Resilient Intrusion Detection Systems Against Adversarial Attacks: Toward Robustness in Data-Driven Security



Rui Santos
Master in Network and Information Systems Engineering Department of Computer Science
Faculty of Sciences of the University of Porto
2026

Resilient Intrusion Detection Systems Against Adversarial Attacks: Toward Robustness in Data-Driven Security

Rui Santos

Dissertation carried out as part of the Master in Network and Information Systems Engineering

Department of Computer Science

2026

Supervisor
Prof. João Resende, Assistant Professor, Faculty of Sciences of the University of Porto

Co-supervisor
Prof. Pedro Brandão, Assistant Professor, Faculty of Sciences of the University of Porto

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor, Prof. João Resende, for giving me the opportunity and privilege to work on this topic. His guidance, the tools he provided, and his continuous support were fundamental in helping me achieve this final result. I also extend my thanks to my co-supervisor, Prof. Pedro Brandão, for his valuable insights and assistance throughout this process.

A special word of thanks goes to my family, especially my parents. They have been incredibly supportive throughout my entire academic journey, and I am deeply grateful for their constant belief in me.

I would also like to thank all the friends and colleagues I have crossed paths with from high school to my master's degree. A special thank you to my closest friends, who have accompanied me on this journey and served as a crucial pillar of support throughout this journey.

Last but certainly not least, I want to thank my girlfriend for always believing in me and for being a constant, reassuring presence whenever I needed her.

Thank you all ! .

Resumo

A crescente dependência de infraestruturas digitais interligadas tornou a cibersegurança uma preocupação central para as organizações modernas. Os Sistemas de Deteção de Intrusões em Rede (NIDS) baseados em Deep Learning têm demonstrado elevado potencial na deteção de ameaças sofisticadas e previamente desconhecidas, mas continuam vulneráveis a ataques adversariais de evasão. Embora mecanismos defensivos como o Treino Adversarial (Adversarial Training, AT) possam aumentar a robustez destes modelos, o seu custo operacional em cenários desequilibrados, packet-based e próximos do tempo real permanece insuficientemente quantificado.

Esta tese investiga a robustez de NIDS packet-based baseados em Deep Learning perante manipulações adversariais compatíveis com os protocolos de rede. Para ultrapassar as limitações de latência associadas à extração convencional de características baseada em fluxos, é proposta uma metodologia de representação packet-to-image orientada por transições. Esta abordagem agrega pacotes em sessões utilizando identificadores (5-tuplo) de fluxo bidirecionais e transforma sequências de pacotes em representações visuais bidimensionais. Ao preservar o contexto temporal e a direcionalidade do tráfego, a metodologia proposta permite uma deteção de intrusões mais precoce e contextualmente mais rica.

Com base nesta representação, são avaliados modelos supervisionados, nomeadamente Redes Neuronais Convolucionais (CNN), e modelos não supervisionados, nomeadamente Autoencoders, sob perturbações adversariais compatíveis com os protocolos, incluindo alterações ao Time-to-Live (TTL), modificações temporais e preenchimento da carga útil dos pacotes. Os resultados mostram que os Autoencoders são particularmente vulneráveis a alterações adversariais localizadas, uma vez que os erros globais de reconstrução tendem a diluir pequenas perturbações relevantes do ponto de vista da segurança. Em contraste, as CNNs combinadas com Treino Adversarial demonstram uma melhoria substancial da robustez, reduzindo as taxas de evasão de 30,5% para menos de 3,1% e diminuindo drasticamente o número de ataques não detetados.

No entanto, esta robustez implica um custo operacional significativo. O Treino Adversarial altera de forma agressiva as fronteiras de decisão do modelo, reduzindo a Precisão Macro de valores superiores a 80% para valores entre 48% e 57%, o que resulta num aumento substancial de falsos positivos e de ameaças incorretamente classificadas. A análise da latência de inferência em ambientes CPU e GPU mostra que o mecanismo

de defesa introduz uma sobrecarga reduzida no tempo de execução; contudo, a manutenção de um débito próximo do tempo real exige aceleração por GPU, uma vez que a execução apenas em CPU provoca estrangulamentos significativos. Em última análise, estes resultados evidenciam o compromisso crítico enfrentado pelos Centros de Operações de Segurança (SOCs): aceitar um maior risco adversarial ou aumentar a carga dos analistas com volumes potencialmente insustentáveis de alertas de menor fiabilidade.

Palavras-chave: Aprendizagem Automática Adversarial, Treino Adversarial, Deep Learning, Deteção Baseada em Pacotes, Representação de Tráfego Baseada em Imagem, Segurança em Tempo Real, Custo Operacional, Compromisso entre Robustez e Exatidão

Abstract

The growing reliance on interconnected digital infrastructures has made cybersecurity a critical concern for modern organizations. Deep Learning-based Network Intrusion Detection Systems (NIDS) have shown strong potential for detecting sophisticated and previously unseen threats, yet they remain vulnerable to adversarial evasion attacks. Although defensive techniques such as Adversarial Training (AT) can improve robustness, their operational cost in imbalanced, packet-level, near real-time detection scenarios remains insufficiently understood.

This thesis investigates the robustness of packet-based Deep Learning NIDS under protocol-compliant adversarial manipulation. To address the latency limitations of conventional flow-based feature extraction, it proposes a transition-aware packet-to-image representation that aggregates packets into sessions using bidirectional 5-tuple flow identifiers and transforms packet sequences into two-dimensional visual representations. By preserving temporal context and traffic directionality, the proposed methodology supports earlier and more context-rich intrusion detection.

Using this representation, the work evaluates supervised Convolutional Neural Networks (CNNs) and unsupervised Autoencoders under adversarial perturbations, including Time-to-Live (TTL) modifications, timing changes, and payload padding. The results show that Autoencoders are particularly vulnerable to localized adversarial changes, as global reconstruction errors dilute small but security-relevant perturbations. In contrast, CNNs combined with Adversarial Training substantially improve resilience, reducing evasion rates from 30.5% to below 3.1% and nearly eliminating undetected attacks.

However, this robustness comes at a significant operational cost. Adversarial Training shifts the model's decision boundaries aggressively, reducing Macro Precision from above 80% to between 48% and 57%, thereby increasing false positives and misclassified threats. Latency analysis further shows that the defense mechanism itself introduces limited inference overhead, but near real-time throughput requires GPU acceleration, as CPU-only execution produces significant bottlenecks. Overall, the findings highlight a critical trade-off for Security Operations Centers: accepting higher adversarial risk or increasing analyst workload through larger volumes of lower-fidelity alerts.

Keywords: Adversarial Machine Learning, Adversarial Training, Deep Learning, Packet-Based Detection, Image-Based Traffic Representation, Real-Time Security, Operational

Cost, Robustness-Accuracy Trade-off.

Table of Contents

List of Tables viii
List of Figures ix

Acronyms and Abbreviations 1
1. Introduction 3
    1.1. Motivation 3
    1.2. Objectives 4
    1.3. Contributions 4
    1.4. Outline 5
2. Background 6
    2.1. IDS background 6
    2.2. Machine Learning 8
    2.3. Deep Learning 11
        2.3.1. Artificial Neuron 11
        2.3.2. Neural Network Architecture 12
        2.3.3. Training a Neural Network 13
        2.3.4. From Shallow Networks to Deep Networks 15
    2.4. ML & DL role in anomaly based NIDS 15
    2.5. Adversarial Threats 17
        2.5.1. Adversary knowledge 18
        2.5.2. Attack Strategy 20
        2.5.3. Adversarial goal 25
        2.5.4. Attack frequency 25
        2.5.5. Adversarial Defenses 25
3. Related work 29
4. Datasets for NIDS 35
    4.1. UNSW-NB15 35
    4.2. CIC-IDS2017 35
    4.3. CIC-DDoS2019 36
    4.4. LITNET-2020 37
5. Metrics 38
    5.1. Binary Classification metrics 38
    5.2. Multiclass Classification metrics 39
        5.2.1. Per-Class Components 39
        5.2.2. Averaging Techniques 40

6. Flow based vs Packet based 41
7. Proposed Methodology 43
7.1. Creating the labeled Image dataset 44
7.2. Testing DL models 48
7.3. Adversarial Attacks + Defenses 54
7.3.1. Adversary assumptions 55
8. Discussion & Results 57
8.1. Labeled Image dataset 57
8.2. Testing DL models 58
8.2.1. CNN 58
8.2.2. AE 63
8.3. Adversarial Robustness (before and after AT) 66
8.3.1. Robustness before AT 67
8.3.2. Robustness after AT 68
8.3.3. Impact of AT on clean data 69
8.4. Overall Discussion 75
8.4.1. Detection Performance Insights 75
8.4.2. Adversarial Robustness Analysis 76
8.4.3. Trade-off: Robustness vs Precision 76
8.4.4. Operational Impact (SOCs) 76
8.5. Operational Overhead Analysis 77
9. Conclusion 79
9.1. Limitations 80
9.2. Future Work 80

Bibliography 81

List of Tables

2.2. Comparison of Supervised and Unsupervised learning methods - Advantages and Disadvantages (based on [7] and [17]) 11
2.4. Comparison of ML, DL, and Ensemble techniques: Strengths, Limitations, and Practical Implications for NIDS(Inspired by [2]) 17
2.5. Evasion attack methods and their classifications based on adversary knowledge (adapted from [61]) 25

3.1. Recent advancements in ML for anomaly-based NIDS. 33
3.2. Comparison of recent contributions in anomaly-based NIDS 34

4.1. UNSW-NB15 distribution of Attack Types [89] 35
4.2. CIC-IDS2017 distribution of Attack Types [90] 36
4.3. CIC-DDoS2019 distribution of Attack Types [91] 37
4.4. LITNET-2020 distribution of Attack Types [92] 37

7.2. Comparison of my image builder and the image builder from [3] 47
7.3. CNN Architecture, Fixed Parameters, and Hyperparameter Search Space 51
7.5. AE Architecture, Fixed Parameters, and Hyperparameter Search Space 53
7.8. Summary of the perturbations used to create adversarial samples from malicious images 54

8.1. Distribution of attack classes of the different versions of the image dataset. 58
8.2. CNN hyperparameter configurations for each trial across multiple dataset versions. 60
8.3. Detailed classification metrics for each class and overall performance from the best CNN models across the different dataset versions. 62
8.4. AE hyperparameter configurations for each trial across multiple dataset versions. 65
8.5. Detailed binary classification metrics and overall performance from the best AE models across different dataset versions. 66
8.6. Detailed classification metrics for each class, as well as overall performance, of the best CNN models with adversarial training across the different dataset versions. 73
8.7. Impact of adversarial training on CPU and GPU inference latency: Mean $\pm$ sample standard deviation across varying dataset versions. 78

List of Figures

2.1. IDS Detection Methods Comparison....7
2.2. IDS Deployment Methods Comparison....8
2.3. Taxonomy of Intrusion Detection Systems (IDS)....8
2.4. Machine Learning Pipeline....10
2.5. Machine Learning Methods....11
2.6. Architecture of an Artificial Neuron....12
2.7. Example of the Architecture of a Neural Network....13
2.8. Visual representation of the Loss Function $J(\hat{y}, y)$, showing the comparison between the model's prediction and the ground truth label.....14
2.9. Backpropagation process in a Neural Network....15
2.10. Shallow vs Deep Neural Networks....15
2.11. Machine Learning and Deep learning algorithms typically used in NIDS....17
2.12. Full process of Adversarial Machine learning [34]....18
2.13. Adversarial Knowledge....20
2.14. Illustration of Evasion and Poison attacks,based on [37]....22
2.15. Illustration of model inversion attacks....23
2.16. Types of attack strategies (adapted from [2])....24

6.1. Flow and Packet based features....42

7.1. Full - Proposed Methodology....44
7.2. Stage 1 - Labeled Image dataset creation pipeline....44
7.3. Feature vector obtained after using Payload-Byte[101]....45
7.4. Feature vector used to represent a single packet in the Image dataset....48
7.5. Example of Image generator process....48
7.6. Stage 2 - Hyperparameter tuning pipeline....48
7.7. Stage 3 - Pipeline for adversarial attack and defense testing....54

8.1. Training and validation performance metrics over 10 epochs for the CNN model hyperparameter tuning. Columns from left to right display Categorical Accuracy, F1 Macro score, Categorical Crossentropy Loss, and PR-AUC. Rows correspond to successive dataset versions, beginning with $N = 3$ and ending with $N = 8$.....61
8.2. Training and validation performance metrics over 15 epochs for the AE model hyperparameter tuning. Columns from left to right display Binary crossentropy loss, MAE, and MSE. Rows correspond to successive dataset versions, beginning with $N = 3$ and ending with $N = 8$.....65
8.3. Evasion rate of adversarial attacks against the candidate CNN models across different dataset versions.....67
8.4. Evasion rate of adversarial attacks against the candidate AE models across different dataset versions.....68
8.5. Evasion rate of adversarial attacks against the candidate CNN models (with AT) across different dataset versions.....69
8.6. CNN performance metrics (Missed Threats, Misidentified Threats, and False Alarms) without AT.....74
8.7. CNN performance metrics (Missed Threats, Misidentified Threats, and False Alarms) with AT.....74

8.8. Impact of AT on Macro and Weighted performance metrics across CNN architectures.74

List of Algorithms

1. Algorithm for the Image Building component 47

Acronyms and Abbreviations

A2PM	Adaptative Perturbation Pattern Method
AD	Adversarial Denoising
AE	Auto Encoder
AI	Artificial Intelligence
AML	Adversarial Machine learning
APE_GAN++	Adversarial Perturbation Elimination with Generative Adversarial Networks
AT	Adversarial Training
BFAM	Brute Force Attack Method
BIM	Basic Iterative Method
BMI-FGSM	Black-box Momentum Iterative Fast Gradient Sign Method
C&W	Carlini & Wagner
CNN	Convolutional Neural Network
DAE	Denoising AutoEncoder
DL	Deep Learning
DM	Defense Models
DNN	Deep Neural Network
DT	Decision Tree
EDL	Ensemble Deep Learning
EMP-GANs	Ensemble Multi-Player - Generative Adversarial Networks
FSGM	Fast Gradient Sign Method
GAN	Generative Adversarial Network
GB	Gradient Boosting
GDA	Gaussian Data Augmentation
GSA-GAN	Gray-box Shilling Attack - Generative Adversarial Networks
HC	High Confidence
HIDS	Host-based Intrusion Detection Systems
IDS	Intrusion Detection Systems
JSMA	Jacobian-based Saliency Maps Attacks
KNN	K-Nearest Neighbors
LDA	Linear Discriminant Analysis
LID	Local Intrinsic Dimensionality
LOF	Local Outlier Factor
LR	Logistic Regression
LS	Label Spread
LSTM	Long Short-Term Memory

MAE	Mean Absolute Error
ML	Machine Learning
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
MTD-AD	Moving Target Defense - Adversarial Defense
NIDS	Network-based Intrusion Detection Systems
PGD	Projected Gradient Descent
PR-AUC	Precision-Recall Area Under the Curve
READ	Reconstruction Error-based Adversarial Detection
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
SM	Substitute Models
SMOTE	Synthetic Minority Over-sampling Technique
SOM	Self-Organising Map
SVM	Support Vector Machine
TM	Target Models
WGAN	Wasserstein Generative Adversarial Networks
ZOO	Zeroth Order Optimization

1. Introduction

The growing dependence on interconnected digital infrastructures has made cybersecurity a critical concern for modern organizations. As network environments continue to increase in scale, complexity, and traffic volume, protecting systems against sophisticated cyber threats has become progressively more challenging. Attackers constantly develop new techniques capable of bypassing traditional security mechanisms, creating the need for more adaptive and intelligent intrusion detection solutions.

Intrusion Detection Systems (IDS) play a fundamental role in modern cybersecurity by monitoring network or host activity to identify malicious behavior and security violations. Among the different IDS paradigms, Network Intrusion Detection Systems (NIDS) are particularly relevant because they operate at the network level, allowing the detection of attacks targeting multiple devices and services simultaneously.

Traditionally, NIDS relied on signature-based detection methods capable of identifying previously known attack patterns with high precision. However, these approaches struggle to detect zero-day attacks and rapidly evolving malicious behaviors, since they depend heavily on continuously updated attack signatures. To address these limitations, anomaly-based detection approaches powered by Machine Learning (ML) and Deep Learning (DL) have gained significant attention in recent years. By learning patterns associated with normal network behavior, these systems can generalize to previously unseen attacks and identify more complex malicious activities [1].

This thesis investigates the robustness of packet-based Deep Learning NIDS against adversarial manipulation. More specifically, this work explores transition-based packet-to-image representations for network traffic classification, evaluates the vulnerability of Deep Learning models under adversarial conditions, and assesses the effectiveness and operational impact of Adversarial Training as a defense mechanism.

1.1. Motivation

The integration of ML and DL into cybersecurity, while offering significant advancements, also introduces a new attack surface known as adversarial attacks. Through carefully crafted perturbations, attackers can manipulate network traffic in ways that deceive ML-based detection systems while preserving protocol validity and malicious functionality. As

a result, adversarial attacks can significantly reduce detection accuracy and compromise the reliability of data-driven NIDS in operational environments [2].

At the same time, existing research mainly focuses on flow-based traffic representations and conventional feature engineering approaches, often overlooking packet-based representations capable of preserving sequential and directional traffic information [3], [4] . Furthermore, many adversarial defense mechanisms introduce substantial computational overhead, limiting their practical applicability in real-time detection environments [5].

This thesis is motivated by the need to better understand how adversarial attacks affect packet-based DL NIDS and whether robustness mechanisms such as Adversarial Training (AT) can provide meaningful security improvements without imposing impractical operational costs. By exploring packet-based traffic representations, adversarial robustness, and the trade-offs between security and efficiency, this work aims to contribute toward the development of more resilient and practical data-driven intrusion detection systems.

