

poor performance when confronted with dynamic traffic patterns or evolving stealth attack vectors.

Another challenge is the gap between the performance of the offline model and the real-world defensive capability after deployment. Evasive, adversarial attacks or previously unseen anomalous traffic can cause many AI models to become unstable or even ineffective compared to their performance in controlled laboratory environments.

In particular, proactive prevention differs fundamentally from detection alone by requiring extremely low-latency decision-making, often in user space to maintain both accuracy and scalability. This requires system architectures that are lightweight, highly parallelized, and capable of context-sensitive flow sensing, so that computational resources are prioritized for high-risk flows while avoiding unnecessary overhead in regular traffic.

Although several approaches have attempted to address these issues, most current solutions suffer from trade-offs: sacrificing accuracy for speed, focusing solely on detection while overlooking proactive defense, or failing to optimize for large-scale, real-world deployments. Very few solutions simultaneously achieve the goals of accuracy, real-time responsiveness, scalability, resilience, and adaptability.

From this practical perspective, the central research problem can be stated as follows: **How can we design an AI-powered intrusion prevention system that is operationally viable, ensuring high-throughput real-time processing, robust detection accuracy, scalability, and resilience to sophisticated attacks in large-scale network environments?**

To address this question, this dissertation focuses on proposing a proactive AI-based defense architecture that integrates multiple key components:

- Integration of flow sensing strategies to dynamically determine inference needs.
- Real-time system evaluation under emulated large-scale traffic with emphasis on latency, accuracy, and deployment feasibility.
- The design of NetIPS: a user-space intrusion prevention architecture with parallel inference across core models.

The goal is to build a holistic defense system ready for deployment in large-scale networks, providing high performance, reliability, and strong adaptability. To validate the effectiveness of the proposed solution, this research conducts extensive experiments on large-scale datasets and realistic network simulation environments, benchmarking latency, accuracy, scalability, and overall defensive efficacy against current state-of-the-art methods.

4.2 Proposed Holistic Intrusion Detection Framework

4.2.1 Approach Direction

In reality, deploying an IDPS system on a network in inline mode presents several obstacles. Thus, inspecting all traffic in detail to detect anomalies is impossible, particularly for ML/DL methods and large-scale network traffic with a throughput of 10Gbps or even 100Gbps. Our comprehensive intrusion detection approach uses deep AI-powered analysis to identify anomalous behavior and signatures of previous intrusions, namely APELID, as illustrated in Figure 4.1 and Algorithm 4.1. To ensure high-throughput network traffic, this method combines three core inspectors based on the *shallow*, *AI-powered deep analysis* and *Sandbox*. The *shallow analysis* uses a ruleset based on known intrusion signatures to inspect all traffic flows on the network. However, the *AI-powered deep analysis* focuses only on flows that do not match any rule in the current IDPS ruleset. Meawhile *Sandbox* focuses to detect malware file that transmit between networks. To increase resilience when conducting deep analysis in large-scale networks, our approach is to control and trigger the AI-Powered deep analysis regularly using a flow-sensing mechanism that samples the traffic flows depending on two factors: sampling cycle and duration.

The traffic assessment is conducted as described below. First, network traffic is captured in both the receiving and transmitting directions, and then decoding is performed. Next, we apply rule-based detection to network traffic to determine known attacks. Each rule has a unique pattern or signature that identifies malicious network traffic. Network traffic will be analyzed and granted four actions based on the protocol: (i) *drop* the packet; (ii) *reject* the packet (discard the packet and notify the source that sent the packet); (iii) *alert* and allow to pass the packet; and (iv) *pass* without warning. The remaining case, denoted by *'Other'*, corresponds to network traffic flows that do not match rules. Our primary objective in deep inspection is to identify anomalous network traffic behavior using AI-powered analysis.

The **Signature-Based Detector** and the flow-sensing strategy are deployed in a traditional IDPS, such as Suricata, Snort, or Zeek. Traditional IDPS has to be fine-tuned in order to be able to capture "Other" flows regularly by using the global setting variable *"Sensing"* to perform AI-powered analysis in **DeepAnalyzer**.

For large-scale networks, we propose a flow-sensing mechanism that periodically samples network traffic to prevent analysis bottlenecks. Our concept of periodic sampling is described as follows: for each cycle T, capture all flows corresponding to the 'Other' instance in a δ interval. Those flows are typically represented by the 'PCAP' format and will be sent to DeepAnalyzer within a interprocess communication (IPC) mechanism by using a Unix socket. Our flow-sensing mechanism is described in more detail in Subsection 4.2.3.

The extraction of network traffic flows described above does not affect the processing of network traffic flows corresponding to the *Other* case. It indicates that the deep analysis process does not obstruct or discard network traffic. However, AI-powered deep analysis

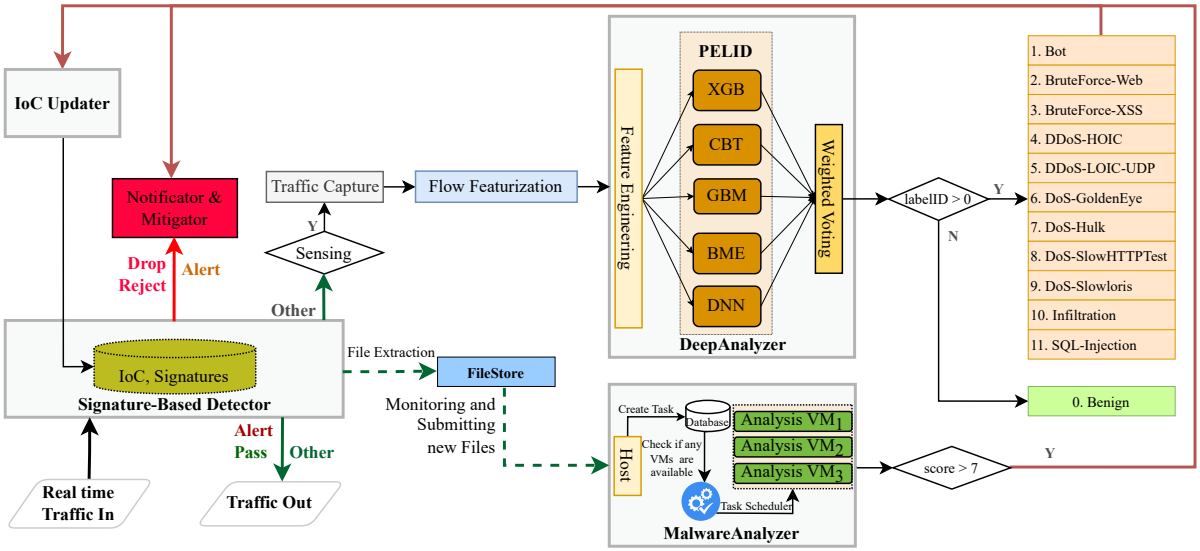


Figure 4.1: Architecture of Holistic Intrusion Detection

concentrates on anomaly detection and updating the IDPS ruleset with new signatures to prevent network attacks. In addition, the alert provides information for the administrator to have a plan for early network attack mitigation.

Moreover, to improve the ability to detect malicious files transferred over the network, our proposed APELID solution is integrated with a **MalwareAnalyzer** based on a sandbox approach as illustrated in Figure 4.1. The MalwareAnalyzer is assumed to perform both static and dynamic file analysis to identify malware threats. Our method for detecting malicious files transmitted by the network is as follows. IDPS will initially acquire network-transmitted files and store them in the *FileStore* folder. Then, we construct a Python program that periodically examines the folder *FileStore* for new files. Therefore, all new files are automatically submitted to MalwareAnalyzer for sandbox-based malware analysis.

4.2.2 Parallel Ensemble Inference-based Intrusion Detection

Two ideas motivated our intrusion detection method: the ensemble learning approach and parallel computing. The first proposal tries to improve the quality of intrusion detection, while the second helps to reduce the latency of intrusion detection. As a result, the intrusion detection approach suggested in our study is known as **PELID**, which stands for "Parallel Ensemble Learning for Intrusion Detection."

As described in [40, 72, 56, 110], the most effective new AI models for intrusion detection are DNN, XGB, CBT, GBM, and BME. In addition, as illustrated in Table 1.2, the accuracy and F1-score results for intrusion detection of these models currently exceeded 98%. Thus, it predominantly affects our selection of these models for our PELID ensemble.

In PELID, the combination of numerous AI models is performed by soft-voting method. However, the effect of each individual model is regulated by a weighted score ($\omega_i \in (0, 1)$)

Algorithm 4.1 Holistic intrusion Detection by flow sensing strategy and deep analysis

Input: f - Traffic In Flow, S - Signature Set, $Sensing$ - perform AI-powered deep analysis or not, F - Files that transfer between network.

Output: f, msg, S - (Traffic Out Flow; Alert Message; Updated Signature Set)

$action \leftarrow RuleBasedDetector(f, S)$
$IoC_{\text{set}} \leftarrow \emptyset$
if $action = Drop/Reject$ **then** ▷ Drop/Reject flow due of a detected critical attack
    Drop/Reject(f)
    $msg \leftarrow' CriticalAttack'$
    return $(none, msg, S)$
else if $action = Alert$ **then** ▷ Generate an alert
    $msg \leftarrow' Alert\_based\_on\_Signature'$
else if $action = Pass$ **then** ▷ Stop further inspection of the flow
    $msg \leftarrow None$
else if $Sensing = True$ **then** ▷ f does not match any rules, then AI-powered deep analysis is triggered by the sensing mechanism
    $(msg_{deep}, IoC_{deep}) \leftarrow DeepAnalyzer(f)$ ▷ Inspect F deeply by PELID and return a message and new IoC if an intrusion attack is detected.
    $IoC_{set} \leftarrow IoC_{set} \cup IoC_{deep}$ ▷ Update IoC_{set} with new indication of compromise IoC_{deep}
end if
for each $t \in F$ **do**
    $(msg_t, IoC_t) \leftarrow MalwareAnalyzer(t)$ ▷ Analysis t deeply by Sandbox return a message and new IoC_t if an malware file is detected.
    $IoC_{set} \leftarrow IoC_{set} \cup IoC_t$ ▷ Update IoC_{set} with new indication of compromise IoC_t
end for
$S \leftarrow S \cup IoC_{set}$ ▷ Update S with new indication of compromise IoC_{set}
return (f, msg, S)

on the overall PELID model. In general, with n AI models, the total sum of these scores must be 1: $\sum_{i=1}^{n} \omega_i = 1$. All of these scores will be determined experimentally in order to identify the optimal combination of a variety of AI models. Consequently, Algorithm 4.2 shows our PELID algorithm.