1.2. Objectives

The primary aim of this research is to evaluate and improve the robustness of anomaly-based NIDS against adversarial manipulation in packet-based DL environments. To achieve this goal, the following objectives were defined:

1. Study adversarial threats against ML and DL-based NIDS, with a particular focus on evasion attacks and their impact on packet-based intrusion detection systems.
2. Develop a packet-based image representation methodology capable of preserving temporal and directional relationships between network packets for Deep Learning-based traffic classification.
3. Evaluate the effectiveness of DL models for packet-based intrusion detection under both normal and adversarial conditions.
4. Assess the effectiveness and operational impact of AT as a defense mechanism for improving the robustness of packet-based NIDS.

1.3. Contributions

During this work, a publication was submitted to the MASCOTS 2026. At the time of writing this dissertation, we are waiting for the submission result.

1.4. Outline

To systematically address the defined research objectives, this document is organized as follows. Chapter 2 provides the theoretical foundation by examining IDS, principles of ML, the landscape of adversarial threats, and the role of ML & DL in anomaly based NIDS. Chapter 3 reviews recent advancements in ML for anomaly-based NIDS to situate this research within the existing literature. The experimental baseline is established across Chapters 4, 5, and 6, which explore key network security datasets, present metrics commonly used for evaluating ML models and their impact on IDS, and explain the choice of packet-based over flow-based feature extraction. Chapter 7 describes the core methodology, including the creation of a labeled image dataset using Payload-Byte, the tuning of model hyperparameters, and the setup of the adversarial testing environment. Chapter 8 analyzes the experimental results, focusing on baseline detection performance, model vulnerabilities, the trade-offs involved in adversarial training, and an operational overhead analysis. Finally, Chapter 9 summarizes the key findings, acknowledges the limitations of this work, and recommends directions for future research.

2. Background

Establishing a solid theoretical foundation is essential for thoroughly understanding the vulnerabilities of modern IDS and their defense strategies. This chapter offers the necessary background, covering network security, AI, and AML, arranged to progress from basic concepts to complex systemic threats. It starts in Section 2.1 by summarizing the core architectures and traditional paradigms of IDS. Next, Sections 2.2 and 2.3 present the fundamental principles of ML and DL, focusing on the transition to data-driven pattern recognition. These computational approaches are integrated in Section 2.4, which emphasizes the crucial role of ML and Deep Learning (DL) in contemporary anomaly-based NIDS, demonstrating how NN are used to classify intricate network traffic. Finally, after explaining how these intelligent systems function, Section 2.5 examines their key vulnerabilities by outlining the main elements of adversarial threats.

2.1. IDS background

In computer security, the primary focus usually revolves around intrusion prevention, that aims to protect systems and networks from attackers. Firewalls and antivirus software serve as examples of intrusion prevention systems that act as the first line of defense against unauthorized access. However, like all cybersecurity measures, these prevention mechanisms can fail, allowing intrusions to occur. When such failures happen, IDS come into play, enabling the detection of attacks before, during, and after they happen. Before the advent of IDS, detecting signs of an attack often required manually reviewing log files and looking for unusual behavior. With IDS, this manual process can be automated, allowing for more efficient detection [6, Section 8.10].

Although the literature provides multiple definitions of IDS, they typically describe them as security mechanisms (either software or hardware) that monitor network traffic to detect malicious activity or policy violations. When such threats are identified, the system alerts network administrators, providing the visibility needed to respond to incidents and implement mitigation measures [7]. IDS detects intrusions using signature-based, anomaly-based, and hybrid techniques. Signature-based detection relies on information from previous known attacks and vulnerabilities to create signatures [1]. However, as attacks evolve, this method quickly becomes insecure and difficult to maintain since it requires constant updates for new attack variants and substantial storage for signatures.

Despite these drawbacks, it offers the highest precision and the lowest false-positive rates for known threats, but it struggles to detect unknown or emerging ones. Conversely, anomaly-based approaches define a baseline for normal behavior, and when this baseline is exceeded (indicating a potential attack) a warning is triggered [6, Section 8.10]. This distinction between normal and abnormal activity improves detection of zero-day and unfamiliar threats but also results in a higher false-positive rate [1]. Finally, the hybrid approach combines signature-based and anomaly-based detection to leverage their respective strengths and address each other's weaknesses. Figure 2.1 shows the main advantages and disadvantages of each detection method discussed.

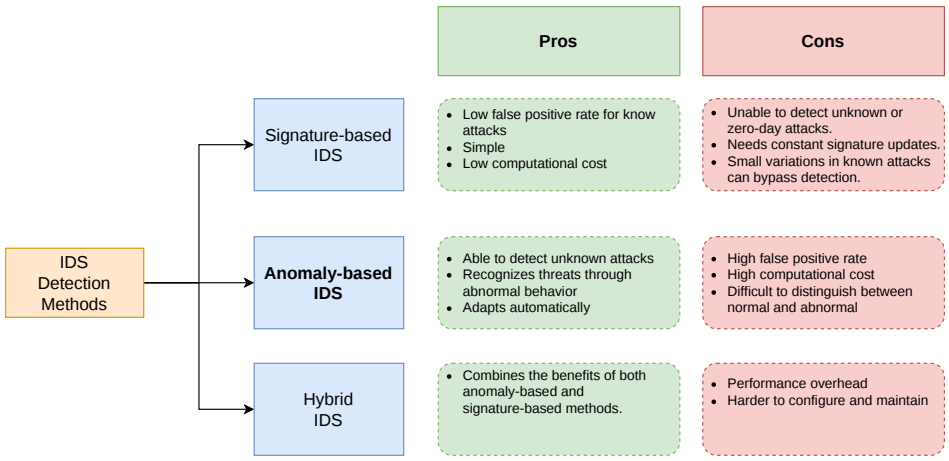


Figure 2.1: IDS Detection Methods Comparison

Deployment of IDS can be divided into two primary scenarios: Host-based IDS (HIDS) and Network-based IDS (NIDS). HIDS is installed on a single host, monitoring local activities like analyzing system logs, checking file integrity and inspecting running processes. Its primary goal is to protect the host on which it is installed. Conversely, NIDS is deployed within the network to detect intrusions that target the network itself, aiming to safeguard both the network and connected users [8]. It actively monitors network traffic for security breaches. Figure 2.2 summarizes the main advantages and disadvantages of each deployment method discussed, while Figure 2.3 presents the overall taxonomy for IDS.

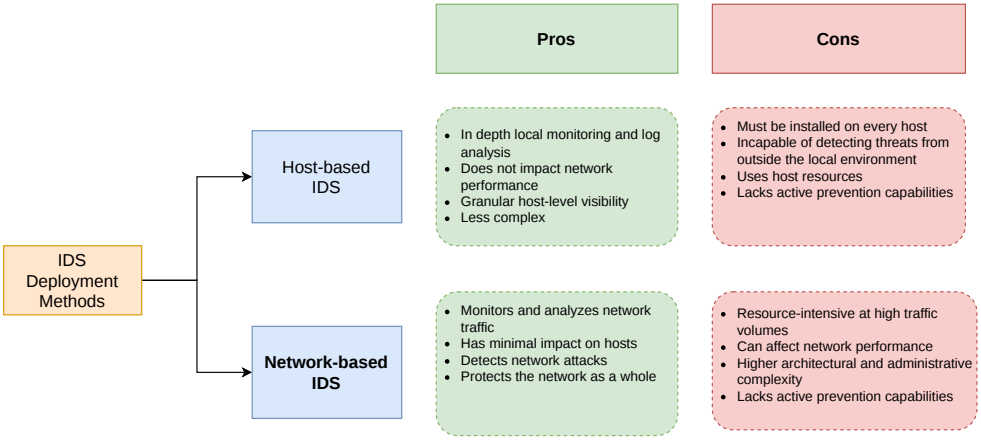


Figure 2.2: IDS Deployment Methods Comparison

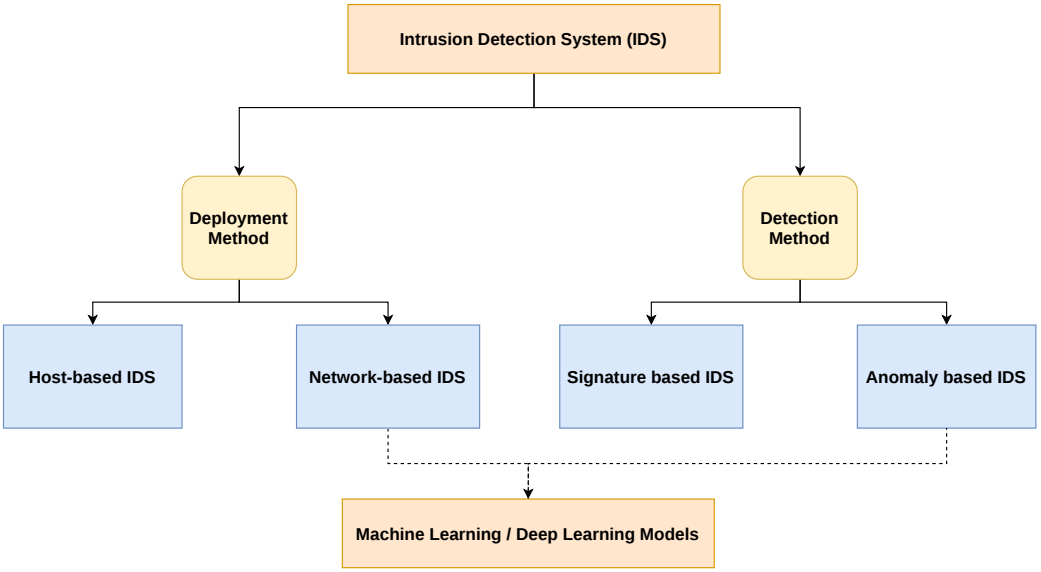


Figure 2.3: Taxonomy of Intrusion Detection Systems (IDS)

2.2. Machine Learning

ML is a branch of AI that employs algorithms to detect patterns in training data and apply these insights to new data [9]. Instead of relying solely on explicitly programmed rules, these algorithms iteratively learn to build generalized models that, when optimized, can perform complex tasks such as image classification, time-series forecasting, and anomaly detection with high accuracy [10]. A key benefit of ML is its ability to function autonomously, with trained models classifying data and predicting outcomes without human input, enabling analysis of high-dimensional datasets that would otherwise require extensive time and effort from human experts to process manually [11].

As illustrated in Figure 2.4, the typical ML pipeline used to develop a model consists of six main steps, each briefly defined below:

1. **Data Collection:** Collect relevant data from multiple sources (APIs, CSV files, etc.) for the task at hand, noting that the model's effectiveness depends strongly on the quality and diversity of the data gathered [12].

2. **Preprocessing:** Transform raw, unstructured data into a clean, standardized format suitable for analysis. This includes cleaning data to resolve missing values, duplicates, and noise; normalizing and scaling numerical features; and encoding categorical variables to ensure compatibility with ML models. Proper preprocessing maintains data integrity and ensures unbiased model learning [13].

3. **Feature Engineering:** Select important variables and generate new features to improve model performance.

4. **Model Training:** Train the ML model on the prepared dataset so it can adjust its internal parameters and learn underlying patterns. This training process enables the model to make predictions or classify new, unseen data based on what it has learned.

5. **Evaluation:** Evaluate the trained models performance using metrics like accuracy, precision, recall, and others.

    - If the model does not meet the performance benchmarks during this phase, the preprocessing step can be revisited and adjusted to improve results in subsequent evaluations.

6. **Deployment:** Deploy the model with satisfactory performance into a real-world environment, making it available for practical use and integration into production systems.

    - Since models inevitably experience concept drift, which occurs when their predictive accuracy declines as new data patterns appear, the ML process counteracts this by retraining the model [14]. Essentially, this means restarting the pipeline steps repeatedly to enable ongoing improvement and adaptation over time.

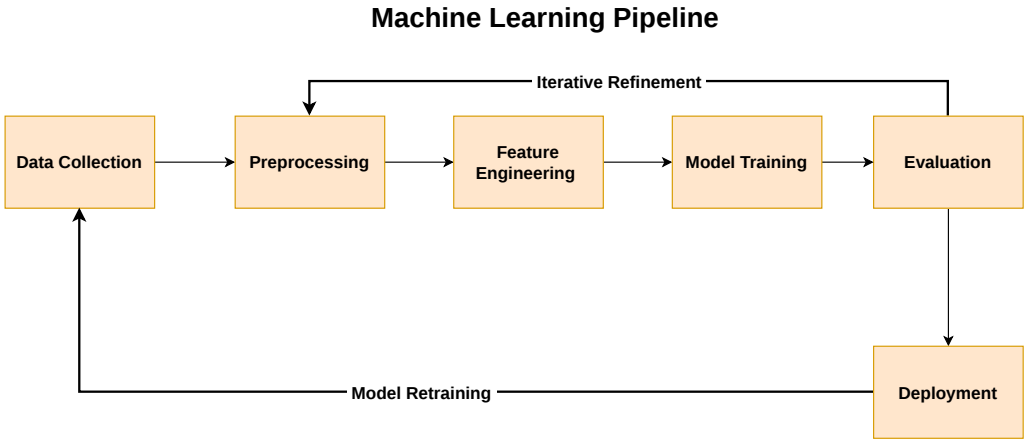


Figure 2.4: Machine Learning Pipeline

Despite the existence of various types of ML methods, this thesis primarily focuses on supervised and unsupervised learning. In supervised learning, the goal is to define a model that predicts an output based on a set of features that includes the target feature (also known as ground truth). In practical terms, a labeled dataset is required, which can then be used to develop a model for either classification or regression, depending on whether the target feature to be predicted is categorical or numerical, respectively [7]. This requirement for labeled datasets is a key disadvantage of supervised learning, as it requires experts to manually label each data point accurately, which is costly and time-consuming process [15]. Unsupervised learning differs from supervised learning because it lacks access to ground truth. The model improves automatically without user supervision and attempts to discover patterns and information from the provided data [7]. Unsupervised learning can be categorized into three main problem types clustering, association, and dimensionality reduction [7]. As defined in [16], clustering can be informally described as the process of dividing a set of data points into clusters of highly similar data points. In association rules, a set of transactions is provided, and rules are defined to predict the occurrence of an item based on the presence of other items in the transaction [16]. As the name suggests, dimensionality reduction is a technique used to decrease the number of features in a dataset, simplifying the handling and processing of the dataset [7]. Figure 2.5 provides an overview of these methodologies. Table 2.2 compares the key advantages and disadvantages of each learning method.

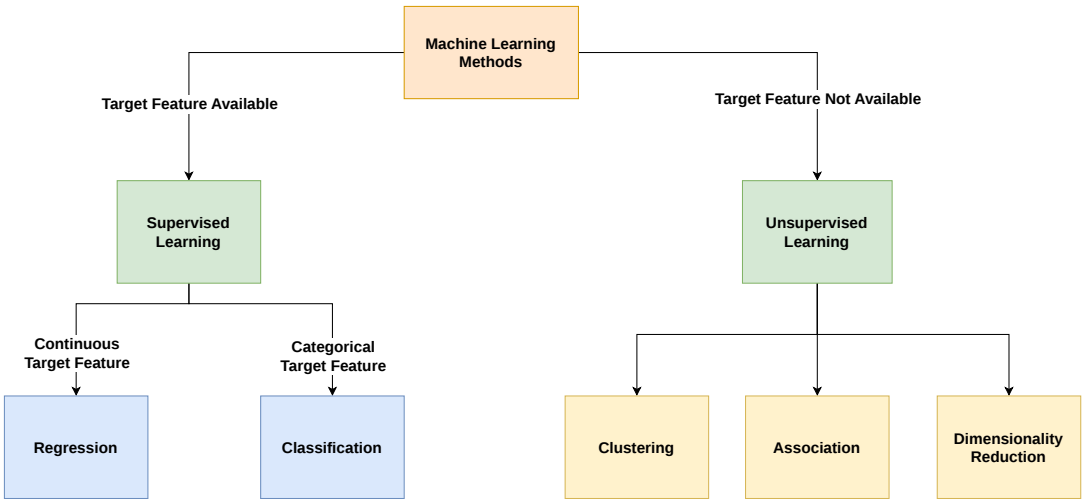


Figure 2.5: Machine Learning Methods

ML Method	Advantages	Disadvantages
Supervised	Works well with larger datasets; Can produce highly accurate models; Uses labeled data; Works well in predictive environments.	Takes longer time to compute; Relies heavily on labeled datasets; High risk of overfitting with noisy or complex data.
Unsupervised	Works well with smaller datasets; Excels at finding hidden patterns on raw data; Uses unlabeled data; Works well in analytical environments.	Can be computationally intensive with larger datasets; Can lead to suboptimal or unexpected results.

Table 2.2: Comparison of Supervised and Unsupervised learning methods - Advantages and Disadvantages (based on [7] and [17])

2.3. Deep Learning

While traditional ML is effective for basic pattern recognition, the growing complexity of network traffic calls for more advanced methods. DL tackles these issues by replicating the hierarchical organization of the human brain through multiple layers of interconnected nodes [18]. This section examines DL's architecture, from the basic artificial neuron to complex multi-layer networks that can detect subtle, non-linear markers of contemporary cyber threats.

2.3.1 Artificial Neuron

The basic unit of a NN consist of the artificial neuron [19]. As shown in Figure 2.6, a single neuron in an NN receives a vector of inputs ($\vec{x}$) multiplied by a vector of weights ($\vec{w}$), summed together along with a bias term (b). A larger weight w_i indicates a greater

influence of the corresponding input x_i on the neuron, increasing the likelihood of its activation [19]. To determine whether the neuron activates, the product $\vec{w}_i \cdot \vec{x}_i$ plus the bias b is passed through an activation function, which triggers the neuron if the activation threshold is reached [20].

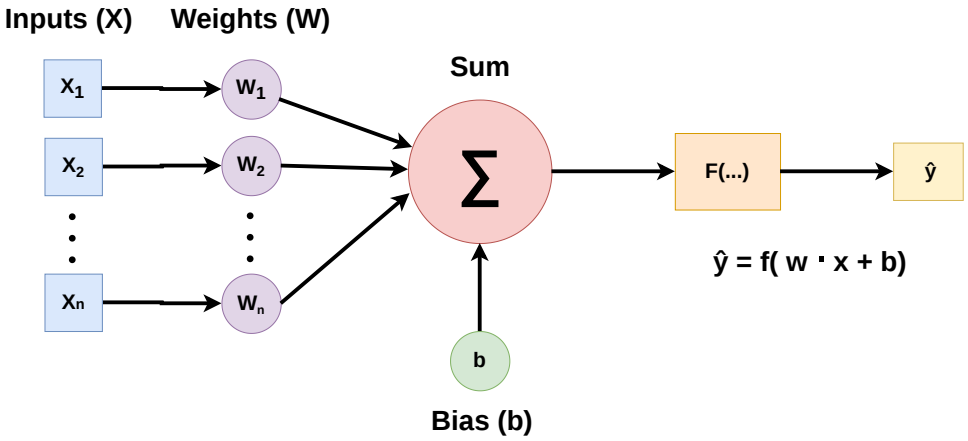


Figure 2.6: Architecture of an Artificial Neuron

2.3.2 Neural Network Architecture

A NN can be constructed using various neurons, with its architecture divided into three main layers, as depicted in Figure 2.7. The input layer contains special neurons that do not perform any computation but simply store features from the input data, with each neuron corresponding to a specific feature. The hidden layers handle the primary computations, often comprising multiple layers of neurons that transform the input data into a form that the output layer can interpret. The final, or output, layer generates the network's predictions, and the number of neurons it contains depends on the task. For example, binary classification uses two neurons, whereas multiclass classification uses one neuron per class [21].

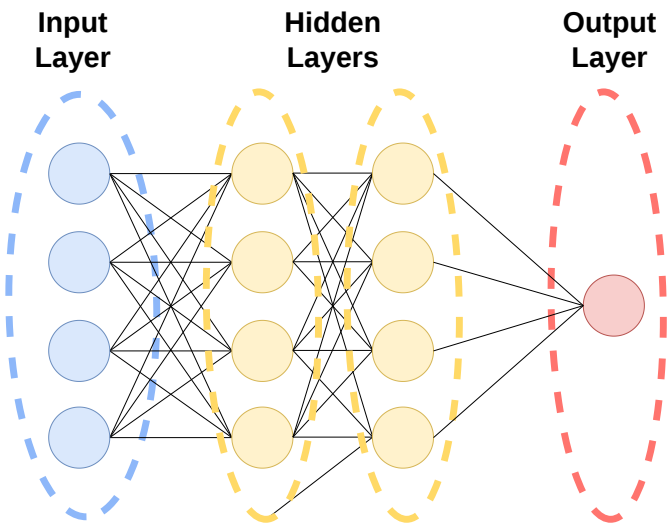


Figure 2.7: Example of the Architecture of a Neural Network

2.3.3 Training a Neural Network

To train a single neuron, a dataset containing inputs ($\vec{x}$) and outputs (y) is necessary. The activation function should be chosen based on whether the output is continuous or binary, for example a linear function for continuous outputs or a step function for binary outcomes. Additionally, a loss function must also be defined to quantify how much the current parameters, $\vec{w}$ and b, deviate from accurately representing the data (see Figure 2.8 for a visual of the loss function in NN). Training a NN primarily involves adjusting parameters to minimize loss, typically achieved using gradient descent, which updates parameters and reduces loss by following these three steps:

1. **Measure prediction error**: The network evaluates how closely its prediction matches the correct answer, effectively measuring the error through a loss function.
2. **Calculate the direction**: Where the ”gradient” indicates how the weights and bias should be updated in order to reduce error.
3. **Update parameters**: Where the network adjust the weights (w_i) and bias b for each neuron, in the direction and repeats this process until the final error is minimal.

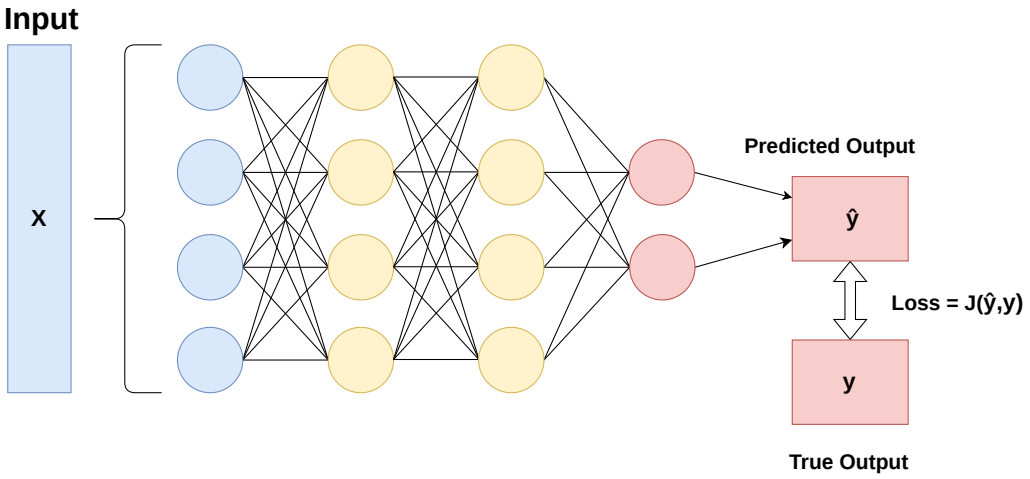


Figure 2.8: Visual representation of the Loss Function $J(\hat{y}, y)$, showing the comparison between the model's prediction and the ground truth label.

The algorithm that enables a NN to learn from its mistakes is known as backpropagation [22]. In this process, information is passed backward from the output layer to the input layer, allowing each connection to adjust its parameters based on its contribution to the total loss [23] (see Figure 2.9 for the steps involved in backpropagation in a neural network). The steps for backpropagation can be summarized as:

1. **Forward Pass**: The data is fed into the NN, which makes a prediction. During this step, the activation function values for each neuron are stored [24].
2. **Backward Pass**: The final error is computed and propagated backward through all the layers. Using the derivatives calculated, the NN determines the exact contribution of each neuron to the final loss and adjusts the weights (w_i) and biases (b_i) proportionally to minimize the overall network error [22].

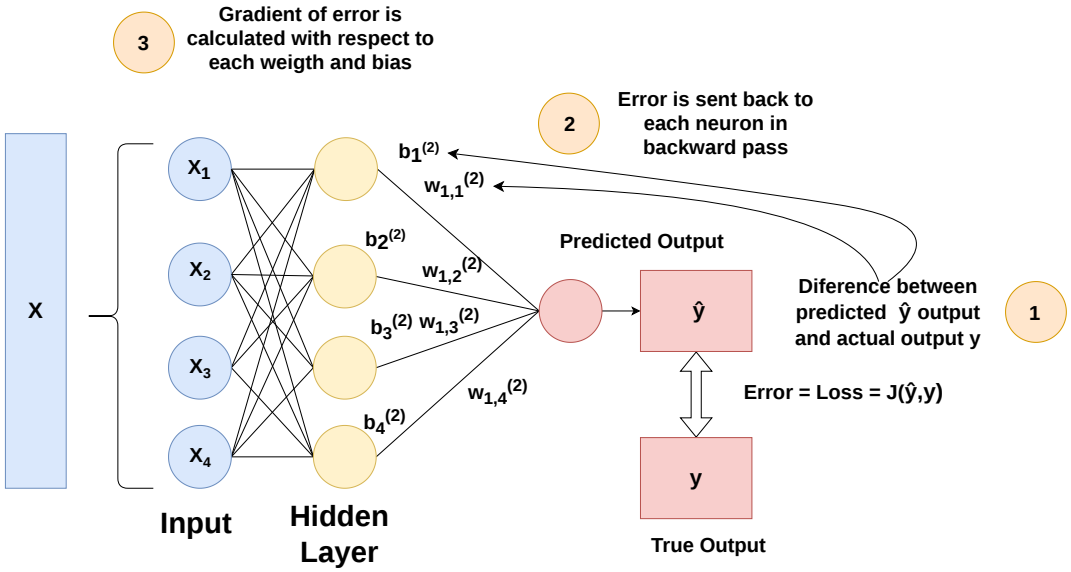


Figure 2.9: Backpropagation process in a Neural Network

2.3.4 From Shallow Networks to Deep Networks

The difference between shallow networks (also known as NN) and DNN is that shallow networks have only a single hidden layer, whereas DNNs contain two or more hidden layers (as depicte by Figure 2.10). These additional hidden layers enable DNNs to learn more abstract and complex features from the data [25].

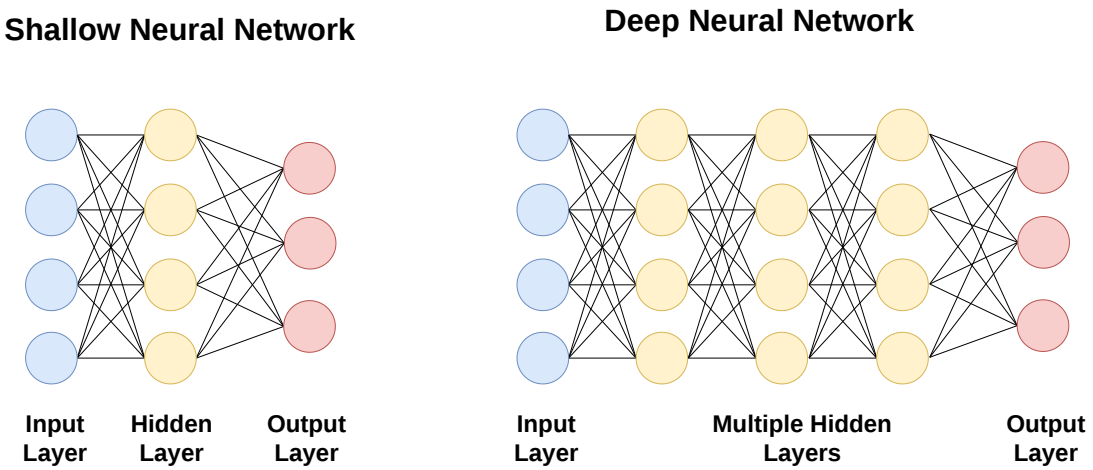


Figure 2.10: Shallow vs Deep Neural Networks

2.4. ML & DL role in anomaly based NIDS

Over the years, various ML techniques have been proposed and proven effective anomaly based NIDS [26]. This trend is justified by the ability of ML algorithms to identify complex relationships between various features and to effectively generalize to new, unseen data

[2]. In NIDS, the primary modeling techniques are divided into ML (also known as shallow models) and DL (a subset of ML) [27]. ML-based NIDS relies on manual feature engineering to learn valuable insights from network traffic [28]. In contrast, DL-based NIDS learns without the need for explicit feature engineering and excels at automatically learning complex features directly from raw data due to its deep structure [29]. Several research papers, including [7], [27], [30], and others suggest that DL is the leading trend in anomaly-based NIDS because of its ability to learn from large raw datasets and produce highly accurate results models. Despite their popularity, DL algorithms are more resource-intensive and need adequate hardware to provide real-time results compared to traditional ML methods. It is also worth noting that, although DL is the current trend, ML algorithms can often achieve comparable or even better performance on specific tasks, so choosing the appropriate model depends on testing which solution meets the required performance levels within the available computational resources. Ensemble methods are a third modeling approach that integrates multiple algorithms from ML and DL, or both. By combining the strengths of different classifiers through merging several ”weak classifiers” and applying a voting algorithm to form a more robust and ”stronger classifier” [27]. Table 2.4 offers a comparative review of the strengths, limitations, and practical applications for NIDS of ML, DL, and ensemble-based methods, while Figure 2.11 illustrates the most commonly used ML and DL algorithms for anomaly-based NIDS.

Although significant advancements have been made in applying ML algorithms for anomaly-based NIDS, several research gaps and limitations remain that need to be addressed [31]. Firstly, there is a lack of labeled datasets specifically for training and evaluating IDS models. Existing publicly available datasets have issues such as limited diversity, class imbalance, and outdated attack types. This limitation is particularly problematic because it impacts the most critical aspect of ML, namely data quality, and consequently hampers the ability of IDS to identify novel and sophisticated attacks [31]. Secondly, there is a lack of transparency and interpretability in ML-based IDS models, especially in DL algorithms that function as black boxes, thereby making it difficult to understand the model’s classifications. The absence of transparency reduces trust and hampers the adoption of these models, particularly in critical network security environments where explaining detected threats is essential [31]. Finally, recent research indicates that IDS models based on ML are highly susceptible to adversarial attacks, which are explained in more detail on Section 2.5.

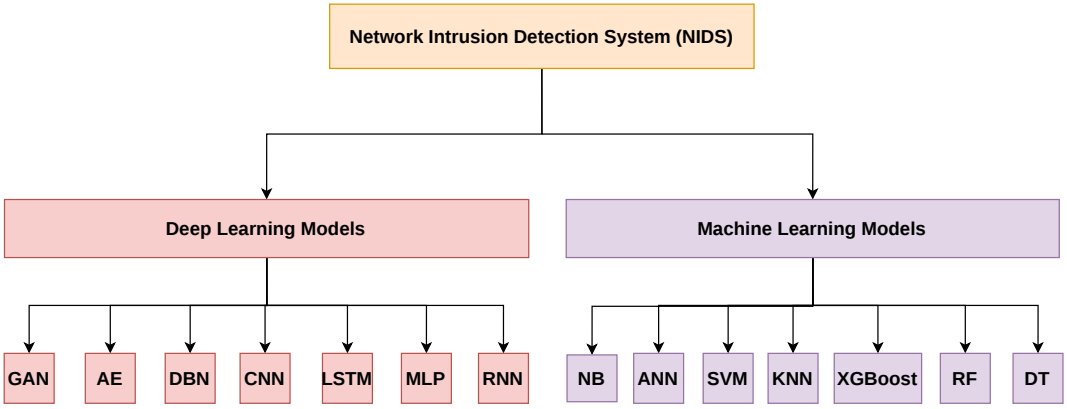


Figure 2.11: Machine Learning and Deep learning algorithms typically used in NIDS

Learning Technique	Strengths	Limitations	Practical implications for NIDS
ML	Simpler models; Requires less computation resources; Easy to interpret; Works well with smaller datasets;	Manual feature extraction; Struggles with high-dimensional data;	Suitable for low-resource environments but insufficient against evolving threats.
DL	Automatic feature extraction; Capable of managing highly-dimensional data;	Computationally demanding; Difficult to interpret; Performs better with larger datasets;	Effective against modern attacks but challenging to deploy in resource-constrained environments.
Ensemble	Improved accuracy; More robust; More generalizable;	Greater complexity and higher computational costs; Difficult to interpret; Synchronization challenges between components;	Promising for large-scale networks but requires careful balancing of scalability and efficiency.

Table 2.4: Comparison of ML, DL, and Ensemble techniques: Strengths, Limitations, and Practical Implications for NIDS(Inspired by [2])

2.5. Adversarial Threats

Adversarial threats involve adding a tiny perturbation to the clean input, aiming to cause ML models to make incorrect predictions [2]. In anomaly based NIDS, traditional ML and DL techniques are vulnerable to this type of attack, as demonstrated in studies like [32] and [33]. These studies show that adversarial attacks can achieve high success rates, sometimes exceeding 95%, with minimal perturbation. Additionally, the models' accuracy significantly drops under adversarial conditions, while evasion rates increase. Adversarial threats have long been studied in image classification problems with unstructured data, but recently, they have also gathered increased attention in anomaly-based ML NIDS

classification tasks that utilize structured data like network packets. The key difference between adversarial attacks on unstructured and structured data is that perturbations are limited in valid structured data, whereas they can be unlimited in unstructured data. For example, a network packet adheres to specific protocol rules, and we cannot arbitrarily modify its contents without ensuring the perturbation complies with these rules to maintain a valid adversarial example [2].

In network security, AML often resembles an arms race between two sides: an adversary aiming to breach the network with malicious payloads, and a defender working to protect the network from these attacks [34]. In simple terms, AML involves adversaries trying to deceive a ML model into making wrong decisions. This can be done by modifying the input data either during training (poisoning attack) [35] or during inference (evasion attack) [36]. Figure 2.12 illustrates the overall AML process and how adversarial samples are developed by generating data points x' from a dataset with features x and label y such that a classifier c predicts an incorrect label for x' while x' remains nearly indistinguishable from the original data point x, using various adversarial attack methods that solve an optimization problem to find the minimal perturbation that maximizes the model's loss [34].



Figure 2.12: Full process of Adversarial Machine learning [34]

The taxonomy of adversarial attacks in NIDS can be categorized based on four criteria: adversary knowledge, attack strategy, adversary goal, and adversary frequency [2]. A more comprehensive explanation of each criterion is available in the rest of this subsection.

2.5.1 Adversary knowledge

The adversary's knowledge encompasses the information the adversary has about the ML-based NIDS system. As displayed in Figure 2.13, this knowledge can be categorized as **White-box**, **Gray-box** or **Black-box** attacks [34]. The attack methods linked to these knowledge levels are described as follows:

- **White-box attacks:** In this scenario, the adversary is assumed to have complete knowledge of the training data, the learning algorithm, the trained model, and the respective parameters used during training [34]. These attacks represent the worst-case scenario, where the attacker leverages knowledge of the model to create more effective adversarial samples tailored to the system's specific features, often used to assess the model's robustness [2]. In terms of real-world feasibility, this kind of attack is the least feasible [34].