In Algorithm 4.2, the network traffic flow is first captured, extracted, and modeled by a feature vector F. In this step, the CICFlowMeter tool [97] can be used and return a vector of 83 features for each flow. Next, F will be cleaned by removing unused features and normalizing the rest. With CICFlowMeter explicitly, this step retains only 73 features by eliminating [*FlowID, SrcIP, SrcPort, Label, BwdPSH- Flags, BwdURGFlags, FwdByts/bAvg, FwdPkts/bAvg, FwdBlkRateAvg, BwdByts/bAvg*]. Among them, two features [$DstPort, Protocol$] are used as categorical variables and the rest 71 are considered

Algorithm 4.2 PELID: Parallel Ensemble Learning-based Intrusion Detection

Model: XGB, GBM, CBT, BME, DNN - XGB, GBM, CBT, BME and DNN trained model, and their ensemble weight ω_i where $\sum_{i=1}^{5} \omega_i = 1$.

Input: f - traffic flow.

Output: (msg, R) - (alert messages; new generated rules)

$R \leftarrow \emptyset$

$F \leftarrow Featurize(f)$ ▷ Extract features of traffic flow f.

$Fin \leftarrow Normalize(F)$ ▷ Perform the feature engineering: remove unused features and normalize the rest.

$Cats \leftarrow [DstPort, Protocol]$ ▷ Categorical variables

$Conts \leftarrow Fin \setminus Cats$ ▷ Continuous variables

Perform in parallel five processes P1, P2, P3, P4, P5:

P1: $pXGB \leftarrow XGB.predict(Cats, Conts)$ ▷ Perform the prediction using XGB.

P2: $pGBM \leftarrow GBM.predict(Cats, Conts)$ ▷ Perform the prediction using GBM.

P3: $pCBT \leftarrow CBT.predict(Cats, Conts)$ ▷ Perform the prediction using CBT.

P4: $pBME \leftarrow BME.predict(Cats, Conts)$ ▷ Perform the prediction using BME.

P5: $pDNN \leftarrow DNN.predict(Cats, Conts)$ ▷ Perform the prediction using DNN.

Wait P1, P2, P3, P4, P5 finished.

$scores \leftarrow (pXGB * \omega_1 + pGBM * \omega_2 + pCBT * \omega_3 + pBME * \omega_4 + pDNN * \omega_5)$

$FC \leftarrow scores.argmax(axis = 1)$ ▷ Get the flow predicted label.

if $FC! = 0$ **then** ▷ Classified as network attacks

    $msg \leftarrow Alert(FC, f)$ ▷ Generate an alert by using metadata from the flow f; set alert category being as predicted label.

    $R \leftarrow RuleGenerator(FC, f)$ ▷ Generate a new signature based on its indicator of compromise.

end if

return $msg; R$

continuous variables. It should be noted that all of the above steps are also used to prepare the training set before training each AI model. Moreover, all AI models have to be trained by using AWGAN augmented datasets before using PELID, which is present in Section 2.4.

Returning to the PELID algorithm, the normalized vectors are then fed into AI models to run the prediction step. Here, in order to enhance the speed of intrusion detection, AI-based predictions are performed in parallel. Concretely, in PELID, five $P1$, $P2$, $P3$, $P4$, and $P5$ processes are run simultaneously to compute the probability of intrusion. In the event that network activities under an intrusion attack are identified, PELID will send an alert message to the administrator and generate rules in the form of an indicator of compromise (IoC) to update the IDPS's signature database.

4.2.3 Strategy for AI-powered real-time intrusion detection

When the AI-powered model for detecting intrusions is implemented, a large-scale traffic network, such as an optical one, experiences significant network latency or bottleneck congestion. We employ a rule-based engine to detect and prevent common network attacks. The APELID will then concentrate on network traffic flows missed by the current rules-based engine to decrease analysis time. It operates in two phases, as described in the following sections:

Initially, a well-known IDPS, such as Suricata or Snort, is used to capture network traffic in both the receiving and transmitting directions. Next, we use a rules-based engine to analyze network traffic in order to detect and prevent known network attacks: *drop*, *reject*, *alert*, and *pass*. APELID performing an in-depth analysis will identify abnormal network traffic behavior for the remaining case, denoted by 'Other.'

Second, during the in-depth analysis phase, the IDPS is modified to capture flows corresponding to 'Other.' This flow data will be then analyzed by the PELID method to identify one of the twelve labels: *Benign, DoS-Slow HTTPTest, BruteForce-Web, BruteForce-XSS, DDOS-LOIC -UDP, DDoS-HOIC, DoS -Hulk, DoS-GoldenEye, Bot, DoS -Slowloris, Infiltration, SQL-Injection*.

For large-scale network traffic, the deep analysis certainly causes the stuck of IDPS. Therefore, we propose an efficient strategy to sense the traffic flows. Thus, we control the periodic deep analysis sampling strategy using 6 variables: DI_Cycle, DIC_Min, DIC_Max, and DI_Window, DIW_Min, DIW_Max. These parameters are all natural numbers with units of seconds, and their meanings are as follows:

- DI_Cycle: is the sampling cycle T for deep analysis. Suppose that this parameter has a value equal to 0, and the IDPS system will include a deep traffic analysis with a random cycle in the range of DIC_Min, DIC_Max. These default parameters are 60, 30, and 300 seconds, respectively.
- DI_Window: is the window size for deep analysis. If this parameter is 0, the system will capture the flows for deep analysis in a random window size from DIW_Min to DIW_Max. The default value of DI_Window is 10 seconds, and DIW_Min, DIW_Max is 1 and 30 seconds, respectively.

In IDPS, these parameters are selected and configured. The sampling cycle and duration will determine the performance of IDPS for high-volume network traffic (throughput of 10Gbps or more). If DI_Cycle is tiny or big DI_Window, DeepAnalyzer must make more predictions. It leads to increased latency and possibly causes bottlenecks. Consequently, these parameters are also chosen based on the context of network throughput and IDPS's computing capacity.

In light of these findings, Figure 4.1 illustrates the system architecture necessary to integrate our APELID into an inline IDPS. Note that if the DeepAnalyzer detects anomalous network behavior, the IDPS will generate an indicator of compromise (IoC) and add it

Algorithm 4.3 Malware Detection

Input: F - New files transferred in network and accumulated in $FileStore$ folder.

Output: (msg, R) - (Alert Message, New Rules generated based malware detected files).

1: $Ready \leftarrow Wait\_Sandbox\_Ready$ ▷ Blocking-function until Sandbox is ready.
2: $IngestFiles(F)$ ▷ Send F in the $FileStore$ folder to Sandbox
3: $score = HybridAnalyzer(F)$ ▷ Determine the overall score of both static and dynamic analysis.
4: **if** $score > 7$ **then** ▷ Critical suspicious file
5: $R \leftarrow RuleGenerator(F)$ ▷ Update the rule to block connection.
6: $msg \leftarrow$ '$Detected\_Malware\_Files$'
7: **return** msg, R
8: **end if**

to the signature-based ruleset of the IDPS. Consequently, it notifies the administrator of current traffic flows and thwarts future network attacks of a similar nature.

4.2.4 Hunting Malware by Sandbox Approach

In order to improve the capability to detect malicious files transferred over the network, our proposed APELID solution is integrated with a *MalwareAnalyzer* based on a sandbox approach, as illustrated in Figure 4.1. Algorithm 4.3 illustrates our strategy to analyze and identify this malware file. This will further enrich the IoC rule set based on domains, hosts, and IP-Ports for a more proactive approach to detecting and preventing malware through the IDPS system. For example, in cases where the IDPS cannot detect a malware file, the *MalwareAnalyzer* can identify it and provide IoC indicators to complement the IDPS's IoC ruleset, thereby enabling detection and prevention in similar instances in the future. Thus, it is assumed to perform both static and dynamic file analysis to identify malware threats. Our method for detecting malicious files transmitted by the network is as follows. IDPS will initially acquire network-transmitted files and store them in the *FileStore* folder. Then, we construct a Python program that periodically examines the folder *FileStore* for new files. Therefore, all new files are automatically submitted to MalwareAnalyzer for sandbox-based malware analysis.

MalwareAnalyzer enables the deployment of a well-known sandbox, such as Cuckoo, with two essential entities: the Host and the Agent. Each agent can be launched on a virtual machine that has been quarantined (Analysis VM). The analysis VM will execute the file and record its complete behavior. For further investigation, it can also identify malware-associated behaviors, such as extracted artifacts, registry modifications, *dropped* files, related processes, DLL library files in use, and network activity data.

MalwareAnalyzer also considers statistical analysis utilizing the Yara utility. Here, we

investigate the file signatures, hashes, strings, and other data related to the suspicious file. MalwareAnalyzer integrates with several additional malware analyzers, including Virus Total.

Lastly, MalwareAnalyzer combines the static and dynamic analysis results and gives the analysis report with a severity score of 0 to 10. If the severity score exceeds 7 in our proposal, MalwareAnalyzer will send an alert to the administrator and collect file-related information, such as the incoming IP address, domain, URL, etc. These data permit the development of a new IoC-based rule and its incorporation into the IDPS signatures database, thus preventing similar future threats. Consider a scenario in which the severity score is less than 7, such as anti-executable malicious code on analysis virtual machines. In this case, the results of the simulator analysis will also be sent to the administrator to provide additional information. In some cases, human reverse engineering analysis is required to assess the actual malware risk.

In particular, MalwareAnalyzer is not designed to prevent malware files in real time. However, our method enables us to proactively enhance and improve the IDPS IoC and signature database in response to future comparable threats.

4.3 Experiments and Evaluation

To demonstrate the performance of our method, we conducted a comprehensive experiment to answer the following research questions:

1. RQ1: Does combining multiple AI models of PELID, both traditional ML and DL, allow enhancing the performance of network intrusion detection and reducing analysis time?
2. RQ2: When deploying an IDPS inline system in an intranet with large-scale network traffic, is it fast enough to conduct a deep analysis of network flows for intrusion detection with the AI model generated by the APELID method to ensure that network flows are handled in real time?
3. RQ3: Is it possible to implement malware file detection in the inline IDPS system combined with deep analysis based on the AI model?

Our rigorous experiments were conducted to answer the above research questions. The following sections, in turn, detail the results we obtained while experimenting and evaluating our APELID method. We use DS2, prepared in Subsection 2.5.1, to experimentally evaluate the effectiveness of our approach. DS2 consists of two datasets: CSE-CIC-IDS2018 and NSL-KDD, both of which have been augmented using the method described in Subsection 2.3.2.