- **Black-box attacks:** In this situation, the attacker does not have access to the model's internal architecture, weights, or training data. They can only interact with it as if it were an oracle, by submitting specific inputs and observing the outputs [2]. The severity of black-box attacks presents a greater threat in practice because it indicates that an adversary with no information about the model's internal parameters has 'broken' the model [34]. The model's query capacity as an oracle in real-world systems is more restricted than in the theoretical black-box model, where the adversary can query the model unlimited times and has access to its full output [34]. One way to overcome the previous limitations is to create a surrogate model of the detection system that allows unrestricted querying and access to the full output, leveraging the transferability of ML models to generate adversarial samples that deceive both the substitute classifier and the actual detector [37], [38]. In terms of real-world feasibility, this attack is more practical than white-box attacks and represents a more realistic attack scenario.

- **Grey-box attacks:** This context assumes the adversary has partial knowledge of the final model and may have limited access to the training data [26]. Unlike the black box, where only inputs and outputs of the model can be observed, a gray box allows some limited understanding or control over the model's internal structure or behavior [2]. In a real-world scenario, this attack is feasible because, although the adversary lacks exact information, they have enough to attack the model and cause it to fail [26].

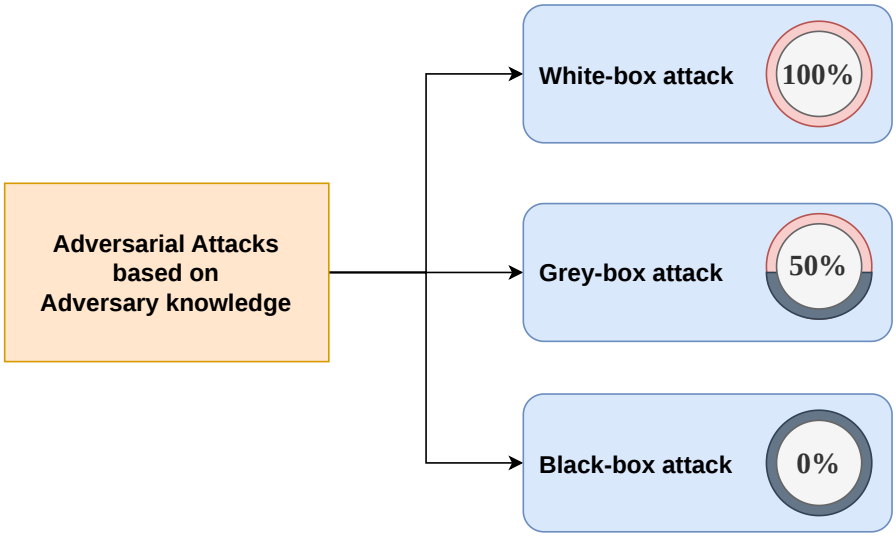


Figure 2.13: Adversarial Knowledge

2.5.2 Attack Strategy

Adversarial attacks on ML-based NIDS can happen during training (poisoning), deployment (evasion), or via model inversion:

- **Poisoning attacks:** A poisoning attack occurs when an adversary deliberately inserts malicious or manipulated data into the training set during the model's learning phase [2]. The main objective is to corrupt the training process so the ML model learns incorrect patterns, ultimately misclassifying malicious samples as benign. Such attacks can gravely undermine the integrity of the training data, often causing the resulting model to consistently misclassify inputs that resemble the tampered examples. Notable poisoning techniques include label flipping, where labels are intentionally altered to degrade dataset reliability; backdoor attacks, which embed hidden triggers that induce targeted misclassification; and clean-label attacks, where labels remain correct but features are subtly modified to bypass detection [39]. These attacks depend on adversarial access to training data, a risk heightened by the use of publicly available datasets.

- **Evasion attacks:** Evasion attacks occur when adversaries deliberately manipulate input data to avoid detection by a previously trained model [2]. These attacks are broadly categorized as gradient-based or gradient-free [34]. Gradient-based attacks leverage knowledge of the model's internal architecture, utilizing loss function gradients to identify perturbations that maximize the model's error rate and enable

evasion [40]. Conversely, gradient-free attacks create adversarial samples by analyzing only the model's outputs, requiring no access to its internal mechanisms [40]. In addition, two further strategies are commonly employed. The first involves training surrogate models to approximate the decision boundary of the target NIDS, with the expectation that adversarial examples effective against the surrogate will also bypass the actual model. The second utilizes sophisticated optimization algorithms to craft input samples specifically designed to mislead the model [2]. To provide a structured overview, Table 2.5 summarizes various evasion attack methods based on the adversary's level of knowledge, distinguishing among White-box, Grey-box, and Black-box scenarios.

- **Model inversion attacks:** To the best of our knowledge, no research has documented this specific exploitation of the ML-based NIDS retraining loop using static payloads. This attack involves an adversary persistently persuading an ML-based NIDS that a malicious payload is benign through repetition rather than perturbation. The process conceptually resembles 'brute forcing' malicious examples through the NIDS model, with the goal that, through the automated re-training pipeline, the system will eventually mislabel the malicious samples as benign. By repeatedly submitting the same malicious payload to a NIDS that relies on its previous classifications as "pseudo-labels" for incremental learning, the attacker induces boundary drift. Over time, this shifts the model's decision boundary and may ultimately allow malicious samples to bypass detection.

Figures 2.14 and 2.15 illustrate each of the attacks discussed earlier. Additionally, Figure 2.16 provides a comprehensive summary of all the previous adversarial attack strategies.

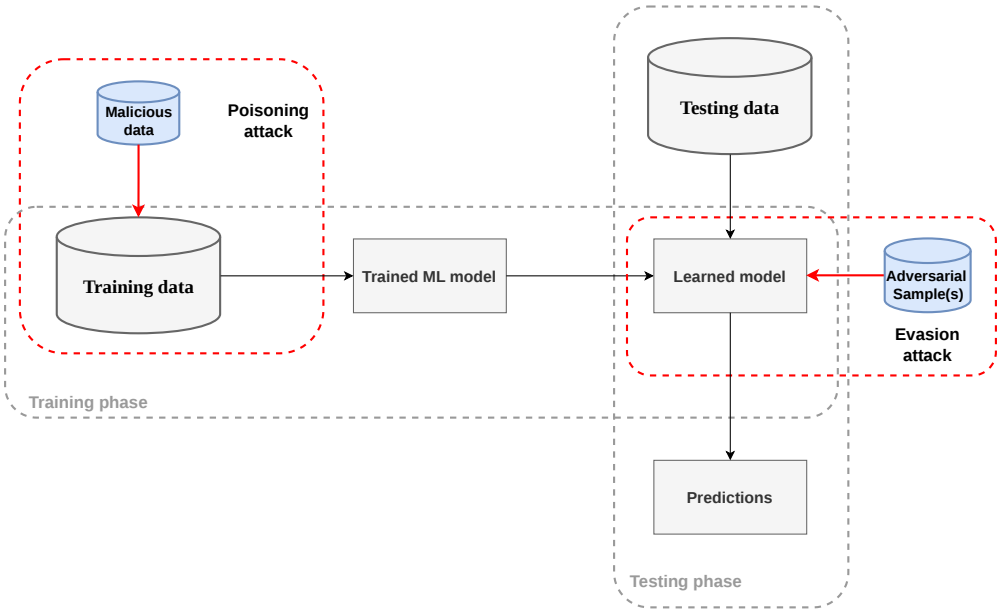


Figure 2.14: Illustration of Evasion and Poison attacks,based on [37]

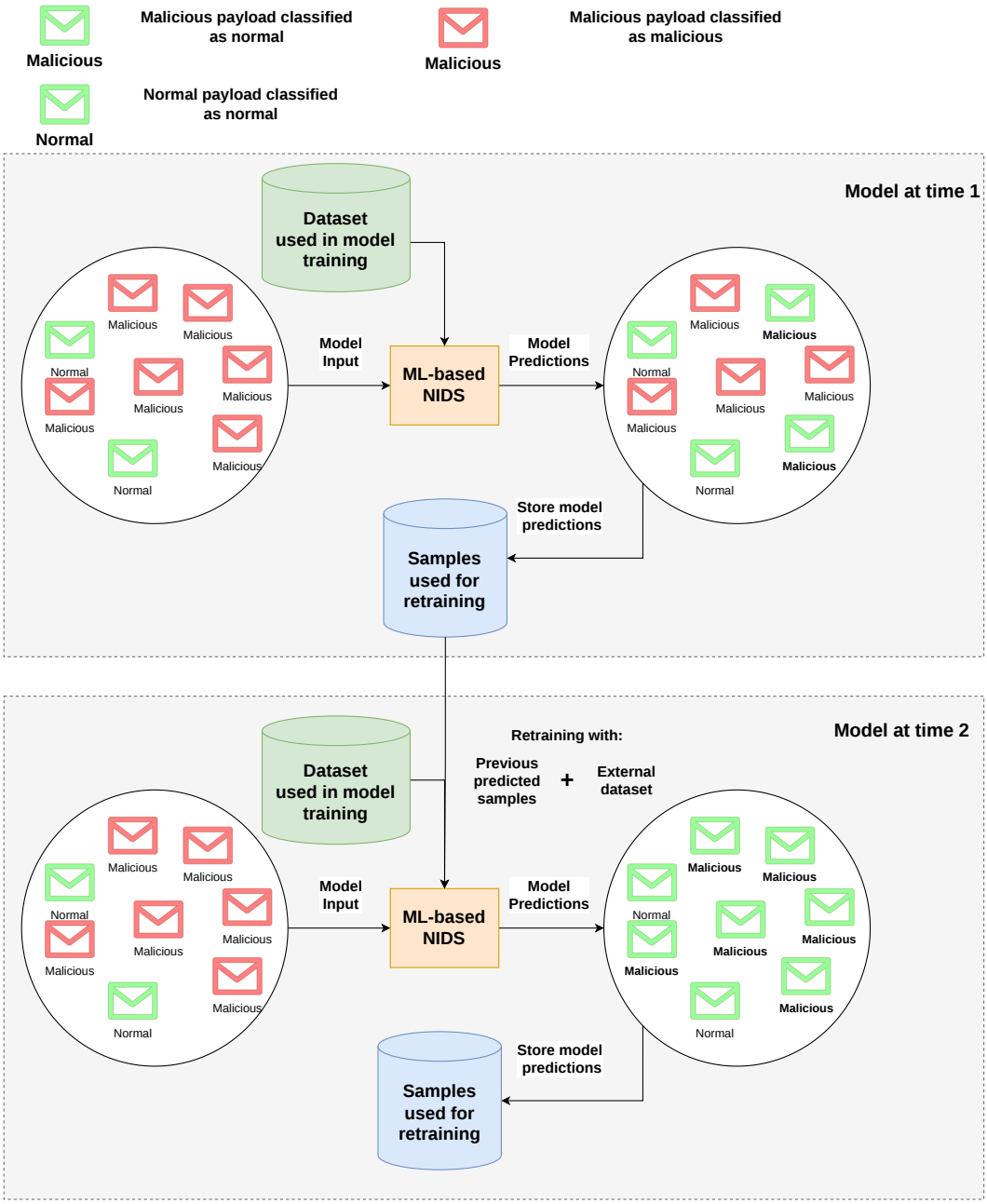


Figure 2.15: Illustration of model inversion attacks

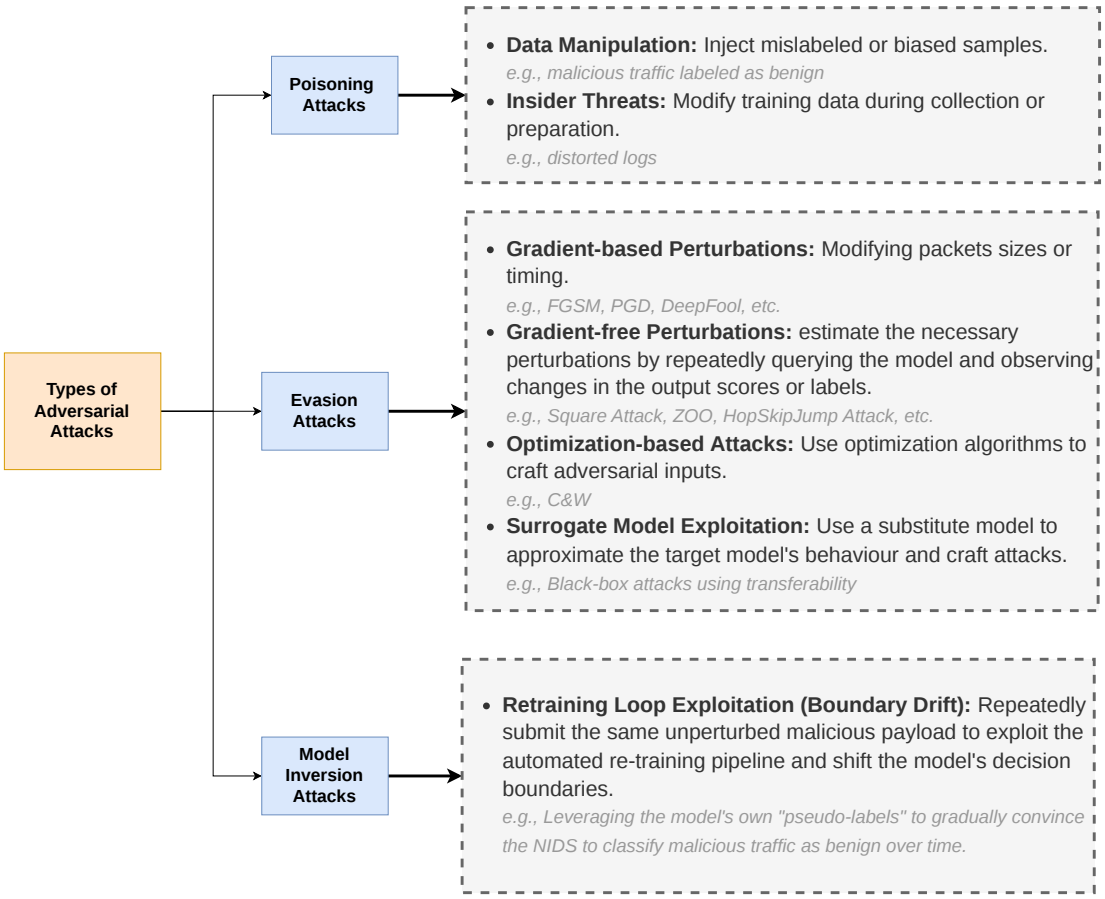


Figure 2.16: Types of attack strategies (adapted from [2])

Adversarial research currently prioritizes evasion attacks, neglecting data poisoning and model inversion threats to incremental learning.

Method	Knowledge	Reference
BIM		[41]
C&W		[42]
DeepFool		[43]
FGSM	**White-box**	[44]
Houdini		[45]
JSMA		[46]
PGD		[47]
GSA-GAN		[48]
IDS-GAN		[49]
Polymorphic	**Grey-box**	[50]
A2PM		[51]
DoSBoundary		[52]
BFAM		[53]
BMI-FGSM		[54]
OnePixel		[55]
WGAN	**Black-Box**	[56]
ZOO		[57]
Boundary		[58]
GADGET		[59]
HopSkipJump		[60]

Table 2.5: Evasion attack methods and their classifications based on adversary knowledge (adapted from [61])

2.5.3 Adversarial goal

Adversarial attacks can be targeted, aiming to misclassify inputs as a specific incorrect class, or untargeted, designed to cause any misclassification without specifying a particular class [2]. Untargeted attacks are generally easier to carry out because they do not focus on a specific class and simply cause misclassification. However, this is only true in multiclass classification; in binary classification, untargeted and targeted attacks are effectively the same [26].

2.5.4 Attack frequency

Adversarial attacks can be either one-shot, generated through a single-step algorithm, or iterative, relying on multiple queries to refine the perturbations [2].

2.5.5 Adversarial Defenses

Adversarial samples have been shown to be effective against various ML models, including those used in NIDS. To reduce the effects of these attacks, researchers have proposed

techniques to improve model robustness against such threat [2]. This work classifies existing defenses according to the framework in [2], which distinguishes adversarial defenses into adversarial training and architecture reinforcement, while also considering continuous training, as introduced by [62], as a third supplementary category, with the main points summarized below.

1. **Adversarial training:** This method involves adding adversarial samples to the training dataset during the training phase, which improves the model's robustness against adversarial attacks [2], [63]. According to [2], AT is defined as the process of finding a set of model parameters θ that minimize expression 2.1, where δ is an adversarial perturbation, Δ consists of a set of allowable adversarial perturbations, D denotes the data distribution over inputs $x \in \mathbb{R}^d$ and labels y, $l(\theta, x, y)$ is the loss function for the model parameters θ. In summary formula 2.1 defines the minimax optimization process for adversarial robustness. It aims to find model parameters θ that minimize the expected loss when facing a "worst-case' adversary who identifies a perturbation δ to maximizes the error within a defined constraint set Δ. Many researchers in the network security domain have found that AT improves ML model classification accuracy and resilience against adversarial attacks [64, 65].

$$\underset{\theta}{\text{argmin}}\ E_{(x,y)\sim D}\left[\max_{\delta\in\Delta} l(\theta, x+\delta, y)\right] \tag{2.1}$$

2. **Architecture reinforcing:** Involves multiple approaches to enhance the model's architecture using different strategies, outlined below:

   - **Detection and Rejection**: It involves using trained detector $g(x)$ to evaluate model inputs and detect potential adversarial perturbations [2]. When an adversarial perturbation is detected, the system executes a series of specialized steps to counteract or remove the identified threat [66]. According to [2], inputs exceeding a predefined threshold T are flagged and either rejected or transformed following equation 2.2, where $f_\theta(x)$ is the model's output for input x, x_{clean} is the reconstructed clean input, and $\mathbb{I}$ is the indicator function. Overall, function $g(x)$ acts as a binary classifier that flags an input as adversarial if the divergence between the model's output for a test input x and a reconstructed clean version x_{clean} exceeds a specific sensitivity threshold T.

$$g(x) = \mathbb{I}(\|f_\theta(x) - f_\theta(x_{\text{clean}})\| > T) \tag{2.2}$$

- **Input transformation:** Executes a series of transformations on the input data to remove or greatly reduce potential adversarial perturbations before using the samples in the model [2]. Input transformations usually focus on disrupting features changed by adversarial perturbations to reduce the impact of attacks [67]. As described in [2], this transformation function is detailed in equation 2.3.

$$x_{\text{transformed}} = T(x) \tag{2.3}$$

  $T(x)$ can include:

  (a) **Noise reduction:** $T(x) = filter(x)$, which removes high-frequency components most likely caused by adversarial perturbations.

  (b) **Feature smoothing:** $T(x) = x - \nabla_x \mathcal{L}(f_\theta(x), y)$ reduces the gradient influence of perturbed features to enhance stability of the input.

- **Gradient Masking:** Intends to obscure or alter the model's gradient information to prevent adversarial manipulations [2]. Most adversarial attacks depend on gradient computations to find effective perturbations, and restricting or distorting this gradient information can make the model less vulnerable [68]. As described in [2], this can be represented by equation 2.4, where $\tilde{\nabla}_x$ is the masked or approximated gradient, and $\mathcal{L}(f_\theta(x), y)$ is the loss function calculated for the model $f_\theta(x)$, input x, and label y. This approach ensures that adversarial perturbations δ, typically calculated using equation 2.5, cannot be effectively generated because the true gradient $\nabla_X \mathcal{L}$ is either unavailable or unreliable [2]. By introducing randomness or deliberately altering the gradient, models can defend against gradient-based attacks, although this often results in reduced accuracy [2].

$$\tilde{\nabla}_x \mathcal{L}(f_\theta(x), y) \approx 0 \tag{2.4}$$

$$\delta = \epsilon \cdot \text{sign}(\nabla_x \mathcal{L}(f_\theta(x), y)) \tag{2.5}$$

- **Manifold Projection**: Adversarial attacks typically work by moving data points into regions of the domain where the model is not well-trained. To counter this, projecting these points back onto the data manifold, which represents the underlying structure of ”natural” data, removes the harmful noise while keeping the original signal intact [2]. As described in [2], the process can be expressed as equation 2.6, where $\mathcal{M}$ presents the manifold of clean data, $\|x - z\|_p$ is the p-norm measuring the distance between the input x and the closest point z on the manifold. The manifold $\mathcal{M}$ can be learned using techniques like AEs [69]. As detailed by the authors in [2], for an AE with an encoder E and a decoder D, the manifold projection is expressed as equation 2.7, where $E(x)$ transforms input x into a lower-dimensional space, and $D(E(x))$ reconstructs x from its encoded representation.

$$x_{\text{projected}} = \underset{z \in \mathcal{M}}{\text{argmin}} \|x - z\|_p \tag{2.6}$$

$$x_{\text{projected}} \approx D(E(x)) \tag{2.7}$$

- **Ensemble methods:** Utilizes ensemble approaches that combine various algorithms, offering a more robust defense against adversarial attacks.

3. **Continuous training**: According to the authors in [62], continuous retraining significantly reduces the impact of adversarial attacks on ML-based NIDS, even without AT. Their experiments revealed that while attacks like FGSM, PGD, and BIM initially caused notable performance declines, the models recovered and achieved accuracy and F1-scores above 0.9 after just one daily retraining session. This resilience arises because periodic retraining modifies the model’s decision boundaries and gradients, rendering the attacker’s prior gradient knowledge ineffective against the updated model. Nonetheless, the authors note that this defensive benefit likely does not extend to simple ”blind” perturbations, as these attacks do not rely on gradient knowledge to bypass detection.

3. Related work

This chapter reviews recent advances in ML for anomaly-based NIDS, with key findings summarized in Tables 3.1 and 3.2. The referenced works vary in focus: while some address adversarial attacks and propose corresponding defenses, others do not consider this aspect. The following paragraphs outline the main contributions and conclusions from each study.

Roshan et al. [70] evaluated four adversarial attacks (FGSM, JSMA, PGD, C&W) on DL-based NIDS and tested three defenses (AT, GDA, HC) in a real-time network setting. They observed that NIDS are highly susceptible, with attacks impairing detection capabilities, but implementing defenses enhanced their robustness. They recommend avoiding the deployment of DL-based NIDS in mission-critical systems unless robust defenses are in place.

Yuan et al. [71] introduced a DLL-IDS framework boosting DL-based NIDS robustness. It evaluates network traffic with a DL model, an LID adversarial detector, and a robust LS model. The authors confirm this hybrid system improves defense against adversarial attacks by better identifying malicious traffic and maintaining high detection accuracy.

Roshan et al. [72] proposed a two-phase defense to enhance DL-based NIDS robustness against untargeted white-box C&W attacks. They employed GDA during training and FS during testing, evaluating their approach on the CIC-DDoS2019 dataset. This method exceeds single-phase defenses, boosting metrics like accuracy, precision, recall, and F1-score in the face of adversarial threats.

Değirmenci et al. [73] introduced READ, a NIDS method combining reconstruction error with aleatoric, epistemic, and entropy metrics to detect adversarial traffic. Tested against FGSM, PGD, BIM adversarial attacks across three datasets, it achieved a 92% to 100% detection rate and outperformed other methods at low levels of perturbations, restoring system accuracy up to 98% during attacks.

Zhang et al. [74] introduce the TIKI-TAKA framework to evaluate the vulnerability of DL-based NIDS and suggest new defensive strategies. They test three NN models against five black-box adversarial attacks and propose three defenses: model voting ensembling, ensembling AT, and adversarial query detection. Their results show that current DL NIDS

are highly susceptible, with attackers bypassing detection up to 35.7%, but their proposed defenses effectively counter these threats, restoring detection rates close to 100%.

Alotaibi et al. [75] contribute to network security by developing two CNN-based NIDS models, testing vulnerabilities against seven distinct adversarial attacks, and implementing an APE_GAN++ defense to counter these threats. They conclude that while current CNN-based IDSs are vulnerable to adversarial manipulations that impair detection, the APE_GAN++ defense effectively removes these perturbations, restores the models' accuracy to highly reliable levels.

Soleymanzadeh et al. [76] address training instability and mode collapse in intrusion detection. Their approach involves an ensemble of generators and discriminators for stable training and a multi-player strategy to prevent mode collapse. They demonstrate that EMP-GANs produce high-quality synthetic minority attack samples that closely match real data, reducing mode collapse and enhancing classifier accuracy and performance.

Zolbayar et al. [77] developed NIDSGAN, a new GAN algorithm that creates realistic adversarial traffic flows to evade ML-based NIDS, while following real-world network constraints. Their main achievements include developing this constraint-enforcing algorithm, evaluating it across various threat models, and incorporating an active learning technique that improves attack success and reduces training data needs. They conclude that current ML-based NIDS are highly vulnerable to these crafted flows across all threat models, indicating that deploying such systems without strong defenses could lead to future network breaches.

Hnamte et al. [78] proposed a two-stage DL-based NIDS that combines LSTM and AE, incorporating data filtering to minimize overfitting and address imbalanced datasets. Their LSTM-AE model, optimized with MAE over 30 epochs, achieved accuracy rates of 99.99% on CICIDS2017 and 99.10% on CSE-CICIDS2018. They concluded that this hybrid approach surpasses traditional models such as DNNs and CNNs for cyber-attack detection. However, despite these findings, the authors did not consider adversarial attacks targeting their model.

Wang et al. [79] evaluated six DL models (DNN, CNN, RNN, LSTM, CNN + RNN, and CNN + LSTM) on the CSE-CIC-IDS2018 dataset, demonstrating that all models can achieve

over 98% accuracy with appropriate data preprocessing. They concluded that single models (DNN, RNN, CNN) are more suitable for real-world NIDS applications than hybrid models, which offer only slight accuracy improvements but longer inference times. Despite this thorough evaluation, the authors did not consider adversarial attacks when assessing any of the models.

Karatas et al. [80] enhanced IDS performance by evaluating six ML algorithms on the CSE-CIC-IDS2018 dataset and applying SMOTE to mitigate data imbalance. They observed that generating synthetic data for minority classes decreases bias toward major classes, improving detection of rare intrusions and raising model accuracy by 4.01% to 30.59%. The authors did not explore adversarial attacks.

Ghadermazi et al. [3] proposed an AI framework for near real-time network intrusion detection, overcoming traditional limits by extracting header and payload data. It transforms network packets into 2D images analyzed by a CNN, using a packet-based approach rather than flow-based (more details in chapter 6). The method achieves detection rates of 97.7% to 99%, enables early threat detection within 4-9 packets, and shows robustness against adversarial attacks and adaptability to new network enviroments without retraining.

Awad et al. [81] created a two-phase ensemble defense for DL-based NIDS using AT, GDA, LS, and AD to combat evasion attacks while maintaining network traffic features. This approach boosts adversarial robustness and classification, overcoming single defense limits by combining strengths, with high accuracy on clean data.

Debicha et al. [82] study advances network security by implementing DL-based NIDS in serial and parallel architectures, proposing a transfer learning-based adversarial detection scheme, and testing multiple adversarial detectors against unseen evasion attacks. They conclude that deploying multiple detectors with Dempster-Shafer fusion rule in a parallel NIDS design greatly enhances detection rates over single detectors or previous methods.

Roshan et al. [83] conducted an empirical study assessing how vulnerable DL-based NIDS are to four adversarial attacks (FGSM, JSMA, PGD, and C&W) and implemented AT as a defensive measure. Their findings indicate that NIDS models are highly susceptible to adversarial examples, which greatly impair accuracy and performance. However, implementing AT effectively counteracts these threats, significantly restoring the models' robustness and confidence scores.

He et al. [84] introduced a defense strategy called MTD-AD to safeguard NIDS from adversarial threats by randomly modifying their decision boundaries. The authors concluded that MTD-AD effectively improves NIDS resilience against both standard and adaptive adversaries while requiring little prior knowledge from the defender. Additionally, they found that creating defense models with appropriate, rather than maximal, diversity leads to better results. They also observed that adaptive attackers must drastically compromise their attack's intensity to successfully evade the MTD-AD system.

Wang et al. [85] investigate adversarial example attacks on ML-based NIDS and propose MANDA, a novel adversarial exampe detection method. MANDA identifies adversarial examples by comparing manifold evaluations with IDS model inferences and measuring model uncertainty near decision boundaries. The study finds that while problem-space AE attacks threaten IDSs, MANDA effectively distinguishes adversarial from clean traffic. MANDA outperforms state-of-the-art statistical models, achieving a 98.41% true-positive rate and 5% false-positive rate under strong attacks, with potential applications beyond IDSs.

Zhang et al. [86] propose the ETA attack framework for ML-based NIDS, using a novel feature selection method to enable transfer-based adversarial attacks. Their method achieves a 70% average attack success rate and shows that vulnerabilities stem from models using non-robust features. They also argue that robust NIDS should prioritize intrinsic, robust features over accuracy.

Alotaibi et al. [87] present Mateen, an adaptive ensemble learning framework for NIDS that detects and adapts to benign traffic shifts. Mateen uses statistical shift detection, unsupervised sample selection to reduce labeling, and dynamic complexity reduction. It outperforms existing methods on various shift scenarios, maintaining anomaly detection accuracy with only 0.5% to 1% labeled data.

Barik et al. [88] developed IDS-Anta, an open-source Python repository offering a strong defense against adversarial attacks on NIDS while maintaining performance. IDS-Anta combats attacks like ZOO and FGSM by adaptively choosing the best classifiers via Multi-Armed Bandits with Thomson Sampling and Ant Colony Optimization. The authors found that, despite the continued vulnerability of ML and DL-based NIDS to adversarial samples, IDS-Anta significantly reduces these risks. Although it can't block every attack, IDS-Anta greatly raises the cost and effort for attackers, making exploits less feasible and providing a substantial deterrent.

Table 3.1: Recent advancements in ML for anomaly-based NIDS.

Ref.	Fam.	Dataset(s)	Model(s)	Adversarial setting	Defense
[70]	DL	CIC-2017	DNN	WB; FGSM, JSMA, PGD, C&W	AT, GDA, HC
[71]	ML+DL	NSL-KDD; CIC-2018	DNN; LS	WB/GB/BB; FGSM, BIM, DeepFool, C&W	DLL-IDS
[72]	DL	CIC-2019	DNN	WB; C&W	GDA, FS
[73]	ML+DL	CIC-2018; KDDCup99	CNN, LR	WB; FGSM, PGD, BIM	READ
[74]	DL	CIC-2018; CIC-2017	MLP, CNN, CNN+LSTM	BB; Pointwise, NES, Boundary	Ensemble AT; Query Det.
[75]	DL	CIC-2017	NN, GAN	WB; FGSM, JSMA, BIM, DeepFool, PGD, C&W, Auto-PGD	APE_GAN++
[76]	DL	NSL-KDD; UNSW-NB15	EDL, CNN	–	–
[77]	DL	NSL-KDD; CIC-2017	DNN, GAN	WB/BB; NIDSGAN	–
[78]	DL	CIC-2017; CIC-2018	AE, LSTM	–	–
[79]	DL	CIC-2018	DNN, CNN, RNN, LSTM, CNN+RNN, CNN+LSTM	–	–
[80]	ML	CIC-2018	KNN, RF, GB, AdaBoost, DT, LDA	–	–
[3]	DL	CIC-2017; CIC-2018	CNN	BB; specific perturbations	SPIN-IDS
[81]	DL	CIC-2017; CIC-2018	DNN, DAE	BB/GB; FGSM, PGD, DeepFool, JSMA, C&W	AT, LS, GDA, DSA
[82]	DL	NSL-KDD; CIC-2017	Ensemble of DNNs	GB; FGSM, PGD, DeepFool	Transfer-based adv. detector
[83]	DL	CIC-2017	DNN	WB; FGSM, JSMA, PGD, C&W	AT
[84]	DL	UQ-2021	TM: Ensemble AE; DM: SOM, LOF, SVM	WB/GB; FGSM, PGD, DeepFool	MTD-AD
[85]	ML+DL	NSL-KDD; CIC-2017; MNIST	TM: MLP, CNN, others; DM: LR, TL	WB; FGSM, BIM, JSMA, C&W	MANDA
[86]	ML+DL	CIC-2017; Kitsune	SM: MLP, DT, others; TM: MLP, RF, AE, others	BB; ETA	–
[87]	DL	CIC-2017; CIC-2018; Kitsune	DAEs, MLPs	–	–
[88]	ML+DL	CIC-2017; CIC-2018; CIC-2019	RF, SVM, LR, DNN	BB/WB; ZOO, FGSM	IDS-Anta

Fam.: family of methods (ML, DL, ML+DL).
CIC-2017: CIC-IDS2017; *CIC-2018*: CSE-CIC-IDS2018; *CIC-2019*: CIC-DDoS2019; *UQ-2021* : UQ-IoT-IDS-2021.
WB, *GB*, *BB*: white-box, gray-box, black-box.
AT: adversarial training; *GDA*: Gaussian data augmentation; *HC*: high-confidence; *FS*: feature squeezing; *LS*: label smoothing; *DSA*: dynamic sample augmentation.
ETA: Explainable Transfer-based Attack.
READ, *MANDA*, *SPIN-IDS*, *MTD-AD*, *IDS-Anta*, *DLL-IDS*, *APE_GAN++*: defense/framework names as reported in the original works.
TM/*DM*/*SM*: target/defense/substitute model (when used by the authors).

Table 3.2: Comparison of recent contributions in anomaly-based NIDS

		Strengths			Limitations			Gaps		
Ref.	**Focus**	**Perf.**	**Rob.**	**Dep.**	**Cplx.**	**Scope**	**Eval.**	**Gen.**	**Real.**	**Eff.**
[70]	Robust.	●	◐	●	○	●	●	●	●	◐
[71]	Defense	●	●	◐	●	●	○	◐	●	●
[72]	Defense	●	◐	◐	●	●	●	●	●	○
[73]	Defense	●	●	◐	●	○	○	○	●	●
[74]	Robust.	●	●	◐	●	●	○	◐	◐	●
[75]	Defense	●	●	○	●	●	○	◐	●	●
[76]	Detect.	●	○	○	●	◐	●	◐	●	●
[77]	Attack gen.	●	○	○	●	◐	○	○	○	●
[78]	Detect.	●	○	◐	●	○	●	◐	●	●
[79]	DL bench.	●	○	◐	●	●	●	●	●	●
[80]	ML base.	●	○	●	○	●	●	●	●	◐
[3]	Evasion res.	●	●	●	◐	◐	◐	○	○	○
[81]	Defense	●	●	○	●	○	○	○	●	●
[82]	Adv. det.	●	●	◐	●	◐	○	○	●	●
[83]	Defense	●	◐	●	○	●	●	●	●	◐
[84]	Adapt. def.	●	●	○	●	●	◐	◐	●	●
[85]	Adv. det.	●	●	◐	◐	○	○	○	◐	◐
[86]	Attack gen.	●	○	○	●	○	○	○	○	◐
[87]	Detect.	●	○	◐	●	○	◐	○	●	●
[88]	Adv. det.	●	●	●	◐	◐	●	◐	●	●

Legend: ● = strong / clearly present; ◐ = partial / moderate; ○ = absent / not evidenced.
For **Strengths**, symbols indicate the degree to which a contribution demonstrates performance (Perf.), adversarial robustness (Rob.), and deployment potential (Dep.).
For **Limitations**, symbols indicate the severity of complexity/overhead (Cplx.), narrow validation scope (Scope), and evaluation weakness (Eval.).
For **Gaps**, symbols indicate the extent to which open issues remain regarding generalization (Gen.), attack/traffic realism (Real.), and efficiency/deployability (Eff.).

4. Datasets for NIDS

This chapter presents some common datasets used for developing anomaly-based NIDS and provides a brief description of each.