Table 4.1: Confusion Matrix of CSE-CIC-IDS2018-based PELID

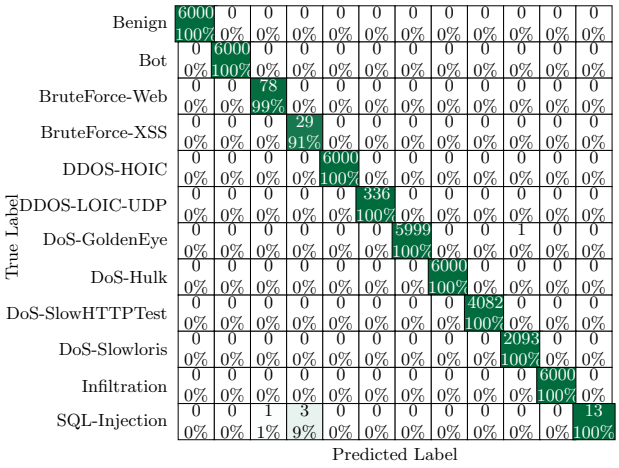
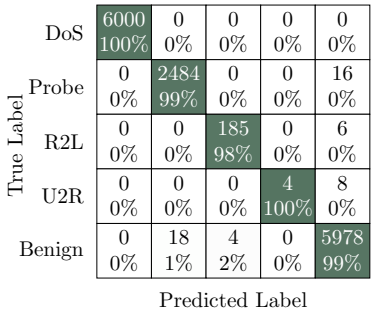


Table 4.2: Confusion Matrix of NSL-KDD-based PELID



4.3.1 Experimental Results

For the experimental environment, we use the setup presented in Section 2.5. We implemented the AGWAN and PELID algorithm and deployed them in an inline IDPS to validate the four research questions mentioned above. Therefore, three scenarios are proposed to evaluate our methods: CSE-CIC-IDS2018-based experiments, NSL-KDD-based experiments, and a practical model for hunting malware in an IDPS using the sandbox method.

The experiment process for both two datasets is the same. First, the augmented training set is used to train five individual AI models of PELID. Then, the testing set is used to assess not just the performance of the five AI models, but also the ensemble model PELID. In addition, the experiment identifies all the evaluation metrics and measures the time required to analyze each traffic flow using PELID-based intrusion detection. To avoid the impact of other processes on the testing server, the time consumption of PELID is the average value obtained from six separate prediction runs. The results obtained will be summarized and evaluated in the next subsection.

Table 4.3: Evaluation of AI models based PELID (%)

Metric	CSE-CIC-IDS2018						NSL-KDD					
	XGB	CBT	GBM	BME	DNN	**PELID**	XGB	CBT	GBM	BME	DNN	**PELID**
F1	99.77	99.92	99.95	99.77	97.75	**99.99**	99.48	99.21	99.48	99.48	98.00	**99.63**
Acc	99.76	99.92	99.96	99.98	97.54	**99.99**	99.49	99.22	99.56	99.43	98.07	**99.65**
Prec	99.83	99.93	99.96	99.98	98.20	**99.99**	99.49	99.21	99.49	99.41	98.03	**99.65**
Rec	99.76	99.92	99.96	99.98	97.54	**99.99**	99.49	99.22	99.49	99.43	98.07	**99.65**
FPR	**0**	**0**	0.03	**0**	0.13	**0**	0.67	1.27	0.63	0.77	1.22	**0.37**
FNR	**0**	0.01	**0**	**0**	1.37	**0**	0.37	0.39	0.30	0.32	2.26	**0.34**
AUC	**100**	**100**	99.99	99.99	98.69	**100**	**99.99**	99.98	**99.99**	99.89	99.85	**99.99**

4.3.1.1 CSE-CIC-IDS2018-based Results

These experiments focus on evaluating our APELID method utilizing the CSE-CIC-IDS2018 dataset, augmented by CSE-CIC-IDS2018 by AWGAN, to train and evaluate AI models. In this scenario, we trained all five specialized models and incorporated them into the PELID model based on the GPU computing infrastructure mentioned above. After training, the CSE-CIC-IDS2018 test set is used to evaluate both the five single models and the ensemble model according to the five evaluation metrics. The detailed results of the CSE-CIC-IDS2018 experiment are illustrated in the first part of Table 4.3 and the confusion matrix shown in Table 4.1. In which, for the order of the labels from left to right in the predicted label, they correspond to the order of the true labels from top to bottom.

All five individual models evaluated in this study achieved an F1-score of 99.77% or above, indicating excellent performance. This demonstrates the excellent efficiency gains in intrusion detection made possible by data augmentation using the AWGAN algorithm.

These experiment results show that 1/17 $SQL-Injection$ attacks were identified as the $BruteForce-Web$, 3/17 $SQL-Injection$ attacks are denoted as the $BruteForce-XSS$, 1/17 $SQL-Injection$ attacks are defined as the $Infiltration$, 1/6,000 $DoS-GoldenEye$ attacks were identified as the $DoS-Slowloris$. All attacks flow (in 42,635 total attacks) are detected by the PELID. There are no false-positive in intrusion detection. In general, the F1 score for PELID is 99.99% and its value is the same for other metrics of Acc, Prec, and Rec.

4.3.1.2 NSL-KDD-based Results

Similar to the previous experiments with CSE-CIC-IDS2018, we also evaluate the APELID proposed method with the NSL-KDD dataset. However, NSL-KDD contains four classes and 41 features. 'Duration' was omitted from this dataset since it was deemed unnecessary among those attributes. The remaining 38 features are considered continuous variables in the DNN model, while two features, "protocol_type" and "service," are employed as

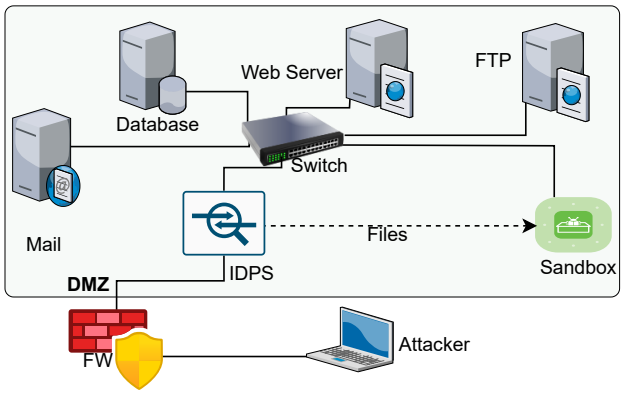


Figure 4.2: Malware Hunting Scenario

categorical variables. We train all five individual models with the NSL-KDD's augmented training set, just like we did with DS1. Both the five individual models and our PELID ensemble model are then evaluated by the NSL-KDD test set. The second part of Table 4.3 shows the experimental results by using NSL-KDD dataset, and Table 4.2 presents the PELID model's confusion matrix. In which, for the labels of the predicted label, it is similar to the confusion matrix of CSE-CIC-IDS2018 dataset.

The experimental findings shown in this scenario demonstrate the efficacy of individual models in intrusion detection, with F1 values of 98% or higher. In particular, we can see that the AWGAN algorithm has greatly improved the quality of the training data set by confirming that all of the AUC measurements are more than 99.85%.

For our PELID method, its confusion matrix illustrated in Table 4.2 shows that the FNR is 0.34%: total 30 network attacks (including 16 *Probe*, 6 *R2L* and 8 *U2R*) are not detected by the PELID, and the FPR is 0.37%: 22 *Benign* flows are considered as intrusions.

4.3.1.3 Malware Hunting Results

This experiment scenario is designed to assess the sandbox-based malware hunting of APELID method. Here, we utilize experimental data consisting of 80 benign and 20 malicious files. The benign files consist of Windows system software files obtained from a newly installed Windows virtual machine and downloaded from reputable Internet sources. The experimental study utilized downloaded malware files from public sources such as `https://bazaar.abuse.ch/` and `https://virustotal.com/`.

It is anticipated that the answer to our RQ3 will be a partial and temporary "YES" because it can use dynamic analysis to detect malware behavior. It provides proof of concept through the experiment that follows. To evaluate the intrusion detection capabilities of sandbox analysis, we conducted experiments on a host system equipped with Ubuntu 20.04 LTS, an Intel i7 CPU clocked at 2.3 GHz, and 8 GB of RAM.

This scenario includes two completely separate networks: DMZ Network (including Web server (HTTP and FTP), Mail Server (SNMP), and Attacks-Network), shown as Figure 4.2.

Table 4.4: Malware Hunting Results.

N	Malware	Type	Hash	VT	APELID
1	QuasarRAT	.exe	832ab3a898d188426d3541e1533b55f9	56/68	Yes
2	Loki	.xlsx	5b6aec60c3be4724f7980a659206531a	29/58	Yes
3	STRRAT	.jar	2199150e7d79d0e831cda314c7ce6f56	28/62	Yes
4	AsynRAT	.doc	da6419e4d4e4528990898bcfdaa85e01	32/60	Yes
5	SnakeKeylogger	.exe	715b0f6390ba4387a4155c1d59a3669c	49/69	Yes
6	AgentTesla	.exe	5c590fcb32aedec16532aa857eec28b5	40/66	Yes
7	OskiStealer	.xlsx	6a9203346218dded19d0a8a1dee24023	20/59	Yes
8	NanoCore	.exe	4bae18ac4a73ff38f7ed718365e6c2b2	41/67	Yes
9	DanaBot	.exe	5f4731a4ef7d1484893213caaf6a6685	42/69	Yes
10	DCRAT	.exe	ea800644b9dfd027807447fdd98241aa	50/68	Yes
11	YellowCockatoo	.dll	df7b2ece343c52df774d72e12ea09009	51/69	Yes
12	RemoteManipulator	.exe	4c5649e9b9a2d9997ac2600a804e0aeb	41/68	Yes
13	Pony	.exe	ab468a5b5cd9470c0895097efa2a687f	63/71	Yes
14	Stealc	.exe	cea30f806e644cebe48399eefa345e51	47/71	Yes
15	njRat	.exe	b17414d6949c2e013de14fdc268cfc89	65/71	Yes
16	RedLineStealer	.exe	8a61e10948c23a9a5c353d28b8738490	35/71	Yes
17	Guildma	.zip	8a61e10948c23a9a5c353d28b8738490	35/71	Yes
18	Gozi	.js	1df2e7a13459223b2cc55b93744add77	24/71	Yes
19	DarkTortilla	.exe	1c354a83f81063dc75612a9a7bd51225	54/71	Yes
20	VectorStealer	.xlsx	5b47098a17ecd534de15df03b12beacb	40/71	Yes

We used 100 files, including 80 normal and 20 malware, to send to the DMZ Network and to upload as administrator to the sandbox.

The IDPS automatically captured files transmitted between networks that used an unencrypted protocol such as FTP or HTTP. We wrote a Python tool to automatically check and send the files to the Sandbox-based analysis for malware hunting. We compared the experimental results with Virus Total (VT), shown in Table 4.4, indicating that our custom sandbox can detect common file types, such as .exe, .dll, and .jar. It demonstrates that we can use the sandbox to detect malware file transfer between networks and proactively hunt malware for suspicious files. Therefore, these results consolidate the affirmation of RQ3 by using sandbox-based dynamic analysis of APELID.

4.3.2 Evaluation

This subsection aims to analyze the experimental results in order to respond to the four research questions specified at the beginning of Section 2.5. We focus on evaluating the performance of both the WGAN algorithm for improving the quality of the training set and

the PELID method in terms of intrusion detection.