4.1. UNSW-NB15

The UNSW-NB15 dataset acts as a comprehensive benchmark for NIDS. Developed by the UNSW Canberra Cyber Range Lab in 2015, it was designed to address the shortcomings of earlier datasets such as KDDCUP99, by accurately reflecting more current network traffic and low-footprint attack types. The dataset contains 2,540,044 objects and 49 features and includes two ground truth labels: one indicating the attack type and another for binary classification. Each record is classified into attack categories like Normal, Fuzzer, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode, and worms. It merges real-world normal activity data with synthetic, contemporary attack patterns to create a comprehensive testing environment [89]. Table 4.1 shows the distribution of attacks in UNSW-NB15.

Attack type	Number of samples	Distribution percentage
Normal	2218761	87.35
Fuzzer	24246	0.96
Analysis	2677	0.11
Backdoors	2329	0.09
DoS	16353	0.64
Exploits	44525	1.75
Generic	215481	8.48
Reconnaissance	13987	0.55
Shellcode	1511	0.06
Worms	174	0.01
Total	**2540044**	**100**

Table 4.1: UNSW-NB15 distribution of Attack Types [89]

4.2. CIC-IDS2017

The CICIDS2017 dataset, developed in 2017 by the Canadian Institute for Cybersecurity, was designed to address the shortcomings of older benchmarks such as KDD99 and DARPA. It offers a trustworthy and extensive dataset that meets eleven key evaluation standards, including attack diversity and complete network configuration. Created within a realistic testbed environment, it features a Victim-Network (running Windows, Linux, and Mac) and a separate Attack-Network, with traffic recorded continuously over five days.

The dataset includes benign traffic generated by a B-Profile system simulating human activity across protocols like HTTP, HTTPS, FTP, and SSH, alongside seven common attack types: Brute Force, Heartbleed, Botnet, DoS, DDoS, Web Attacks, and Infiltration. It contains 2830540 rows and 79 features, of which 78 are numerical and the 'label' is categorical [90]. Table 4.2 shows the distribution of attacks in CIC-IDS2017.

Attack type	Number of samples	Distribution percentage
Normal	2359087	83.3441
DoS Hulk	231072	8.1635
PortScan	158930	5.6148
DDoS	41835	1.478
DoS GoldenEye	10293	0.3636
FTP-Patator	7938	0.2804
SSH-Patator	5897	0.2083
DoS slowloris	5796	0.2048
DoS slowhttptest	5499	0.1943
Botnet	1966	0.0695
Web Attack - BF	1507	0.0532
Web attack - XSS	652	0.023
Infiltration	36	0.0013
Web attack - Inj.	21	0.0007
Heartbleed	11	0.0004
Total	**2830540**	**$\sim$100**

Table 4.2: CIC-IDS2017 distribution of Attack Types [90]

4.3. CIC-DDoS2019

The CICDDoS2019 dataset is a comprehensive benchmark also created by the Canadian Institute for Cybersecurity to address the limitations of earlier datasets. It provides fully labeled, non-anonymized data that covers a wide variety of modern DDoS attacks. The dataset was generated in a realistic testbed environment with a "Victim-Network" and an "Attack-Network," including 12 different attack types such as NTP, DNS, LDAP, MSSQL, NetBIOS, SNMP, SSDP, UDP, UDP-Lag, WebDDoS, SYN, TFTP. It also features naturalistic benign background traffic generated via B-Profile, which simulates human behavior across protocols like HTTP, HTTPS, FTP, SSH, and email. The dataset includes more than 80 traffic features and over 50 milion records [91]. Table 4.3 shows the distribution of attacks in CIC-DDoS2019.

Attack type	Number of samples	Distribution percentage
TFTP	20082580	28.515
Syn	6473789	9.192
MSSQL	5787453	8.218
DrDoS_SNMP	5159870	7.326
DrDoS_DNS	5071011	7.2
DrDoS_MSSQL	4522492	6.421
DrDoS_NetBIOS	4093279	5.812
UDP	3867155	5.491
NetBIOS	3657497	5.193
DrDoS_UDP	3134645	4.451
DrDoS_SSDP	2610611	3.707
DrDoS_LDAP	2179930	3.095
LDAP	1915122	2.719
DrDoS_NTP	1202642	1.708
UDP-lag	366461	0.52
Portmap	186960	0.265
Normal	113828	0.162
UDPLag	1873	0.003
WebDDoS	439	0.001
Total	**70427637**	**∼100**

Table 4.3: CIC-DDoS2019 distribution of Attack Types [91]

4.4. LITNET-2020

LITNET-2020 is a benchmark dataset for NIDS gathered from a real academic network in Lithuania over ten months, from March 2019 to January 2020. It aims to overcome the limitations of synthetic or outdated datasets by capturing authentic traffic, including normal activity and 12 specific attack types such as DoS, scanning, and worms.The dataset includes 85 features per flow, which cover basic network details as well as 19 specialized attack-detection features, and is accurately labeled to differentiate between background traffic and malicious actions [92]. Table 4.4 shows the distribution of attacks in LITNET-2020.

Attack type	Number of samples	Distribution percentage
Normal	36423860	91.971
TCP SYN-flood	1580016	3.99
Code Red Worm	1255702	3.171
Smurf	118958	0.3
UDP-flood	93583	0.236
LAND attack	52417	0.132
W32.Blaster Worm	24291	0.061
ICMP-flood	23256	0.059
HTTP-flood	22959	0.058
Scanning/Spread	6232	0.016
Reaper Worm	1176	0.003
Spam bot	747	0.002
Packet fragmentation	477	0.001
Total	**39603674**	**100**

Table 4.4: LITNET-2020 distribution of Attack Types [92]

5. Metrics

This chapter describes the metrics used to evaluate the DL models used in this study. Section 5.1 discusses the primary metrics for binary classification, including components of the confusion matrix, Accuracy, Precision, Recall, F1-Score, and the Area Under the Curve for Precision-Recall. Section 5.2 extends these concepts to multiclass classification, covering per-class metrics and the averaging methods, Macro and Weighted, which offer a comprehensive assessment of system performance across all target classes.

5.1. Binary Classification metrics

The following section discusses common metrics used to evaluate ML-based NIDS performance in binary classification, where the target feature is either attack or normal. Before outlining these metrics, it is essential to define the key components of the confusion matrix relevant to the NIDS context:

- True Positive (TP): When the classifier correctly identifies data objects as an attack.
- True Negative (TN): The instances are accurately identified as normal.
- False Positive (FP): Data instances incorrectly identified as attacks.
- False Negative (FN): Data instances incorrectly labeled as normal.

Using elements from the confusion matrix, the following performance metrics can be determined:

- **Accuracy**: Measures the percentage of correct predictions made by a model. In NIDS, accuracy can be misleading because class imbalance allows a model to achieve high scores simply by identifying normal traffic and missing rare attacks. For this reason, accuracy is not a reliable metric for evaluating NIDS performance.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

- **Precision**: Refers to the percentage of correctly predicted Attack samples out of all actual Attack samples. In practice, high precision means the NIDS generates fewer false alarms.

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

- **Recall**: Also known as the true positive rate (TPR) or Sensitivity, it is calculated by dividing the number of correctly predicted Attack cases (TP) by the total number of actual attack instances. Maximizing recall is highly critical to ensure malicious packets do not slip through the NIDS undetected.

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

- **F1-Score**: Represents the balance point between Precision and Recall. It's a statistical measure used to assess a system's reliability by evaluating its performance in terms of precision and recall.

$$F1\text{-}Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{5.4}$$

- **PR AUC**: The Precision-Recall Area Under the Curve condenses the entire PR curve into a single metric by calculating the area under the curve across all thresholds. A higher PR AUC indicates that the model sustains high recall while maintaining precision [93], which is essential for NIDS, where overlooking an attack (FN) can be very costly.

5.2. Multiclass Classification metrics

In a multiclass classification setting, the target feature includes multiple categories (e.g., Normal, DoS, DDoS, Portscan, etc.). While the basic principles of the confusion matrix remain the same, they must be adjusted to evaluate the model's performance across N classes. This is typically achieved using a One-vs-All approach, where metrics are calculated for each class i by considering it as the positive class and all others as negative [94]. In the following subsections, the metrics and their components used in multiclass classification are briefly discussed.

5.2.1 Per-Class Components

For each class i, the main elements of the confusion matrix are defined as:

- TP_i: Instances of class i correctly predicted as class i.

- TN_i: Instances of other classes correctly predicted as not belonging to class i.
- FP_i: Instances of other classes incorrectly predicted as class i.
- FN_i: Instances of class i incorrectly predicted as any other class.

Based on the confusion matrix for multiclass elements above, the equations for binary metrics (accuracy, precision, recall, etc.) can be easily derived by replacing the components of the binary confusion matrix with those of the multiclass confusion matrix.

5.2.2 Averaging Techniques

To provide a overview of NIDS performance across all N attack types, individual class metrics can be synthesized using, for example, one of the following averaging methods:

- **Macro-Averaging**: Calculates the metric separately for each class and then averages them without weighting. This approach treats all classes equally, making it a rigorous metric that emphasizes poor performance in minority attack classes [94].

$$Macro\text{-}Metric = \frac{1}{N}\sum_{i=1}^{N} Metric_i \tag{5.5}$$

- **Weighted-Averaging**: Calculates the average of all per-class metrics, with each class weighted by its $Support$ (the count of true instances of that class) [94]. This is highly relevant for NIDS datasets, which tend to be imbalanced, with frequently occurring ”Normal” traffic and infrequent specific attacks.

$$Weighted\text{-}Metric = \sum_{i=1}^{N}\left(Metric_i \times \frac{Support_i}{Total\ Instances}\right) \tag{5.6}$$

6. Flow based vs Packet based

ML-based NIDS employ two main types of feature extraction methodologies to analyze network traffic data: flow-based features, which gather information from packet headers, and packet-based features, which are derived directly from packet data [3], as illustrated in Figure 6.1.

Flow-based NIDS models, which analyze traffic between sender and receiver to detect malicious activities only after the communication flow has fully terminated, are more suitable for offline traffic analysis and have several limitations [95]. Because these models primarily extract features from the lower layers of the TCP/IP stack, they often struggle to identify attacks at the application layer [96]. Moreover, their reliance on flow features, which do not capture the functional behaviors of individual network packets, can hinder accurate attack detection [3]. To complicate matters further, flow-based feature extraction tools like CIC Flowmeter [97] and Zeek (Bro) [98] generate different sets of features, making it difficult to benchmark trained models in new network environments [3].

Packet-based NIDS are well suited for real-time or near real-time detection because they extract features directly from raw packet data. However, this approach presents several challenges. Distinguishing benign packets from those related to attacks can be difficult, since not every packet signals malicious intent. For example, the establishment of a TCP connection is typically harmless, even though subsequent packets may indicate suspicious activity [3]. In addition, most packet-based NIDS analyze each packet independently and overlook the order in which packets appear within a flow. As a result, they fail to capture the temporal relationships among packets [99]. Another common limitation is often overlooking packet direction. However, understanding whether packets are moving forward or backward within a flow can be vital for attack detection. For example, DDoS attacks frequently show notable differences in forward and backward traffic patterns compared to regular traffic [3], [4].

In current research, flow-based feature extraction continues to be increasingly prominent [3], [100]. The widespread use of flow-based models is supported by the availability of structured datasets like CIC-IDS2017 [90] and UNSW-NB15 [89], which are often provided in CSV formats with pre-extracted flow features. Although these datasets typically include the original PCAP files, most published studies prefer using the summarized flow

records because they are more suitable for traditional ML algorithms. While flow-based features have become increasingly popular, this thesis intentionally emphasizes packet-based features for their more immediate real-world applicability, eliminating the impracticality of waiting for an entire communication session to end before issuing an alert.

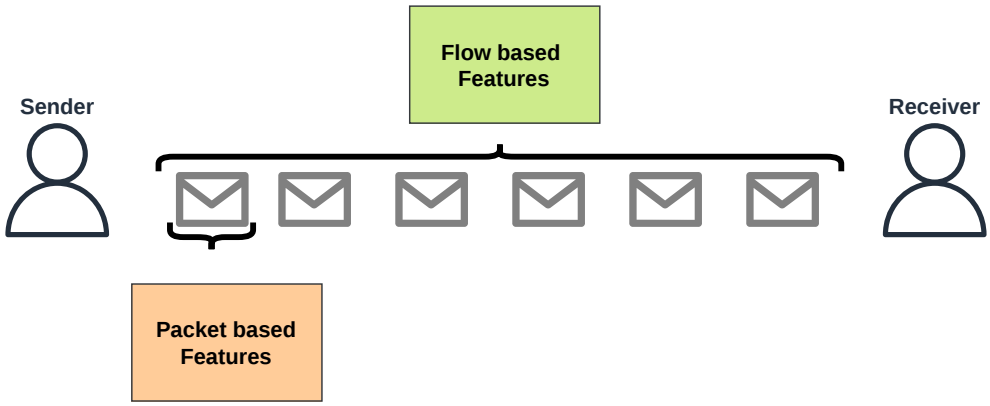


Figure 6.1: Flow and Packet based features

7. Proposed Methodology

This chapter explains how the research was practically carried out, describing the steps used to build, train, and test the network intrusion detection models. As shown in Figure 7.1, the process is divided into three main stages: (1) creating a labeled, image-based dataset from network traffic, (2) testing and tuning the hyperparameters of the DL models, and (3) evaluating adversarial robustness and implementing defensive strategies.

The foundation of this methodology is built on the CIC-IDS2017 dataset, which was chosen as the main data source of network traffic. This choice was influenced by several important reasons. First, CIC-IDS2017 serves as a modern, widely recognized benchmark that accurately represents current network topologies and includes a variety of realistic attack scenarios [90]. Second, the feature extraction tool used in this work, Payload-Byte [101], has a pre-configured pipeline specifically optimized for parsing and labeling raw pcap files from the CIC-IDS2017 dataset. This guarantees high data quality, reduces labeling conflicts, and ensures a consistent preprocessing process.

A key difference in the evaluation methods for the chosen DL models, CNN and AE, lies in their classification strategies. The supervised CNN was specifically trained and tested as a multi-class classifier covering 15 different network traffic categories present in CIC-IDS2017. While previous studies on image-based packet representations, such as [3], mainly focused on binary classification, a multi-class approach was selected for the CNN. This allows for a more thorough assessment of its ability to not only detect anomalies but also to correctly identify specific and complex attack types.

In contrast, the AE was assessed within a strictly binary classification approach. As an unsupervised model trained solely on benign traffic, it effectively acts as an anomaly detector rather than a typical classifier. It identifies malicious traffic as out-of-distribution data based on a reconstruction error threshold. Therefore, the AE is inherently designed for binary detection, identifying deviations from normal behavior rather than performing detailed multi-class classification.

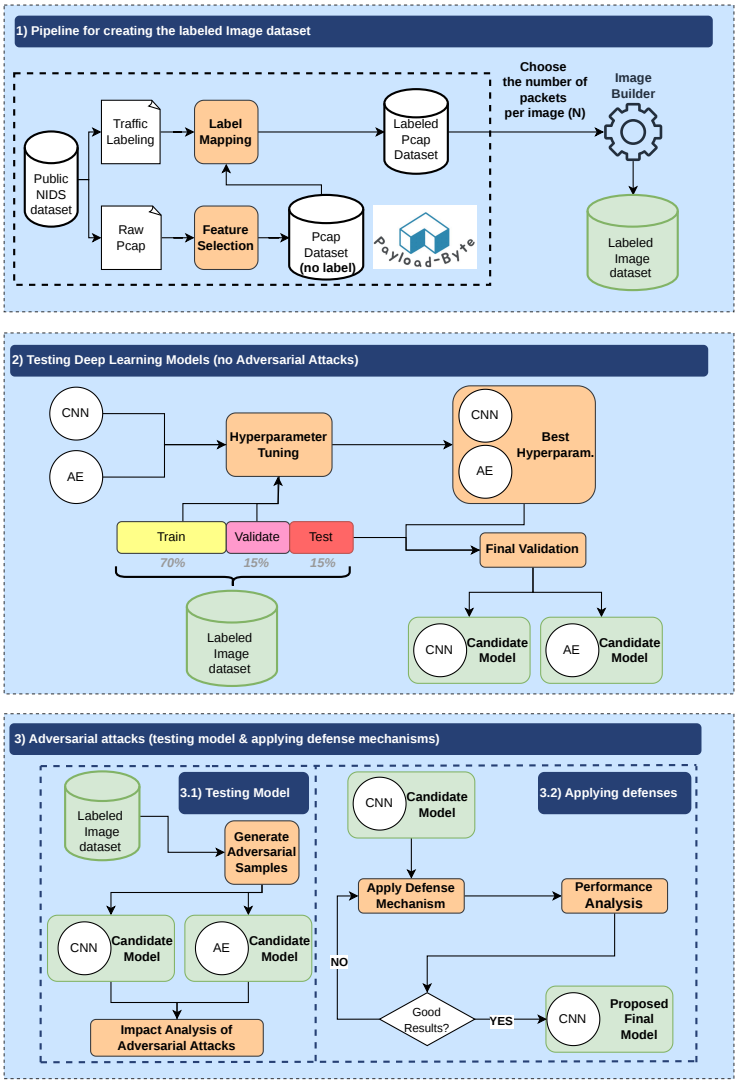


Figure 7.1: Full - Proposed Methodology

7.1. Creating the labeled Image dataset

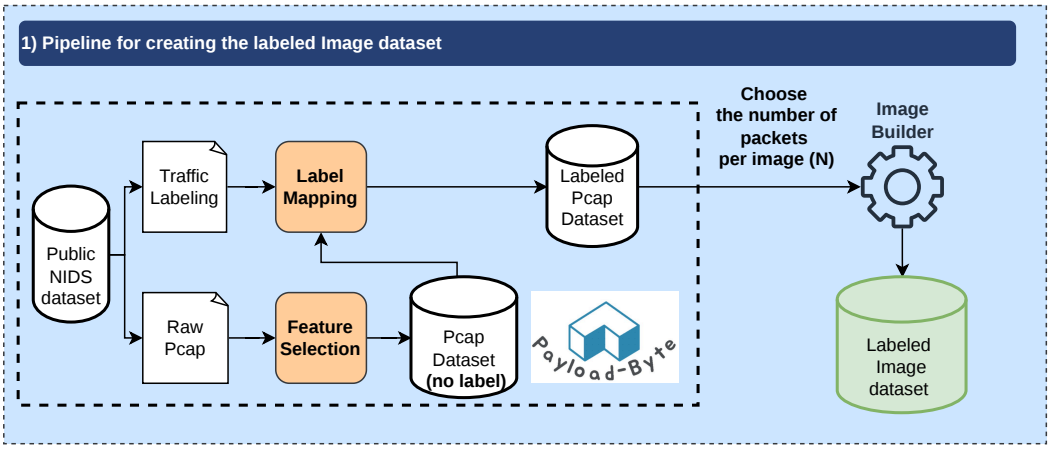


Figure 7.2: Stage 1 - Labeled Image dataset creation pipeline

To create the labeled image dataset, the process began by utilizing Payload-byte [101] and the CIC-IDS2017 dataset to produce a labeled pcap dataset. Payload-byte incorporates metadata and raw network packets from publicly available NIDS datasets like CIC-IDS2017 or UNSW-NB15 and attempts to assign labels to each individual pcap. To address the high occurrence of missing or inconsistent timestamps in the CIC-IDS2017 dataset, Payload-Byte uses a two-phase labeling system: first, it associates data based on a 5-tuple flow-ID <SrcIp, DstIp, SrcPort, DstPort, Protocol>, and then refines this using a temporal validation step that checks against documented attack durations. By removing benign traffic that overlaps with attack periods, the tool generates a conflict-free, standardized dataset by separating malicious packets from non-malicious ones that share the same flow ID [101]. This process yields labeled pcap data, with columns representing raw packet payloads and various headers [101]. Unlike [3], who also adopt a packet-based approach but only concentrate on TCP packets and overlook other protocols like UDP, the Payload-byte developers extracted some header features and respective payloads from various protocols (TCP, UDP, etc.). Payloads that are smaller than 1500 bytes are padded with zeros across all protocols to maintain consistency in the feature vector. After modifying the original Payload-byte source code, the feature vector shown in Figure 7.3 was generated, with the timestamp obtained from the CICIDS-2017 traffic labeling files to indicate the date and time of the first packet in a network flow [90], and T_delta denoting the difference in seconds between consecutive packets. The remaining features include header details from the Network and Transport layers, along with 1500 padded payload bytes [101].