4.3.2.1 Efficacy of PELID in Intrusion Detection

With the data augmentation algorithm AWGAN and the ensemble learning method PELID, APELID achieves the outstanding performance of intrusion detection: 99.99% for Accuracy, Precision, F1-score, and Recall for the CSE-CIC-IDS2018 dataset. Moreover, for the NLS-KDD dataset, these evaluation metrics also are excellent values of 99.65% for all. Table 4.3 also shows that the AUC of all five single models is close to 100% for both datasets. In particular, the AUC of the PELID model gives 100% results with CSE-CIC-IDS2018 and 99.99% with NSL-KDD. These two values clearly demonstrate the ideal classification efficiency of the PELID model and basically allow it to handle the problem of data imbalance in the classes.

Compared with individual AI models, as illustrated in Table 4.3, it is clear that PELID gives outstanding F1 results from 0.04% (with the GBM model) to 2.24% (with DNN) with CSE-CIC-IDS2018 dataset, from 0.17% (with XGB, GBM, and BME) to 1.63% (with DNN) with NSL-KDD dataset. In addition, both FPR and FNR rates are lower than that of all five models DNN, XGB, CBT, GBM, and BME. These results privilege us to respond to **RQ1**: combining multiple AI models of PELID allow for improved network intrusion detection. With a very high F1 and both FPR and FNR less than 0.37%, PELID can detect unknown ntrusion attacks. In addition, the combination of five individual AI models in PELID will also increase its resilience to adversarial attacks. In addition, the PELID model trained by CSE-CIC-IDS2018 has nearly perfect intrusion detection performance: 99.99% F1-score, 100% AUC, and both zero FPR and FNR. Therefore, this model is selected for integration into our IDPS in order to improve the detection efficacy of eleven types of intrusions.

4.3.2.2 Efficacy of PELID in Time Consumption

By performing the individual AI models in parallel, PELID theoretically permits to reduce the execution time. Our experiments enable us to demonstrate it conclusively.

Figure 4.3 shows that the average time the PELID prediction of $14,703$ flows in the NSL-KDD testing set from six different runnings is $251.81ms$. Meanwhile, PELID needs an average of $950.48ms$ from six runnings for analyzing $42,635$ flows in the CSE-CIC-IDS2018 testing set. Therefore, the average time to investigate one flow of the PELID is 17.13 µs and 22.29 µs in NSL-KDD and CSE-CIC-IDS2018, respectively. These experiments also indicate that the PELID-based analysis of NSL-KDD is shorter than CSE-CIC-IDS2018. It comes from the fact that the NSL-KDD has fewer features and labels than CSE-CIC-IDS2018: 40/71 and 4/12, respectively.

The time consumption of PELID with CSE-CIC-IDS2018 and NSL-KDD can be used to determine the throughput of network traffics. Based on Figure 4.3, the PELID-based deep analysis can perform 44,863 flows/s for the model trained by CSE-CIC-IDS2018 and 58,377

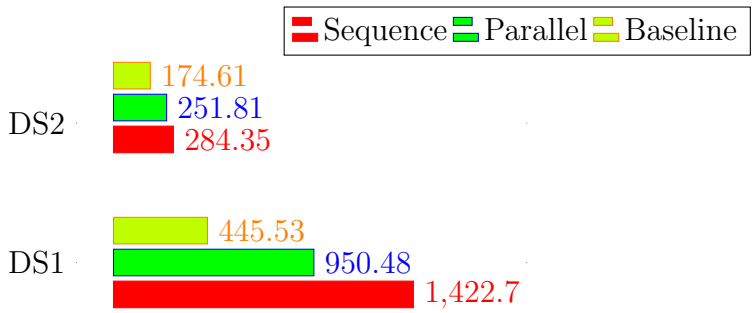


Figure 4.3: Comparing time consumption (milliseconds) between parallel and sequential processing of PELID. 'Baseline' illustrates the average execution time of five individual AI models.

flows/s for the model trained by NSL-KDD. By using notions of "mouse" and "elephant" flows of Alvarez-Horcajo et al. [12], we constate that the PELID can analyze the network throughput of $44,863 * 10KB \simeq 438MB/s \simeq 3.42Gps$ or $58,377 * 10KB \simeq 570MB/s \simeq 4.45Gbps$ for mouse flows (less than 10 KB/flow), respectively CSE-CIC-IDS2018 or NSL-KDD-based model. For elephant flows (more than 10 MB), PELID can reach up to $44,863 * 10MB \simeq 448,630MB/s = 3,504Gps$ or $58,377 * 10MB \simeq 583,770MB/s = 4,560Gbps$, respectively CSE-CIC-IDS2018 or NSL-KDD-based model. Therefore, PELID-based intrusion etection can be performed in large-scale networks. Consequently, **RQ2** has been resolved by all these experimental results.

The experiments in Table 4.3 show that the prediction in the CSE-CIC-IDS2018-based PELID has higher precision, precision, and F1 score than in NSL-KDD-based PELID. Therefore, we build an inline IDPS based on the Suricata solution and integrate the PELID model trained by the CSE-CIC-IDS2018 dataset. It is presently successfully deployed in inline mode to detect and prevent intrusions on our university's 10 Gbps large-scale network. It also demonstrated that the amount of time the PELID model requires to analyze a network traffic flow in parallel is fast enough to qualify as real-time in practice. Note that the open-source Cuckoo sandbox is also incorporated into our IDPS to hunt for malware and contribute more to **RQ3** responses.

Table 4.5: Comparison of APELID with SOTA Methods (%)

Method	Acc	Prec	F1	Rec
CSE-CIC-IDS2018				
APELID (our)	**99.99**	**99.99**	**99.99**	**99.99**
MMM-RF [47]	99.98	–	–	–
GAN+RF [62]	99.83	98.68	95.04	92.76
KNN-MQBHOA [36]	99.78	99.56	99.65	99.87
HDLNIDS [87]	98.90	98.63	99.03	99.14
CNN [75]	98.17	95.00	94.00	95.00
AUE [114]	97.90	98.00	98.00	98.00
miniVGGNet [68]	96.99	97.46	97.04	96.97
NSL-KDD				
APELID (our)	**99.65**	**99.65**	**99.63**	**99.65**
KNN-MQBHOA [36]	99.00	99.00	97.00	98.00
FFO-PNN [85]	98.99	96.97	96.97	96.97
DLNID [34]	90.73	86.38	89.65	93.17
GMM-WGAN-IDS [26]	86.59	88.55	86.88	86.59
Adaptive-Ensemble [35]	85.20	86.50	86.50	85.20
CAFE-CNN [98]	83.34	85.35	82.60	83.44

4.3.3 Comparison with SOTAs

The experimental findings of APELID are compared with those of other SOTA methods utilizing the same well-known datasets in order to evaluate our proposed approach. The comparison findings, which were reported directly from their published papers, are illustrated in Table 4.5. Note that these metrics are the macro average for the multi-label classification. Due to the lack of confusion matrix in most SOTA works, we cannot determine the FPR or FNR metric. However, the FNR, which can be derived from the True Positive Rate (TPR) or the Recall metric, can be used to compare the number of missing intrusion flows across various methods. It can be calculated using the following formula: $FNR = 1 - TPR = 1 - Recall$.

Table 4.5 demonstrates that APELID outperforms SOTA and achieves the greatest scores across all evaluation metrics. APELID achieves an F1-score of 99.99% and 99.65%, respectively, which is higher than all SOTA models based on CSE-CIC-IDS2018 and NSL-KDD. In addition, APELID achieves an outstanding true positive rate (or Recall) of 99.65% and an exceptional false negative rate (FNR) of 0.00% and 0.34% for these well-known datasets (as shown in Table 4.3). As a result, these comparisons enable us to validate the

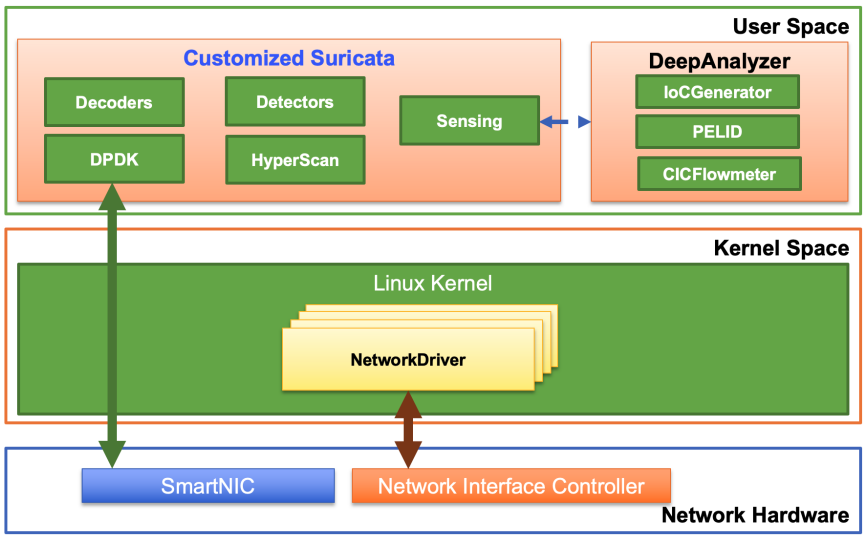


Figure 4.4: APELID-based NetIPS Architecture

efficacy of our APELID method and contribute more to answering **RQ1.**

4.4 NetIPS: Deployment of Network Intrusion Detection and Prevention

Based on the experimental findings and preceding explanation, it is evident that the PELID model may be effectively utilized to conduct a more comprehensive analysis of network traffic flows. Within this section, we provide in detail the practical implementation of the APELID method, utilizing the PELID-based deep analysis specifically designed to detect and prevent intrusions in large-scale networks effectively. The following subsections further outline the techniques employed in NetIPS, such as hypermatching and userspace-based analysis, to enhance the efficiency of deep analysis of network traffic flows.

4.4.1 Deployment Model

We built an IDPS named *NetIPS* to prove the ability of APELID in practice. Its architecture is illustrated in Figure 4.4 and divided into three layers. The lower layer is the network hardware, including SmartNIC (network accelerator) and traditional network interfaces, used to analyze traffic and manage the NetIPS. The middle layer, *Kernel Space*, controls the standard network hardware interfaces. In the top layer, *User Space*, we deploy the essential components of NetIPS: the customized Suricata as a Rule-based Detector and DeepInspector to perform the APELID-based intrusion detection.

The configuration of the appliance used in our real deployment is the same as the experimental environment described in Section 2.5. In this deployment model, Suricata v6.0.3 is used as the Rule-based Detector. However, in order to further analyze large-scale

network flows, we have customized Suricata to implement the flow sensing strategy for AI-powered deep analysis, as mentioned in Subsection 4.2.3.

For the DeepAnalyzer, the PELID model trained by CSE-CIC-IDS2018 is used for AI-powered intrusion detection. In combination with the flow sensing strategy implemented in the customized Suricata, our APELID method allows for efficient and effective real-time large-scale network traffic analysis for intrusion detection.

4.4.2 Hypermatching for Signature-based Detector

In the proposed system architecture, shown as Figure 4.4, when the DPDK (Data Plane Development Kit[1]) library packets are passed to the Decoder, they are decoded using a packet decoder and sent to the Detectors. In the Rule-based Detector, the Hyperscan technique is utilized to enhance the efficacy of the ruleset matching procedure. It matches more effectively than other methods (such as Aho-Corasick, Boyer-Moore).

To increase the efficacy of pattern matching, Hyperscan divides a regex pattern into multiple components and coordinates the order of component matching using fast string matching. It would reduce the number of wasteful CPU cycles caused by redundant matching, thereby improving performance. In addition, it provides multi-string matching and single finite automaton matching algorithms that take advantage of SIMD operations [106].

Each Detector module utilizes the Hyperscan library when analyzing and detecting network intrusion attacks using a combination of rule-based matching and in-depth analysis. It communicates with the PELID-based DeepInspector component via Unix sockets with the interprocess communication mechanism. In the case of sensing enabled, the PCAP data is sent to DeepAnalyzer for further analysis.

4.4.3 Accelerating AI-powered Intrusion Detection in User Space

The RX queue receives incoming packets for applications that employ network devices. The Direct Memory Access (DMA) mechanism transfers it to the main memory. The system must then be notified of the new packet and move the data into a buffer that has been specifically allotted (Linux allocates these buffers for each packet). For the Linux operating system, every new transmission the system receives requires a context-switching mechanism to allocate these buffers. The userspace networking system will handle the packet [66] in the subsequent phase.

The mechanism mentioned above causes congestion when more packets must be processed due to increased resource consumption, thereby decreasing the system's overall efficacy. Particularly, the packet allocation mechanism in the kernel requires numerous CPU-to-main-memory data transfer cycles. The data structure of the Linux network stack

[1]See more at https://www.dpdk.org

is typically compatible with as many protocols as feasible. The excessively complex stack slows down processing compared to simply analyzing network packets.

In addition to requiring a great deal of context switching, packet processing in the kernel has a negative impact on performance. When a userspace application must send or receive a payload, it makes a system call. The context is transferred from user mode to kernel mode and back again, consuming a significant amount of system resources.

To improve NetIPS performance for controlling large-scale network traffic, we use a Napatech SmartNIC NT40E3 4x10Gbps, instead of traditional NIC ports. All incoming traffic will be passed directly to the DPDK library [41] are then analyzed by NetIPS as follows:

1. Every incoming packet first goes to the ring buffer and then is passed to NetIPS via the DPDK library. It also takes the role of checking this buffer area for new packets received periodically.
2. If the ring buffer contains a new packet, NetIPS refers to the DPDK packet in the buffer, a specially allocated memory area using pointers.
3. If the ring buffer contains no packets, NetIPS will queue the network devices under the DPDK and refer to the buffer again.

In NetIPS, DeepInspector is also deployed in the userspace as an independent process. Therefore, NetIPS enhances the speed and reduces the latency of flow inspection for large-scale networks. The alert messages generated by DeepInspector are stored in a JSON file with the same structure as Suricata. New rules created by DeepInspector are represented as indicators of compromise (IoCs) and updated into the ruleset of Suricata.

Currently, our NetIPS is successfully deployed in inline mode to proactively detect and prevent intrusions on the large-scale network of the Vietnam National University (VNU, a federation of numerous universities) with a maximum throughput of 10 Gbps.

4.5 Summary

Chapter 4 addresses the critical challenge of deploying AI-powered intrusion detection and prevention systems in large-scale, real-world environments, where requirements for real-time performance, scalability, and operational reliability are paramount. Building upon the data enhancements and ensemble modeling innovations developed in previous chapters, this chapter introduces and evaluates a comprehensive architecture for practical, high-throughput network defense.

The chapter begins by identifying key bottlenecks encountered by traditional AI-based security systems when deployed in enterprise or large service provider networks, such as high computational latency, limitations due to sequential inference, and insufficient adaptability to sudden fluctuations in network traffic. These constraints significantly hinder the ability to operate at wire speed and respond promptly to emerging threats.

To overcome these issues, Chapter 4 proposes the design and implementation of NetIPS, a scalable intrusion prevention architecture that integrates several key innovations:

- Parallel ensemble inference: The system uses parallel processing techniques to concurrently execute multiple deep learning and boosting models, drastically reducing inference latency and allowing real-time analysis of high-speed data streams.
- Dynamic flow sensing strategy: NetIPS employs adaptive sensing methods to prioritize suspicious or high-risk traffic flows, ensuring efficient allocation of computational resources without compromising detection effectiveness.
- User-space Processing Architecture: Using user-space packet processing frameworks, the system achieves high flexibility and throughput, allowing seamless integration with existing network infrastructure.
- Integrated sandbox: A notable practical contribution is the ability to dynamically integrate a sandbox, where suspicious samples, especially files, are automatically transferred to an isolated environment (sandbox) for execution and behavioral analysis. This enables detection, deep analysis, and automatic labeling of new or unclear samples, while significantly enhancing the capability to detect zero-day malware and variants not previously seen in the training data. The sandbox feature also extends the pipeline from detection to root cause analysis and supports early response to advanced threats.

Evaluation metrics such as throughput, latency, and detection accuracy show that NetIPS consistently maintains wire-speed performance and high detection rates. Furthermore, comparative studies with other state-of-the-art systems demonstrate NetIPS's superior scalability and robustness, especially in scenarios involving sudden traffic surges or sophisticated multistage attacks.

These research results have been partially presented in published works, including two articles in respected journals (VVH-J1, VVH-J2) and one conference article (VVH-C1), highlighting the novel and significant contributions discussed in this chapter. Specifically, VVH-J1 and VVH-J2 describe the design, implementation, and real-world evaluation of the NetIPS system, a scalable real-time AI-powered intrusion prevention architecture that integrates parallel ensemble inference, dynamic flow sensing, user-space packet processing, and sandbox integration. VVH-C1 presents and analyzes the combination of rule-based and deep learning detection in practical environments. Together, these publications demonstrate the creativity and scientific value of the contributions of the chapter, laying the foundation for next-generation, real-world cybersecurity solutions.

Conclusions and Future Work

Contribution Highlights

Intrusion and malware attacks pose serious risks to modern digital infrastructure, primarily due to their ability to bypass traditional defense layers and evade detection using increasingly sophisticated techniques. These threats are often difficult to detect because they can mimic legitimate behavior and hide within large-scale network traffic or complex system environments. This dissertation approaches the problem from three main directions: (i) designing data-centric augmentation strategies to improve the quality and balance of machine learning datasets; (ii) developing robust hybrid ensemble frameworks that combine deep learning and boosting algorithms to achieve superior detection capabilities; and (iii) deploying large-scale network intrusion detection systems utilizing strategies using flow sensing, parallel execution, and sandbox. The dissertation leverages advances in deep learning and boosts learning and ensemble modeling to enhance overall model performance while increasing resilience in intrusion detection.

Our dissertation systematically surveys topics related to intrusion detection, including network attack techniques and threat detection, and synthesizes related work to identify existing research gaps. From there, the directions and objectives are clearly defined, focusing not only on balancing dataset and optimizing detection performance but also on ensuring scalability, transparency, and real-world deployment. Through the development and evaluation of augmentation dataset, feature optimization, mutual ensemble learning techniques, and the successful deployment of a real-time AI-powered intrusion prevention system. At the end of the study, the following contributions have clearly demonstrated the achievement of all research objectives.

- Propose a machine learning pipeline with data augmentation and feature optimization (WGAN-powered augmentation + SHAP-based feature optimization) to balance and enhance the quality of training datasets, thereby improving the detection capability for minority-class attacks.
- Introduce a deep and boosting mutual inference framework that strengthens the accuracy and resilience of intrusion and malware detection systems.
- Propose a solution to address data bottlenecks in large-scale network intrusion prevention through a time-interval and frequency-based flow sensing strategy, combined with parallelized inference of deep and boosting mutual inference models.
- Integrate the proposed methods into the NetIPS real-time intrusion detection and prevention system, which leverages AI-based models at the user level to process high-volume traffic (on a large scale), making it suitable for enterprise and ISP networks.

Dissertation Limitations

Although the research has yielded promising results, it is important to acknowledge several limitations.

- All tests were performed using fixed datasets that were prepared in advance, which means that we cannot see how well the model would adapt to real-life situations or when the data change over time.
- The NetIPS component has not yet been extensively validated in various real-world scenarios. In particular, comprehensive evaluations of hardware performance and deployment feasibility have not been conducted in large-scale production networks.
- The current experimental design does not include ablation studies to quantify the contribution of individual components or techniques to the overall performance. Such evaluations could provide more details on the effectiveness of the system and guide future optimizations.
- The models were trained primarily on structured network or PE data. More complex attack vectors, such as encrypted traffic, multistage malware, or supply chain attacks, were not within the scope of this study.

Future Research Directions

Building upon the foundations laid in this dissertation, several research directions are open for exploration:

- Online and continuous learning: Integrating online learning methods and incremental retraining into detection pipelines could allow models to adapt to evolving threats and handle dynamic environments more effectively.
- Future systems could use different types of data, such as how hosts behave, process trees, user activities, and patterns in encrypted traffic, all within a single detection framework.
- Automated response and defense integration: Improving detection systems with immediate actions, like automatically blocking threats, updating rules, or prioritizing alerts, can connect simple detection with active defense.
- Making it easier to understand decisions: Creating simple and user-friendly tools that explain how AI systems work, particularly for endpoint systems, can build trust and help security analysts work better with AI tools.

Personal Publications

Journals

VVH-J1 **Hoang V. Vo** and Hanh P. Du and Hoa N. Nguyen, AI-powered intrusion detection in large-scale traffic networks based on flow sensing strategy and parallel deep analysis, *Journal of Network and Computer Applications* 220 (2023) 103735. DOI: 10.1016/j.jnca.2023.103735; (IF 8.0, SCI-E, top 2% Q1-Scopus)

VVH-J2 **Hoang V. Vo** and Hanh P. Du and Hoa N. Nguyen, APEPID: Enhancing real-time intrusion detection with augmented WGAN and parallel ensemble learning, *Computers and Security* 136 (2024) 103567. DOI: 10.1016/j.cose.2023.103567; (IF 5.4, SCI-E, top 7% Q1-Scopus)

VVH-J3 **Hoang V. Vo** and Hanh P. Du and Hoa N. Nguyen, MDOB: Enhancing Resilient and Explainable AI-Powered Malware Detection Using Feature Set Optimization and Mutual Deep+Boosting Ensemble Inference. *Journal of Information Security and Applications* 2025 93 (2025) 104175. DOI: 10.1016/j.jisa.2025.104175; (IF 3.7, SCI-E, top 8% Q1-Scopus)

Conferences

VVH-C1 **Hoang V. Vo**, Hoa N. Nguyen, Tu N. Nguyen, Hanh P. Du, *SDAID: Towards a Hybrid Signature and Deep Analysis-based Intrusion Detection Method*, in: GLOBECOM 2022 - 2022 IEEE Global Communications Conference, 2022, pp. 2615–2620. DOI: 10.1109/GLOBECOM48 099.2022.10001582. (WoS, Scopus)

VVH-C2 **Hoang V. Vo**, Duong H. Nguyen, Tuyen T. Nguyen, Hoa N. Nguyen, Duan V. Nguyen, *Leveraging AI-Driven Realtime Intrusion Detection by Using WGAN and XGBoost*, in: Proceedings of the 11th International Symposium on Information and Communication Technology, Association for Computing Machinery, New York, NY, USA, 2022, p. 208–215. DOI: 10.1145/3568562.3568660. (WoS, Scopus)

VVH-C3 **Hoang V. Vo**, Phong H. Nguyen, Hau T. Nguyen, Duy B. Vu, Hoa N. Nguyen, *Enhancing AI-Powered Malware Detection by Parallel Ensemble Learning*, in: 2023 RIVF International Conference on Computing and Communication Technologies (RIVF), 2023, pp. 503–508. DOI: 10.1109/RIVF60135.2023.10471855. (WoS)

VVH-C4 **Hoang V. Vo**, Hanh P. Du and Hoa N. Nguyen, *AWDLID: Augmented WGAN and Deep Learning for Improved Intrusion Detection*, 2024 1st International Conference On Cryptography And Information Security (VCRIS), Hanoi, Vietnam, 2024, pp. 1-6, DOI: 10.1109/VCRIS63677.2024.10813392. (WoS)

Bibliography

[1] Mahmoud Abbasi, Amin Shahraki, and Amir Taherkordi. Deep learning for network traffic monitoring and analysis (ntma): A survey. Computer Communications, 170: 19–41, 2021. ISSN 0140-3664. doi: https://doi.org/10.1016/j.comcom.2021.01.021.