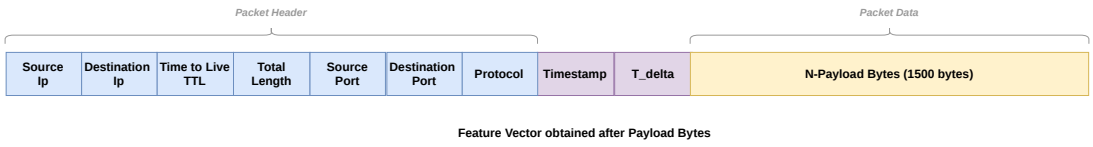


Figure 7.3: Feature vector obtained after using Payload-Byte[101]

As described by the authors in [3], analyzing single packets independently ignores the sequential and temporal relationships within a network flow, which can result in misclassifications, and increased false positives and false negatives. To address this, it is essential to incorporate these sequential and temporal relationships to give the model the necessary context for effective learning. To do this, the framework that converts sequential packets into a two-dimensional RGB image, originally proposed by [3], was adapted. Only the incremental image construction process and the use of color channels to encode

traffic direction, as in the original framework, were retained. The next paragraphs explain how the proposed image builder component differs from the one in [3], and Table 7.2 summarizes their similarities and differences.

Labeling the generated images is crucial because it enables the model to learn how to differentiate malicious images (group of packets within the same network session) from normal ones. In [3], this process involves assigning each generated image the overall label of that flow, which is defined as the 5-tuple <SrcIp, DstIp, SrcPort, DstPort, Protocol>. Using this approach, if a flow contains 10 packets and only the last 5 are part of a DDoS attack, with the flow's overall label being DDoS, then all images generated from this flow will be labeled DDoS, even if the attack began only in the last 5 packets. In the proposed approach, labeling is based on packet-level determinism, enabled by the labeled pcap dataset obtained from Payload-byte [101] and assigns each image the label of the last packet added to it. This ensures that "Mixed" images, which contain both benign and malicious traffic, are accurately labeled according to the network session state at the moment when the image was created.

In [3], a single group of traffic is defined by its 5-tuple flow ID. The proposed approach also utilizes this same 5-tuple flow ID but incorporates a 60-activity timeout. By examining the time difference between packets (using the timestamp feature present in each packet's feature vector, as shown in Fig. 7.3), an attempt is made to split long-lived communications into distinct network sessions.

Finally, the last major difference concerns the functioning of the triggering mechanism. This mechanism determines when an image gets generated, the image builder in [3] employs a sequential trigger that produces an image only at the beginning of a flow, from 1 packet up to P packets. In the suggested approach, an image is also generated with the first packet of a network session but incorporates a transition-based approach, creating an image whenever the labels of network sessions change. If the label shifts between consecutive packets within an image, it is considered as a 'Security event' and a new image is started from that specific packet.

The pseudocode for the proposed image building component algorithm is detailed in Algorithm 1. Figure 7.4 depicts the final feature vector representing a single network packet in the image dataset, while Figure 7.5 illustrates the incremental labeling image process generated from a sequence of packets, highlighting how label changes between the second and third packets result in different images.

Characteristic	Image builder [3]	Proposed Image Builder
Labeling Strategy	Flow	**Last Packet**
Packet Grouping	Flow	**Network Session**
Triggering Mechanism	Sequential	**Transition Based**
Image Building Process	Incremental	Incremental
Packet Directionality	RGB Channels	RGB Channels

Table 7.2: Comparison of my image builder and the image builder from [3]

Algorithm 1: Algorithm for the Image Building component

Input: Labeled pcap dataset D, Max packet per image N, Timeout $T = 60s$

Output: Image-based dataset (X, y)

`// Step 1: Bidirectional Flow Aggregation`

Standardize labels and normalize packet features to $[0, 255]$ range

Aggregate bidirectional flows by sorting $\{IP, Port\}$ pairs

Partition flows into sessions where inter-arrival time $\Delta t < T$

Sort packets within each $FlowID$ by arrival time

`// Step 2: Incremental RGB Image Synthesis`

foreach *session in* $D.group_by(FlowID)$ **do**

  Identify $Source \leftarrow$ session$[0].srcip$ `// First packet establishes Forward source`

  Identify $TriggerPoints$ (label changes)

  foreach $start$ *in* $TriggerPoints$ **do**

    for $n \leftarrow 1$ **to** $\min(\textit{size}(session) - start, N)$ **do**

      $I \leftarrow$ Initialize $N \times 1505$ RGB matrix with zero-padding

      for $p \leftarrow 0$ **to** $n - 1$ **do**

        $pkt \leftarrow session[start + p]$

        if $pkt.srcip = Source$ **then**

          $I[p, :, \text{Red}] \leftarrow pkt.pixels$ `// Forward Traffic`

        else

          $I[p, :, \text{Green}] \leftarrow pkt.pixels$ `// Backward Traffic`

        end

      end

      Append $(I, label_{start+n-1})$ to dataset

    end

  end

end

return (X, y)

Time to Live TTL	Total Length (L)	Total Length (H)	Protocol	T_delta	N-Payload Bytes (1500 bytes)

Feature Vector used in the Image based dataset

Figure 7.4: Feature vector used to represent a single packet in the Image dataset

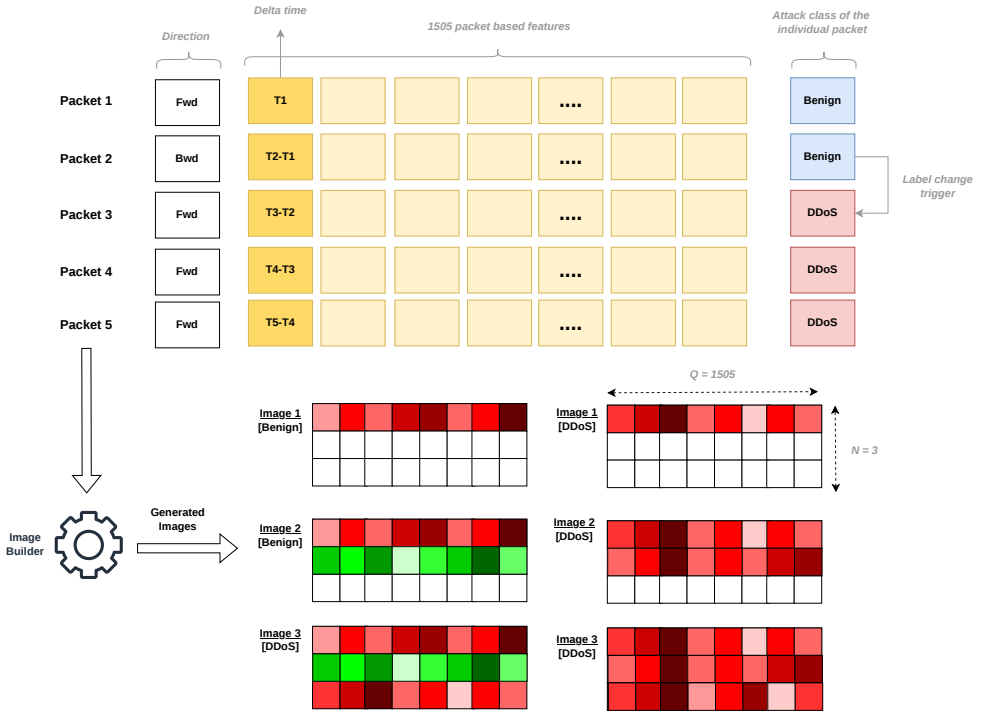


Figure 7.5: Example of Image generator process

7.2. Testing DL models

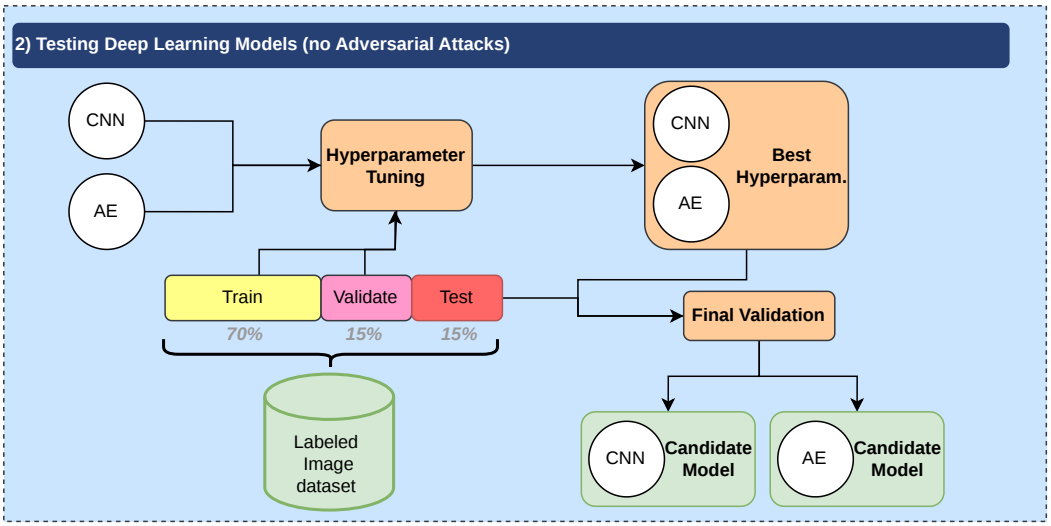


Figure 7.6: Stage 2 - Hyperparameter tuning pipeline

To begin testing DL models, two different architectures were selected: CNN and AE. The choice of CNN was driven by the image-based dataset, as it is well suited for image classification tasks. By leveraging convolutional filters, CNNs can capture spatial information and underlying patterns associated with network attacks, which other DL models might miss [102]. An AE was also chosen to evaluate whether, through image representation, it could learn to accurately reconstruct normal image data and potentially serve as an alternative to CNN, which is a more common model for this type of image classification task.

After defining the models, an approach was employed that involved hyperparameter tuning to identify optimal hyperparameter combinations for each model. Keras Tuner [103] and TensorFlow [104] were used for hyperparameter tuning, with TensorBoard [105] utilized to visualize the results. These packages facilitated the establishment of two hyperparameter tuning pipelines for each model. Several tuning strategies were evaluated, including RandomSearch, BayesianOptimization, and Hyperband. Random Search was ultimately chosen as the primary hyperparameter optimization approach, as comparative experiments demonstrated clear advantages over other methods. Both Bayesian Optimization and Hyperband frequently produced a pronounced gap between training and validation loss curves, suggesting the onset of early overfitting. In the case of Bayesian Optimization, the probabilistic surrogate model tends to overfit by iteratively focusing the search on patterns specific to the validation split, which introduces selection bias. Hyperband, on the other hand, utilizes a successive halving mechanism that evaluates many configurations for only a few epochs. This process often rewards hyper-responsive, high-capacity architectures that achieve rapid initial convergence but fail to generalize effectively over the course of full training cycles [103].

The Random Search tuner was set up with a maximum of 10 trials for both the CNN and AE pipelines. The supervised CNN training was limited to 10 epochs per trial. The main goal was to maximize the validation PR-AUC. To enable the accurate calculation of this multi-class PR-AUC, the data loader applied one-hot encoding to the target labels during preprocessing. Since the labels shifted from sparse integers to one-hot vectors, the model was adapted to use *categorical_crossentropy* [106] as the loss function. For each attack category, categorical accuracy, macro-averaged F1-Score, overall Precision-Recall AUC, and per-class recall and precision were evaluated. To optimize computational resources and mitigate overfitting, an early stopping mechanism was implemented to monitor the PR-AUC. If this metric did not improve over 3 consecutive epochs, the trial was stopped

early, and the best weights from that trial were restored. In contrast, the unsupervised AE pipeline needed a slightly adjusted tuning setup to fit its reconstruction-focused learning approach. The AE was allowed to train for up to 15 epochs. Since the main aim of the AE is to understand benign traffic patterns, the tuner focused on minimizing the validation loss. To evaluate the quality of the AEs reconstructions, the pipeline tracks three key metrics: MAE, MSE, and binary crossentropy loss [107]. Both MAE and MSE quantify the physical discrepancy between the original features and the reconstructed outputs. MAE captures the average magnitude of errors, providing a straightforward sense of the typical reconstruction gap, while MSE emphasizes larger errors by squaring each difference, making it more sensitive to substantial reconstruction failures [108]. By reducing these metrics, the model tries to set a clear baseline for normal traffic, so that any future malicious traffic, which produces higher reconstruction errors, can be identified as anomalies. An early stopping mechanism was also used for the AE, monitoring validation loss with the same patience of 3 epochs and restoring the best weights when stopping.

The labeled image dataset was prepared for training within the hyperparameter tuning pipeline by splitting it into Train, Validation, and Test sets with 70%, 15%, and 15% splits, respectively. The training split was used to effectively train the models to distinguish normal images from malicious ones. The validation split evaluated the model's performance during each trial based on the specific hyperparameters in use at that moment. The test split provided an unbiased estimate of the model's performance on unseen data [109]. Since different image dataset versions differ in the number of packets per image (N), and each value of N can result in a different distribution of attack classes, it was essential to establish a fair and consistent basis for comparison. To address this, a stratified splitting method was used to ensure that the original distribution of each attack class for each value of N was preserved exactly across the training, validation, and test splits.

The selected CNN architecture (see Table 7.3) aims to balance effective feature extraction from sequential network traffic with the hardware and protocol requirements of DL. The fixed layers create a consistent processing pipeline, starting with an input shape of $(N, 1505, 3)$ that preserves the sequence of N packets and uses the Red and Green channels to encode flow direction (Forward and Backward). A $1/255$ rescaling layer normalizes raw byte values for numerical stability in the Adam optimizer. A ZeroPadding2D layer expands the feature width to meet CuDNN memory alignment constraints, preventing hardware execution errors. Structural layers such as MaxPooling2D and the final Dense layer with Softmax activation are kept fixed to preserve spatial invariance and map

network traffic features to the 15 distinct attack categories. The hyperparameter search space systematically explores variations in the number of Conv2D filters (from 32 to 128) and Dense units (from 128 to 512) to find the optimal model capacity for accurately classifying diverse network behaviors. This process is crucial to ensure the network is complex enough to capture detailed, protocol-specific features, while remaining general enough to prevent overfitting. By tuning the learning rate alongside these architectural parameters, the process effectively navigates the high-dimensional loss landscape of packet data. The primary objective of the CNN pipeline is to achieve optimal detection performance across 15 distinct traffic categories, creating a robust and accurate baseline for network intrusion detection. This foundation will then be used to rigorously evaluate the model's resilience when subjected to adversarial attacks.