[2] Razan Abdulhammed, Miad Faezipour, Abdelshakour Abuzneid, and Arafat Abumallouh. Deep and machine learning approaches for anomaly-based intrusion detection of imbalanced network traffic. IEEE Sensors Letters, 3:1–4, 01 2019. doi: 10.1109/LSENS.2018.2879990.

[3] Abebe Abeshu and Naveen Chilamkurti. Deep learning: The frontier for distributed attack detection in fog-to-things computing. IEEE Communications Magazine, 56(2): 169–175, 2018. doi: 10.1109/MCOM.2018.1700332.

[4] Claire Adam-Bourdarios, Glen Cowan, Cécile Germain, Isabelle Guyon, Balázs Kégl, and David Rousseau. The higgs boson machine learning challenge. In Proceedings of the 2014 International Conference on High-Energy Physics and Machine Learning - Volume 42, HEPML'14, page 19–55. JMLR.org, 2014.

[5] Priyanka Aggarwal, Sayyed F. Ahamed, Sachin Shetty, and Laura J. Freeman. Selective targeted transfer learning for malware classification. In 2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA), pages 114–120, 2021. doi: 10.1109/TPSISA52974.2021.00013.

[6] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD '19, page 2623–2631, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330701.

[7] Samed Al and Murat Dener. Stl-hdl: A new hybrid network intrusion detection system for imbalanced dataset on big data environment. Computers & Security, 110: 102435, 2021. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2021.102435.

[8] Mohammed Y. Aldarwbi, Arash H. Lashkari, and Ali A. Ghorbani. The sound of intrusion: A novel network intrusion detection system. Computers and Electrical Engineering, 104:108455, 2022. ISSN 0045-7906. doi: 10.1016/j.compeleceng.2022.108455.

[9] Abdullah Alghamdi and Ayad Barsoum. A comprehensive ids to detect botnet attacks using machine learning techniques. In 2024 IEEE 3rd International Conference on Computing and Machine Intelligence (ICMI), pages 1–6, 2024. doi: 10.1109/ICMI60790.2024.10585846.

[10] Fahad T. ALGorain and Abdulrahman S. Alnaeem. Deep learning optimisation of static malware detection with grid search and covering arrays. Telecom, 4(2):249–264, 2023. ISSN 2673-4001. doi: 10.3390/telecom4020015.

[11] Khaled Alrawashdeh and Carla Purdy. Toward an online anomaly intrusion detection system based on deep learning. In 2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA), pages 195–200, 2016. doi: 10.1109/ICMLA.2016.0040.

[12] Joaquin Alvarez-Horcajo, Diego Lopez-Pajares, José M. Arco, Juan Antonio Carral, and Isaías Martinez-Yelmo. Tcp-path: Improving load balance by network exploration. In 6th IEEE International Conference on Cloud Networking, CloudNet 2017, Prague, Czech Republic, September 25-27, 2017, pages 65–70. IEEE, 2017. doi: 10.1109/CloudNet.2017.8071533.

[13] H. S. Anderson and P. Roth. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. ArXiv e-prints, April 2018.

[14] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. Machine Learning, 2017.

[15] Ömer Aslan and Abdullah Asim Yilmaz. A new malware classification framework based on deep learning algorithms. IEEE Access, 9:87936–87951, 2021. doi: 10.1109/ACCESS.2021.3089586.

[16] Aitor Belenguer, Jose A. Pascual, and Javier Navaridas. Göwfed: A novel federated network intrusion detection system. Journal of Network and Computer Applications, page 103653, 2023. ISSN 1084-8045. doi: https://doi.org/10.1016/j.jnca.2023.103653.

[17] Sonam Bhardwaj, Adrian Shuai Li, Mayank Dave, and Elisa Bertino. Overcoming the lack of labeled data: Training malware detection models using adversarial domain adaptation. Computers & Security, 140:103769, 2024. ISSN 0167-4048. doi: 10.1016/j.cose.2024.103769.

[18] Bhoopesh Bhati, Garvit Chugh, Fadi Al-Turjman, and Nitesh Bhati. An improved ensemble based intrusion detection technique using xgboost. Transactions on Emerging Telecommunications Technologies, 32, 06 2021. doi: 10.1002/ett.4076.

[19] Loïc Bontemps, Van Loi Cao, James McDermott, and Nhien-An Le-Khac. Collective anomaly detection based on long short term memory recurrent neural network. CoRR, abs/1703.09752, 2017.

[20] Austin Brown, Maanak Gupta, and Mahmoud Abdelsalam. Automated machine learning for deep learning based malware detection. Computers & Security, 137: 103582, 2024. ISSN 0167-4048. doi: 10.1016/j.cose.2023.103582.

[21] Marta Catillo, Massimiliano Rak, and Villano Umberto. 2l-zed-ids: A two-level anomaly detector for multiple attack classes. In Web, Artificial Intelligence and Network Applications, WAINA2020, Advances in Intelligent Systems and Computing, pages 687–696. Springer International Publishing, 2020. ISBN 978-3-030-44037-4. doi: 10.1007/978-3-030-44038-1_63.

[22] Radhika Chapaneri and Seema Shah. Enhanced detection of imbalanced malicious network traffic with regularized generative adversarial networks. Journal of Network and Computer Applications, 202:103368, 2022. ISSN 1084-8045. doi: https://doi.org/10.1016/j.jnca.2022.103368.

[23] Nitesh Chawla, Kevin Bowyer, Lawrence Hall, and W. Kegelmeyer. Smote: Synthetic minority over-sampling technique. J. Artif. Intell. Res. (JAIR), 16:321–357, 06 2002. doi: 10.1613/jair.953.

[24] Jiaqi Chen, Chong Yuan, Jiashuo Li, Donghai Tian, Rui Ma, and Xiaoqi Jia. Elamd: An ensemble learning framework for adversarial malware defense. Journal of Information Security and Applications, 75:103508, 2023. ISSN 2214-2126. doi: 10.1016/j.jisa.2023.103508.

[25] Ratul Chowdhury, Shibaprasad Sen, Arpan Goswami, Shankhadeep Purkait, and Banani Saha. An implementation of bi-phase network intrusion detection system by using real-time traffic analysis. Expert Systems with Applications, 224:119831, 2023. ISSN 0957-4174. doi: 10.1016/j.eswa.2023.119831.

[26] Jiyuan Cui, Liansong Zong, Jianhua Xie, and Mingwei Tang. A novel multi-module integrated intrusion detection system for high-dimensional imbalanced data. Applied Intelligence, 04 2022. doi: 10.1007/s10489-022-03361-2.

[27] Preethi Devan and Neelu Khare. An efficient xgboost–dnn-based classification model for network intrusion detection system. Neural Computing and Applications, 32(16): 12499–12514, aug 2020. ISSN 0941-0643. doi: 10.1007/s00521-020-04708-x.

[28] Jingyi Ding, Ziqing Chen, Li Xiaolong, and Baoxin Lai. Sales forecasting based on catboost. In 2020 2nd International Conference on Information Technology and Computer Application (ITCA), pages 636–639, 2020. doi: 10.1109/ITCA52113.2020.00138.

[29] Usha Divakarla, K Hemant Kumar Reddy, and K Chandrasekaran. A novel approach towards windows malware detection system using deep neural networks. Procedia Computer Science, 215:148–157, 2022. ISSN 1877-0509. doi: 10.1016/j.procs.2022.12.017. 4th International Conference on Innovative Data Communication Technology and Application.

[30] Ghanshyam Prasad Dubey and Dr. Rakesh Kumar Bhujade. Optimal feature selection for machine learning based intrusion detection system by exploiting attribute dependence. Materials Today: Proceedings, 47:6325–6331, 2021. ISSN 2214-7853. doi: 10.1016/j.matpr.2021.04.643. SI: TIME-2021.

[31] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. Autogluon-tabular: Robust and accurate automl for structured data. arXiv Machine Learning, 2020. doi: 10.48550/arXiv.2003.06505.

[32] Rawaa Farhan, Abeer Tariq, and Nidaaflaih Hassan. Performance analysis of flow-based attacks detection on cse-cic-ids2018 dataset using deep learning deep learning flow-based intrusion detection internet of thing (iot). Indonesian Journal of Electrical Engineering and Computer Science, 20:1413–1418, 12 2020. doi: 10.11591/ijeecs.v20.i3.pp1413-1418.

[33] Mohamed Amine Ferrag, Leandros Maglaras, Sotiris Moschoyiannis, and Helge Janicke. Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study. Journal of Information Security and Applications, 50, 12 2019. doi: 10.1016/j.jisa.2019.102419.

[34] Yanfang Fu, Yishuai Du, Zijian Cao, Qiang Li, and Wei Xiang. A deep learning model for network intrusion detection with imbalanced data. Electronics, 11:898, 03 2022. doi: 10.3390/electronics11060898.

[35] Xianwei Gao, Chun Shan, Changzhen Hu, Zequn Niu, and Zhen Liu. An adaptive ensemble machine learning model for intrusion detection. IEEE Access, 7:82512–82521, 2019. doi: 10.1109/ACCESS.2019.2923640.

[36] Reza Ghanbarzadeh, Ali Hosseinalipour, and Ali Ghaffari. A novel network intrusion detection method based on metaheuristic optimisation algorithms. Journal of Ambient Intelligence and Humanized Computing, pages 1–18, 03 2023. doi: 10.1007/s12652-023-04571-3.

[37] Patrice Godefroid, Hila Peleg, and Rishabh Singh. Learn&fuzz: machine learning for input fuzzing. In Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, page 50–59. IEEE Press, 2017. ISBN 9781538626849. doi: 10.5555/3155562.3155573.

[38] Roopa Golchha, Apoorv Joshi, and Govind Prasad Gupta. Voting-based ensemble learning approach for cyber attacks detection in industrial internet of things. Procedia Computer Science, 218:1752–1759, 2023. ISSN 1877-0509. doi: 10.1016/j.procs.2023.01.153. International Conference on Machine Learning and Data Engineering.

[39] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. 2016.

[40] Arnaldo Gouveia and Miguel Pupo Correia. Recent Advances in Security, Privacy, and Trust for Internet of Things (IoT) and Cyber-Physical Systems (CPS), chapter Network Intrusion Detection with XGBoost, pages 150–156. Chapman and Hall/CRC, 1st. edition, 2020. ISBN 9780429270567.

[41] Bingli Guo, Yu Shang, Yunquan Zhang, Wenzhe Li, Shan Yin, Yongjun Zhang, and Shanguo Huang. Timeslot switching-based optical bypass in data center for intrarack elephant flow with an ultrafast dpdk-enabled timeslot allocator. Journal of Lightwave Technology, 37(10):2253–2260, 2019. doi: 10.1109/JLT.2019.2901600.

[42] You Guo, Hector Marco-Gisbert, and Paul Keir. Mitigating webshell attacks through machine learning techniques. Future Internet, 12:12, 01 2020. doi: 10.3390/fi12010012.

[43] Neha Gupta, Vinita Jindal, and Punam Bedi. Cse-ids: Using cost-sensitive deep learning and ensemble algorithms to handle class imbalance in network-based intrusion detection systems. Computers & Security, 112:102499, 10 2021. doi: 10.1016/j.cose.2021.102499.

[44] Arash Habibi Lashkari. Cicflowmeter-v4.0: a network traffic bi-flow generator and analyser for anomaly detection. https://github.com/iscx/cicflowmeter, 08 2018.

[45] Hassan Jalil Hadi, Yue Cao, Sifan Li, Naveed Ahmad, and Mohammed Ali Alshara. Fcg-mfd: Benchmark function call graph-based dataset for malware family detection. Journal of Network and Computer Applications, 233:104050, 2025. ISSN 1084-8045. doi: 10.1016/j.jnca.2024.104050.

[46] Hashida Haidros Rahima Manzil and Manohar Naik S. Dynamaldroid: Dynamic analysis-based detection framework for android malware using machine learning techniques. In 2022 International Conference on Knowledge Engineering and Communication Systems (ICKES), pages 1–6, 2022. doi: 10.1109/ICKECS56523.2022.10060106.

[47] Mohamed Hammad, Nabil Hewahi, and Wael Elmedany. Mmm-rf: A novel high accuracy multinomial mixture model for network intrusion detection systems. Computers & Security, 120:102777, 2022. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2022.102777.

[48] Jingwei Hao, Senlin Luo, and Limin Pan. Eii-mbs: Malware family classification via enhanced adversarial instruction behavior semantic learning. Computers & Security, 122:102905, 2022. ISSN 0167-4048. doi: 10.1016/j.cose.2022.102905.

[49] Jamal Hussain and Samuel Lalmuanawma. Feature analysis, evaluation and comparisons of classification algorithms based on noisy intrusion dataset. volume 92, pages 188–198, 2016. doi: 10.1016/j.procs.2016.07.345. 2nd International Conference on Intelligent Computing, Communication & Convergence, ICCC 2016, India.

[50] Sumaiya Thaseen Ikram, Aswani Kumar Cherukuri, Babu Poorva, Pamidi Sai Ushasree, Yishuo Zhang, Xiao Liu, and Gang Li. Anomaly detection using xgboost ensemble of deep neural network models. Cybernetics and Information Technologies, 21(3):175–188, sep 2021. ISSN 1314-4081. doi: 10.2478/cait-2021-0037.

[51] P.L.S. Jayalaxmi, Rahul Saha, Gulshan Kumar, Mamoun Alazab, Mauro Conti, and Xiaochun Cheng. Pignus: A deep learning model for ids in industrial internet-of-things. Computers & Security, page 103315, 2023. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2023.103315.

[52] Feng Jiang, Yunsheng Fu, B B Gupta, Fang Lou, Seungmin Rho, Fanzhi Meng, and Zhihong Tian. Deep learning based multi-channel intelligent attack detection for data security. IEEE Transactions on Sustainable Computing, 5:1–1, 01 2018. doi: 10.1109/TSUSC.2018.2793284.

[53] Gülsade Kale, Gazi Erkan Bostancı, and Fatih Vehbi Çelebi. Evolutionary feature selection for machine learning based malware classification. Engineering Science and Technology, an International Journal, 56:101762, 2024. ISSN 2215-0986. doi: 10.1016/j.jestch.2024.101762.

[54] Gautam Karat, Jinesh M. Kannimoola, Namrata Nair, Anu Vazhayil, Sujadevi V G, and Prabaharan Poornachandran. Cnn-lstm hybrid model for enhanced malware analysis and detection. Procedia Computer Science, 233:492–503, 2024. ISSN 1877-0509. doi: 10.1016/j.procs.2024.03.239. 5th International Conference on Innovative Data Communication Technologies and Application (ICIDCA 2024).

[55] Gozde Karatas, Onder Demir, and Koray Sahingoz. Increasing the performance of machine learning-based idss on an imbalanced and up-to-date dataset. IEEE Access, 8:32150–32162, 2020. doi: 10.1109/ACCESS.2020.2973219.

[56] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In Proceedings of the 31st International Conference on Neural Information Processing Systems, page 3149–3157, 2017. ISBN 9781510860964.

[57] Murad Ali Khan, Naeem Iqbal, Imran, Harun Jamil, and Do-Hyeun Kim. An optimized ensemble prediction model using automl based on soft voting classifier for network intrusion detection. Journal of Network and Computer Applications, 212: 103560, 2023. ISSN 1084-8045. doi: https://doi.org/10.1016/j.jnca.2022.103560.

[58] Ansam Khraisat, Iqbal Gondal, Peter Vamplew, and Joarder Kamruzzaman. Survey of intrusion detection systems: techniques, datasets and challenges. Cybersecurity, 2 (1):20, Jul 2019. ISSN 2523-3246. doi: 10.1186/s42400-019-0038-7.

[59] Sumit Lad and Amol Adamuthe. Improved deep learning model for static pe files malware detection and classification. International Journal of Computer Network and Information Security, 14:14–26, 04 2022. doi: 10.5815/ijcnis.2022.02.02.

[60] Giap V. Le, Tung H. Nguyen, Phuc D. Pham, On V. Phung, and Hoa N. Nguyen. Guruws: A hybrid platform for detecting malicious web shells and web application vulnerabilities. Transactions on Computational Collective Intelligence, 11370:184–208, 2019. doi: 10.1007/978-3-662-58611-2_5.

[61] Ha V. Le, Tu N. Nguyen, Hoa N. Nguyen, and Linh Le. An efficient hybrid webshell detection method for webserver of marine transportation systems. IEEE Transactions on Intelligent Transportation Systems, 24(2):2630–2642, 2023. doi: 10.1109/TITS.2021.3122979.

[62] JooHwa Lee and KeeHyun Park. Gan-based imbalanced data intrusion detection system. Personal and Ubiquitous Computing, 25, 02 2021. doi: 10.1007/s00779-019-01332-y.

[63] Sang-Woong Lee, Haval Mohammed sidqi, Mokhtar Mohammadi, Shima Rashidi, Amir Masoud Rahmani, Mohammad Masdari, and Mehdi Hosseinzadeh. Towards secure intrusion detection systems using deep learning techniques: Comprehensive analysis and review. Journal of Network and Computer Applications, 187:103111, 2021. ISSN 1084-8045. doi: https://doi.org/10.1016/j.jnca.2021.103111.

[64] Joffrey Leevy and Taghi Khoshgoftaar. A survey and analysis of intrusion detection models based on cse-cic-ids2018 big data. Journal of Big Data, 7, 11 2020. doi: 10.1186/s40537-020-00382-x.

[65] Yanan Li, Tao Qin, Yongzhong Huang, Jinghong Lan, ZanHao Liang, and Tongtong Geng. Hdfef: A hierarchical and dynamic feature extraction framework for intrusion detection systems. Computers & Security, 121:102842, 2022. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2022.102842.

[66] Yi Li, Kaiqi Xiong, Tommy Chin, and Chengbin Hu. A machine learning framework for domain generation algorithm-based malware detection. IEEE Access, 7:32765–32782, 2019. doi: 10.1109/ACCESS.2019.2891588.

[67] Peng Lin, Kejiang Ye, and Cheng-Zhong Xu. Dynamic network anomaly detection system by using deep learning techniques. In Cloud Computing – CLOUD 2019, page 161–176. Springer-Verlag. doi: 10.1007/978-3-030-23502-4_12.

[68] Lan Liu, Pengcheng Wang, Jun Lin, and Langzhou Liu. Intrusion detection of imbalanced network traffic based on machine learning and deep learning. IEEE Access, 9:7550–7563, 2021. doi: 10.1109/ACCESS.2020.3048198.

[69] Songsong Liu, Pengbin Feng, Shu Wang, Kun Sun, and Jiahao Cao. Enhancing malware analysis sandboxes with emulated user behavior. Computers & Security, 115:102613, 2022. ISSN 0167-4048. doi: 10.1016/j.cose.2022.102613.

[70] Xinbo Liu, Yaping Lin, He Li, and Jiliang Zhang. A novel method for malware detection on ml-based visualization technique. Computers & Security, 89:101682, 2020. ISSN 0167-4048. doi: 10.1016/j.cose.2019.101682.

[71] Xu-Ying Liu, Jianxin Wu, and Zhi-Hua Zhou. Exploratory undersampling for class-imbalance learning. IEEE Transactions on Systems, Man, and Cybernetics, Part B, 39(2):539–550, 2009. doi: 10.1109/TSMCB.2008.2007853.

[72] Maya Hilda Lestari Louk and Bayu Adhi Tama. Dual-ids: A bagging-based gradient boosting decision tree model for network anomaly intrusion detection system. Expert Systems with Applications, 213:119030, 2023. ISSN 0957-4174. doi: 10.1016/j.eswa.2022.119030.

[73] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In Proceedings of the 31st International Conference on Neural Information Processing Systems, page 4768–4777, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964. doi: 10.5555/3295222.3295230.

[74] Benjamin Marais, Tony Quertier, and Christophe Chesneau. Malware analysis with artificial intelligence and a particular attention on results interpretability. In Distributed Computing and Artificial Intelligence, Volume 1: 18th International Conference, pages 43–55. Springer International Publishing, 2022. ISBN 978-3-030-86261-9. doi: 10.1007/978-3-030-86261-9_5.

[75] Mariama Mbow, Hiroshi Koide, and Kouichi Sakurai. Handling class imbalance problem in intrusion detection system based on deep learning. International Journal of Networking and Computing, 12:467–492, 2022. ISSN 2185-2847.

[76] Mamoru Mimura. Evaluation of printable character-based malicious pe file-detection method. Internet of Things, 19:100521, 2022. ISSN 2542-6605. doi: 10.1016/j.iot.2022.100521.

[77] Preeti Mishra, Vijay Varadharajan, Uday Tupakula, and Emmanuel S. Pilli. A detailed investigation and analysis of using machine learning techniques for intrusion detection. IEEE Communications Surveys & Tutorials, 21(1):686–728, 2019. doi: 10.1109/COMST.2018.2847722.

[78] Gowtham Muniraju, Bhavya Kailkhura, Jayaraman J. Thiagarajan, Peer-Timo Bremer, Cihan Tepedelenlioglu, and Andreas Spanias. Coverage-based designs improve sample mining and hyperparameter optimization. IEEE Transactions on Neural Networks and Learning Systems, 32(3):1241–1253, 2021. doi: 10.1109/TNNLS.2020.2982936.

[79] Trivikram Muralidharan, Aviad Cohen, Noa Gerson, and Nir Nissim. File packing from the malware perspective: Techniques, analysis approaches, and directions for enhancements. ACM Computing Surveys, 55(5), 2022. ISSN 0360-0300. doi: 10.1145/3530810.

[80] Sercan Gulburun Murat Dener. Clustering-aided supervised malware detection with specialized classifiers and early consensus. Computers, Materials & Continua, 75(1):1235–1251, 2023. ISSN 1546-2226. doi: 10.32604/cmc.2023.036357.

[81] Nilambari G. Narkar and Narendra M. Shekokar. A rule based intrusion detection system to identify vindictive web spider. In 2016 International Conference on Computing, Analytics and Security Trends (CAST), pages 271–275, 2016. doi: 10.1109/CAST.2016.7914979.

[82] Anjum Nazir and Rizwan Ahmed Khan. A novel combinatorial optimization based feature selection method for network intrusion detection. Computers & Security, 102:102164, 2021. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2020.102164.

[83] T. N. Nguyen, B.-H. Liu, N. Nguyen, B. Dumba, and J.-T. Chou. Smart grid vulnerability and defense analysis under cascading failure attacks. IEEE Transactions on Power Delivery, 36(4):2264–2273, 2021. doi: 10.1109/TPWRD.2021.3061358.

[84] Tuyen T. Nguyen, Phong H. Nguyen, Minh Q. Nguyen, and Hoa N. Nguyen. Tabgan-powered data augmentation and explainable boosting-based ensemble learning for intrusion detection in industrial control systems. In Computational Collective Intelligence, pages 123–136. Springer Nature Switzerland, 2024. ISBN 978-3-031-70819-0. doi: 10.1007/978-3-031-70819-0_10.

[85] Nadir Omer, Ahmed H. Samak, Ahmed I. Taloba, and Rasha M. Abd El-Aziz. A novel optimized probabilistic neural network approach for intrusion detection and categorization. Alexandria Engineering Journal, 72:351–361, 2023. ISSN 1110-0168. doi: 10.1016/j.aej.2023.03.093.

[86] Jakub Palša, Norbert Ádám, Ján Hurtuk, Eva Chovancova, Branislav Madoš, Martin Chovanec, and Stanislav Kocan. Mlmd—a malware-detecting antivirus tool based on the xgboost machine learning algorithm. Applied Sciences, 12:6672, 07 2022. doi: 10.3390/app12136672.

[87] Emad Qazi, Muhammad Faheem, and Tanveer Zia. Hdlnids: Hybrid deep-learning-based network intrusion detection system. Applied Sciences, 13:4921, 04 2023. doi: 10.3390/app13084921.

[88] Quarkslab. Lief documentation. Available at: https://lief.quarkslab.com/ (accessed May 2025).

[89] Edward Raff, Jared Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles Nicholas. Malware detection by eating a whole exe. arXiv Machine Learning (stat.ML), 2018. doi: 10.48550/arXiv.1710.09435.

[90] Ajeet Rai. Optimizing a new intrusion detection system using ensemble methods and deep neural network. pages 527–532, 2020. doi: 10.1109/ICOEI48184.2020.9143028.

[91] Smitha Rajagopal, Poornima Kundapur, and Hareesha S. A stacking ensemble for network intrusion detection using heterogeneous datasets. Security and Communication Networks, pages 1–9, 01 2020. doi: 10.1155/2020/4586875.

[92] Keyan Ren, Shuai Yuan, Chun Zhang, Yu Shi, and Zhiqing Huang. Canet: A hierarchical cnn-attention model for network intrusion detection. Computer Communications, 2023. ISSN 0140-3664. doi: 10.1016/j.comcom.2023.04.018.

[93] Eréndira Rendón, Roberto Alejo, Carlos Castorena, Frank J. Isidro-Ortega, and Everardo E. Granda-Gutiérrez. Data sampling methods to deal with the big data multi-class imbalance problem. Applied Sciences, 10(4), 2020. ISSN 2076-3417. doi: 10.3390/app10041276.

[94] M. Rigaki and S. Garcia. Stealing and evading malware classifiers and antivirus at low false positive conditions. Computers & Security, 129:103192, 2023. ISSN 0167-4048. doi: 10.1016/j.cose.2023.103192.

[95] Ethan M. Rudd, David Krisiloff, Scott Coull, Daniel Olszewski, Edward Raff, and James Holt. Efficient malware analysis using metric embeddings. Digital Threats, 5 (1), mar 2024. doi: 10.1145/3615669.

[96] Joshua Saxe and Konstantin Berlin. Deep neural network based malware detection using two dimensional binary program features. In 2015 10th International Conference on Malicious and Unwanted Software (MALWARE), pages 11–20, 2015. doi: 10.1109/MALWARE.2015.7413680.

[97] Mahmoud Said El Sayed, Nhien-An Le-Khac, Marianne A. Azer, and Anca D. Jurcut. A flow-based anomaly detection approach with feature selection method against ddos attacks in sdns. IEEE Transactions on Cognitive Communications and Networking, 8(4):1862–1880, 2022. doi: 10.1109/TCCN.2022.3186331.

[98] Erfan Shams, Ahmet Rizaner, and Ali Ulusoy. A novel context-aware feature extraction method for convolutional neural network-based intrusion detection systems. Neural Computing and Applications, 33:1–19, 10 2021. doi: 10.1007/s00521-021-05994-9.

[99] Jay Sinha and M. Manollas. Efficient deep cnn-bilstm model for network intrusion detection. In Proceedings of the 2020 3rd International Conference on Artificial Intelligence and Pattern Recognition, AIPR '20, page 223–231, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450375511. doi: 10.1145/3430199.3430224.

[100] Romain Thomas. Lief - library to instrument executable formats. https://lief.quarkslab.com/, apr 2017.

[101] Farhan Ullah, Shamsher Ullah, Gautam Srivastava, and Jerry Chun-Wei Lin. Ids-int: Intrusion detection system using transformer-based transfer learning for imbalanced network traffic. Digital Communications and Networks, 2023. ISSN 2352-8648. doi: 10.1016/j.dcan.2023.03.008.

[102] Laurens van der Maaten and Geoffrey Hinton. Viualizing data using t-sne. Journal of Machine Learning Research, 9:2579–2605, 11 2008.

[103] Mihai Vasilescu, Laura Gheorghe, and Nicolae Tapus. Practical malware analysis based on sandboxing. In 2014 RoEduNet Conference 13th Edition: Networking in Education and Research Joint Event RENAM 8th Conference, pages 1–6, 2014. doi: 10.1109/RoEduNet-RENAM.2014.6955304.

[104] Parag Verma, Shayan Anwar, Shadab Khan, and Sunil B Mane. Network intrusion detection using clustering and gradient boosting. In 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), pages 1–7, 2018. doi: 10.1109/ICCCNT.2018.8494186.

[105] R. Vinayakumar, Mamoun Alazab, K. P. Soman, Prabaharan Poornachandran, Ameer Al-Nemrat, and Sitalakshmi Venkatraman. Deep learning approach for intelligent intrusion detection system. IEEE Access, 7:41525–41550, 2019. doi: 10.1109/ACCESS.2019.2895334.

[106] Xiang Wang, Yang Hong, Harry Chang, KyoungSoo Park, Geoff Langdale, Jiayu Hu, and Heqing Zhu. Hyperscan: A fast multi-pattern regex matcher for modern CPUs. In

16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19), pages 631–648, Boston, MA, February 2019. USENIX Association. ISBN 978-1-931971-49-2.

[107] Dong-Ok Won, Yong-Nam Jang, and Seong-Whan Lee. Plausmal-gan: Plausible malware training based on generative adversarial networks for analogous zero-day malware detection. IEEE Transactions on Emerging Topics in Computing, 11(1): 82–94, 2023. doi: 10.1109/TETC.2022.3170544.

[108] Limin Yang, Arridhana Ciptadi, Ihar Laziuk, Ali Ahmadzadeh, and Gang Wang. Bodmas: An open dataset for learning based temporal analysis of pe malware. In 2021 IEEE Security and Privacy Workshops (SPW), pages 78–84, 2021. doi: 10.1109/SPW53761.2021.00020.

[109] Tao Yi, Xingshu Chen, Yi Zhu, Weijing Ge, and Zhenhui Han. Review on the application of deep learning in network attack detection. Journal of Network and Computer Applications, 212:103580, 2023. ISSN 1084-8045. doi: https://doi.org/10.1016/j.jnca.2022.103580.

[110] Xiao-Dong Zeng, Sam Chao, and Fai Wong. Optimization of bagging classifiers based on sbcb algorithm. In 2010 International Conference on Machine Learning and Cybernetics, volume 1, pages 262–267, 2010. doi: 10.1109/ICMLC.2010.5581054.

[111] Hongpo Zhang, Lulu Huang, Chase Q. Wu, and Zhanbo Li. An effective convolutional neural network based on smote and gaussian mixture model for intrusion detection in imbalanced dataset. Computer Networks, 177:107315, 2020. ISSN 1389-1286. doi: 10.1016/j.comnet.2020.107315.

[112] Lei Zhang, Shuaimin Jiang, Xiajiong Shen, Brij B. Gupta, and Zhihong Tian. PWG-IDS: an intrusion detection model for solving class imbalance in iiot networks using generative adversarial networks. CoRR, abs/2110.03445, 2021.

[113] Xiaoqing Zhang, Fei Yang, Yue Hu, Zhao Tian, Wei Liu, Yifa Li, and Wei She. Ranet: Network intrusion detection with group-gating convolutional neural network. Journal of Network and Computer Applications, 198:103266, 2022. ISSN 1084-8045. doi: https://doi.org/10.1016/j.jnca.2021.103266.

[114] Feng Zhao, Hao Zhang, Jia Peng, Xiaohong Zhuang, and Sang-Gyun Na. A semi-self-taught network intrusion detection system. Neural Computing and Applications, 32, 12 2020. doi: 10.1007/s00521-020-04914-7.