Table 7.3: CNN Architecture, Fixed Parameters, and Hyperparameter Search Space

Layer	Type	Fixed Parameters	Hyperparameter Values
1	`Input`	Shape: (N, 1505, 3)	–
2	`Rescaling`	Scale: 1/255	–
3	`ZeroPadding2D`	Padding: ((pad_top, pad_bottom), (0, 31))	–
4	`Conv2D`	Kernel: (3, 3), Padding: same	Filter: (min=32, max=128, step=32)
5	`MaxPooling2D`	Pool Size: (2, 2)	–
6	`Conv2D`	Kernel: (3, 3), Padding: same	Filter: (min=64, max=256, step=64)
7	`MaxPooling2D`	Pool Size: (2, 2)	–
8	`Flatten`	–	–
9	`Dense`	Activation: ReLU	Units: (min=128, max=512, step=128)
10	`Dense`	Units: 15, Act: Softmax	–
Compilation Settings			
–	`Adam Optimizer`	Loss: Categorical Crossentropy	Learning rate: [1e-2, 1e-3, 1e-4]

The AE architecture (see Table 7.5) is built to perform unsupervised anomaly detection by learning to reconstruct typical benign network traffic and spotting deviations as possible threats. It starts with an input shape of $(N, 1505, 3)$, maintaining directional flow encoding through the Red and Green channels and applying a $1/255$ rescaling layer for normalization. A ZeroPadding2D layer, similar to the CNN architecture, is used, with a matching Cropping2D layer at the output to restore original dimensions for accurate pixel-wise reconstruction error measurement. The hyperparameter search varies the number of filters, ranging from 64 to 128 in the outer layers and 16 to 32 in the inner layers, to identify the optimal compression bottleneck for feature extraction. This process tries to help in identifying a latent space that captures key protocol behaviors without giving the model enough capacity to reconstruct unseen malicious traffic accurately. By tuning the learning rate, filter sizes, and using Binary Crossentropy loss, the AE architecture aims to establish a reliable baseline of ”normal” behavior, and tries to make malicious traffic yield a much higher reconstruction error for threshold-based detection.

The hyperparameter tuning process remains consistent across both models and all versions of the image dataset, ensuring a fair and reliable comparison of their best results.

Table 7.5: AE Architecture, Fixed Parameters, and Hyperparameter Search Space

Layer	Type	Fixed Parameters	Hyperparameter Values
1	`Input`	Shape: (N, 1505, 3)	–
2	`Rescaling`	Scale: 1/255	–
3	`ZeroPadding2D`	Padding: ((pad_top, pad_bottom), (0, 31))	–
4	`Conv2D`	Kernel: (3, 3), Padding: same, Act: ReLU	Filters: (min=64, max=128, step=32)
5	`MaxPooling2D`	Pool Size: (2, 2)	–
6	`Conv2D`	Kernel: (3, 3), Padding: same, Act: ReLU	Filters: (min=16, max=32, step=16)
7	`MaxPooling2D`	Pool Size: (2, 2)	–
8	`UpSampling2D`	Size: (2, 2)	–
9	`Conv2D`	Kernel: (3, 3), Padding: same, Act: ReLU	Filters: (min=16, max=32, step=16)
10	`UpSampling2D`	Size: (2, 2)	–
11	`Conv2D`	Kernel: (3, 3), Padding: same, Act: ReLU	Filters: (min=64, max=128, step=32)
12	`Conv2D`	Filters: 3, Kernel: (3, 3), Act: Sigmoid	–
13	`Cropping2D`	Cropping: ((pad_top, pad_bottom), (0, 31))	–
Compilation Settings			
–	`Adam Optimizer`	Loss: Binary Crossentropy	Learning rate: [1e-2, 1e-3, 1e-4]

7.3. Adversarial Attacks + Defenses

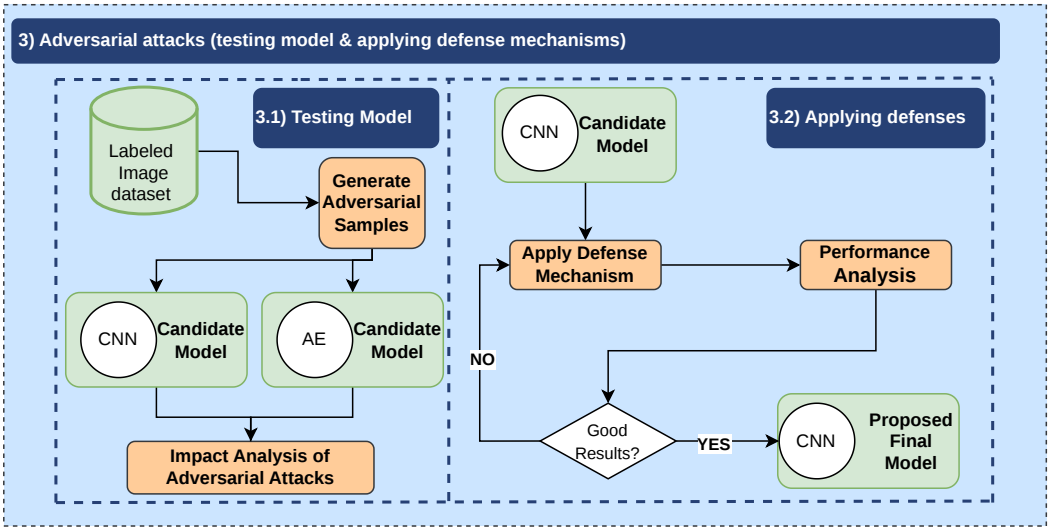


Figure 7.7: Stage 3 - Pipeline for adversarial attack and defense testing

Following hyperparameter tuning, the best models from each version of the labeled image dataset were selected for adversarial evaluation. To do this, the first step involved generating adversarial samples, which in the case of network packets is not as straightforward as in the image domain. In images, small perturbations can be added without disrupting the logic, since it is merely an image. However, in network traffic, random alterations or perturbations of packets are not feasible, as they could affect various factors, especially the maliciousness of the packet and the strict network protocols all packets must follow [2]. To address these limitations and ensure generated packets remain protocol-valid while maintaining original maliciousness, the perturbations described in Table 7.8 were applied.

Perturbation Type	Target Scope	Modification Summary
`Changing TTL`	All active packets	Increases the Time-to-Live (TTL) value of every active packet by a target constant, capped at a maximum value of 255.
`Changing delta time`	Last active packet	Increases the delta time (*t_delta*) of only the final active packet by a target constant, capped at a maximum value of 5.
`Adding payload data`	Last active packet	Inserts 40 bytes of dummy data at the end of the final packet's payload and updates the length header to account for the new data.

Table 7.8: Summary of the perturbations used to create adversarial samples from malicious images

To analyze the impact of adversarial attacks, adversarial samples were fed into each candidate model (AE and CNN) across different versions of the image dataset, and the evasion rate was checked. An evasion was considered successful only if the attack was classified as Benign; otherwise, it was deemed unsuccessful. For sample generation, malicious images from each specific dataset version were used to create an equal number of adversarial samples for each attack class and each listed perturbation in Table 7.8.

The initial evaluation led to the decision not to pursue AT for the AE model. Early results indicated that the model performed poorly against adversarial attacks, likely because these perturbations are subtle and localized. As a result, much of the input image remains unchanged, causing the AE to generate reconstruction errors that are not large enough to be detected as anomalies. Having this in mind, it was concluded that AT would not resolve this fundamental limitation in the model's design. Consequently, further investment of time and resources was considered unlikely to yield significant improvements in performance (see Section 8.3 for details).

AT was selected as the defense method. For each dataset version, specific adversarial samples were generated and incorporated into the retraining process of the top-performing CNN model, while maintaining the same hyperparameters across all versions. After retraining with these adversarial samples, two main aspects were evaluated: the evasion rate after AT and the impact of AT on the models' predictions on normal, clean data. Once an satisfactory balance between robustness against adversarial samples and accuracy on clean data was attained, the adversarial defense process was concluded.

7.3.1 Adversary assumptions

This study evaluates adversarial scenarios based on the following assumptions about the adversary's capabilities and objectives:

1. **Adversary Knowledge**: The adversary is assumed to operate in a Black-box scenario, meaning they have no internal knowledge of the NIDS model or its training data. All perturbations in this work are designed to function without requiring such privileged information.
2. **Perturbation Constraints**: The adversary may only apply the specific perturbations detailed in Table 7.8. Each modification is strictly limited by predefined upper and lower bounds, ensuring that all altered packets remain valid within protocol specifications.

3. **Adversary Goal**: The adversary's primary objective is to evade detection by manipulating malicious network traffic so that the model incorrectly classifies it as benign.

4. **Attack Surface**: The attack surface is strictly confined to the manipulation of packet-based features.

5. **Simulation Environment**: All adversarial capabilities were assessed in a strictly offline environment. This threat model does not account for live network validation or real-time packet injection.

8. Discussion & Results

In this chapter, the results obtained from each step described in the previous chapter are detailed and discussed. The hyperparameter tuning was performed using hardware comprising an Intel(R) Xeon(R) Gold 6252 CPU at 2.10GHz with 24 cores, 1TB RAM, and 4 Tesla V100-SXM2-32GB GPUs. The testing setup, which evaluated the model on the test split and against adversarial attacks both before and after AT, was identical to the hyperparameter tuning setup. However, only one GPU was used during testing to simulate a more resource-constrained environment.

8.1. Labeled Image dataset

The attack class distribution for the different versions of the image dataset can be seen in Table 8.1. It clearly shows the growth of each attack type across the dataset versions, except for the 'SQL Injection' attack class, which appears to plateau at 53 samples when $N = 6$. As N increases, more samples for each attack class are generated, providing additional data for the final model to better distinguish between classes and potentially detect more sophisticated attackers attempting to evade detection with adversarial attacks. Additionally, a side effect of increasing N is that the overall size of the image dataset also naturaly grows.

Extreme minority classes such as SQL Injection were retained to ensure the pipeline could detect critical, low-frequency attacks. For the supervised CNN, the risk of majority-class bias was mitigated by explicitly optimizing for PR-AUC and evaluating with Macro-F1, compelling the network to penalize rare and common events equally and delineate accurate decision boundaries for high-risk threats. Conversely, the unsupervised AE exploits the significant class imbalance by leveraging the overwhelming volume of 'BENIGN' traffic to establish a robust baseline of normal network behavior, subsequently ignoring rare attacks during training to enable their accurate identification as out-of-distribution anomalies during inference.

Image Dataset Version	Attack class distribution														
	BENIGN	DoS Hulk	DDoS	FTP-Patator	DoS Golden.	SSH-Patator	DoS slowloris	DoS Slowhttpt.	Heartbleed	Bot	Web-XSS	Web-Brute	PortScan	Infiltration	SQL Inj.
N=3	4,016,226	591,985	430,083	66,464	21,799	11,613	10,414	10,283	3,559	2,205	1,035	963	542	46	37
N=4	4,611,325	788,294	553,596	89,146	26,377	15,467	12,386	12,372	4,860	2,933	1,366	1,270	653	61	50
N=5	5,099,444	980,886	662,319	111,202	29,003	19,296	14,240	14,166	6,161	3,048	1,713	1,522	718	76	52
N=6	5,567,786	1,170,389	760,001	132,514	31,336	23,100	16,086	15,819	7,464	3,163	2,047	1,752	781	91	53
N=7	5,997,903	1,353,268	829,400	153,367	32,512	26,869	17,926	17,407	8,787	3,208	2,372	1,984	843	106	53
N=8	6,410,232	1,533,127	891,909	173,583	33,642	30,653	19,761	18,941	10,128	3,253	2,718	2,221	906	120	53

Table 8.1: Distribution of attack classes of the different versions of the image dataset.

8.2. Testing DL models

An evaluation of the candidate DL model´s performance prior to exposure to adversarial attacks is presented in this section. To ensure a thorough and organized analysis, it is divided into two parts: one for the CNN in Section 8.2.1 and one for the AE in Section 8.2.2. Each part describes the hyperparameter tuning process specific to that model across all image dataset versions ($N = 3$ to $N = 8$). Detailed tables in these sections show the hyperparameter settings tested during each trial, along with figures demonstrating training and validation metrics over epochs. These plots mainly offer a wider perspective on the models' training processes, showing their stability and convergence during the tuning phase.

8.2.1 CNN

As previously mentioned, to identify the best CNN configuration, the hyperparameter tuning involved 10 separate trials per image dataset version. The configurations tested are summarized in Table 8.2. For clarity, each cell shows hyperparameter values as a 4-tuple (x, y, z, w), formatted as follows:

- x indicates the number of filters for the first Conv2D layer (Layer 4).
- y indicates the number of filters for the second Conv2D layer (Layer 6).
- z represents the number of units in the fully connected Dense layer (Layer 9).
- w denotes the learning rate used by the Adam optimizer.

To clearly indicate the most effective configurations, the top-performing hyperparameters for each dataset version are highlighted in bold in Table 8.2. For a detailed overview of the fixed and variable architectural components of the CNN, refer to Table 7.3.

Figure 8.1 illustrates the CNN's learning behavior during hyperparameter tuning, displaying Categorical Accuracy, F1 Macro score, Categorical Crossentropy Loss, and PR AUC over 10 epochs. The plots reveal that most trials display consistent, stable convergence across all dataset versions. Notably, the Categorical Accuracy, F1 Macro score, and PR AUC initially increase sharply and then level off, while the loss decreases correspondingly. This indicates that the configurations in each trial effectively learned to distinguish the 15 network traffic categories. However, some outliers are evident as flat lines, indicating hyperparameter combinations where the model did not converge. Additionally, early stopping is confirmed by some trial curves ending before 10 epochs, showing training was halted when the validation PR-AUC stopped improving to prevent overfitting.

After identifying the most effective hyperparameter configurations, the leading model for each dataset version was evaluated on the unseen test split. Table 8.3 presents detailed multi-class classification metrics, such as Precision, Recall, F1-Score, and Support for all 15 network traffic categories, as well as overall accuracy and macro and weighted averages for Precision, Recall, and F1-Score. A closer examination of the overall performance indicates that the CNN consistently achieves approximately 92% accuracy across all versions of the image dataset. Nevertheless, as discussed in chapter 5, relying solely on overall accuracy can be deceptive for NIDS. For example, with nearly 1 million instances at $N = 8$, over 960,000 are benign. This allows a model to easily attain 90% accuracy simply by classifying all packets as normal, which undermines its security effectiveness. To accurately evaluate the model's threat detection capabilities, it is crucial to look beyond accuracy and consider metrics like Precision, Recall, and F1-Score, as these better reveal issues related to misclassifying minority classes (refer to chapter 5).

By emphasizing the macro averages of the F1-Score and Recall, all 15 classes are treated equally, preventing benign traffic from skewing performance metrics and demonstrating the model's ability to detect rare attacks. The weighted average also offers valuable insight into the model's stability in real-world conditions. Since the weighted average considers the actual data distribution (the "Support"), it indicates how the model will perform on a day-to-day basis where benign traffic dominates. As shown in Table 8.3, the CNN achieves a strong Weighted F1-Score of around 0.92 to 0.93 across all versions of N.

This confirms that while the model effectively detects minority attacks, it does so without compromising its ability to handle normal traffic, remaining reliable and not generating excessive false positives.

As these metrics are tracked, a clear and favorable trend emerges as the number of packets per image (N) increases. Although the overall accuracy remains relatively stable (varying only slightly between 0.9195 and 0.9266), the Macro averages reveal substantial gains in the model's baseline detection capability. In particular, the Macro Recall improves markedly from 0.7667 at $N = 3$ to 0.8620 at $N = 8$. Importantly, this improvement in threat detection does not come at the expense of the model's exactness: the Macro Precision remains highly stable throughout, increasing only marginally from 0.8063 at $N = 3$ to 0.8188 at $N = 8$. Consequently, because the model maintains its precision while significantly enhancing its recall, the Macro F1-Score follows a strong upward trajectory, rising from 0.7549 at $N = 3$ to 0.8272 at $N = 8$.

This upward trend indicates promising results for the proposed transition-based image generation method. As N increases, the CNN processes images representing a longer sequence of packets, offering the model a much wider temporal context of the network session. This broader sequential view enables the network to learn and detect subtle, prolonged behaviors of complex multi-packet attacks, such as persistent DoS variations or Web-based attacks, which would be nearly indistinguishable from normal traffic if only a small, 3-packet window is considered.

	Image Dataset Version					
Trial	**N=3**	**N=4**	**N=5**	**N=6**	**N=7**	**N=8**
1	(32,128,512,1e-4)	(96,64,512,1e-4)	(96,192,128,1e-4)	(96,64,384,1e-2)	(64,192,384,1e-2)	(96,128,512,1e-2)
2	(96,192,512,1e-2)	(64,192,384,1e-2)	(32,256,128,1e-4)	(64,128,512,1e-4)	(128,128,128,1e-4)	(128,64,384,1e-2)
3	(64,64,128,1e-4)	(128,128,128,1e-4)	**(32,128,256,1e-3)**	(128,128,256,1e-2)	(64,128,512,1e-3)	(32,192,256,1e-4)
4	(96,64,384,1e-2)	(64,128,512,1e-3)	(128,64,128,1e-4)	(64,256,256,1e-2)	(96,256,256,1e-4)	(32,64,384,1e-4)
5	(64,64,384,1e-4)	(96,256,256,1e-4)	(64,256,512,1e-2)	(32,128,384,1e-3)	**(64,128,384,1e-3)**	(64,192,384,1e-2)
6	(96,192,512,1e-4)	**(64,128,384,1e-3)**	(64,64,128,1e-3)	**(96,192,384,1e-3)**	(128,64,512,1e-4)	(96,128,128,1e-4)
7	**(64,192,384,1e-3)**	(128,64,512,1e-4)	(128,192,128,1e-4)	(128,128,384,1e-4)	(64,256,512,1e-2)	(32,192,384,1e-2)
8	(32,256,512,1e-3)	(64,256,512,1e-2)	(32,192,256,1e-4)	(64,128,256,1e-3)	(128,128,512,1e-2)	**(32,64,384,1e-3)**
9	(128,256,256,1e-3)	(128,128,512,1e-2)	(128,128,384,1e-4)	(32,128,512,1e-4)	(64,64,256,1e-2)	(96,128,256,1e-2)
10	(64,128,256,1e-2)	(64,64,256,1e-2)	(32,64,384,1e-3)	(32,192,128,1e-3)	(96,64,128,1e-3)	(32,64,512,1e-4)

Table 8.2: CNN hyperparameter configurations for each trial across multiple dataset versions.

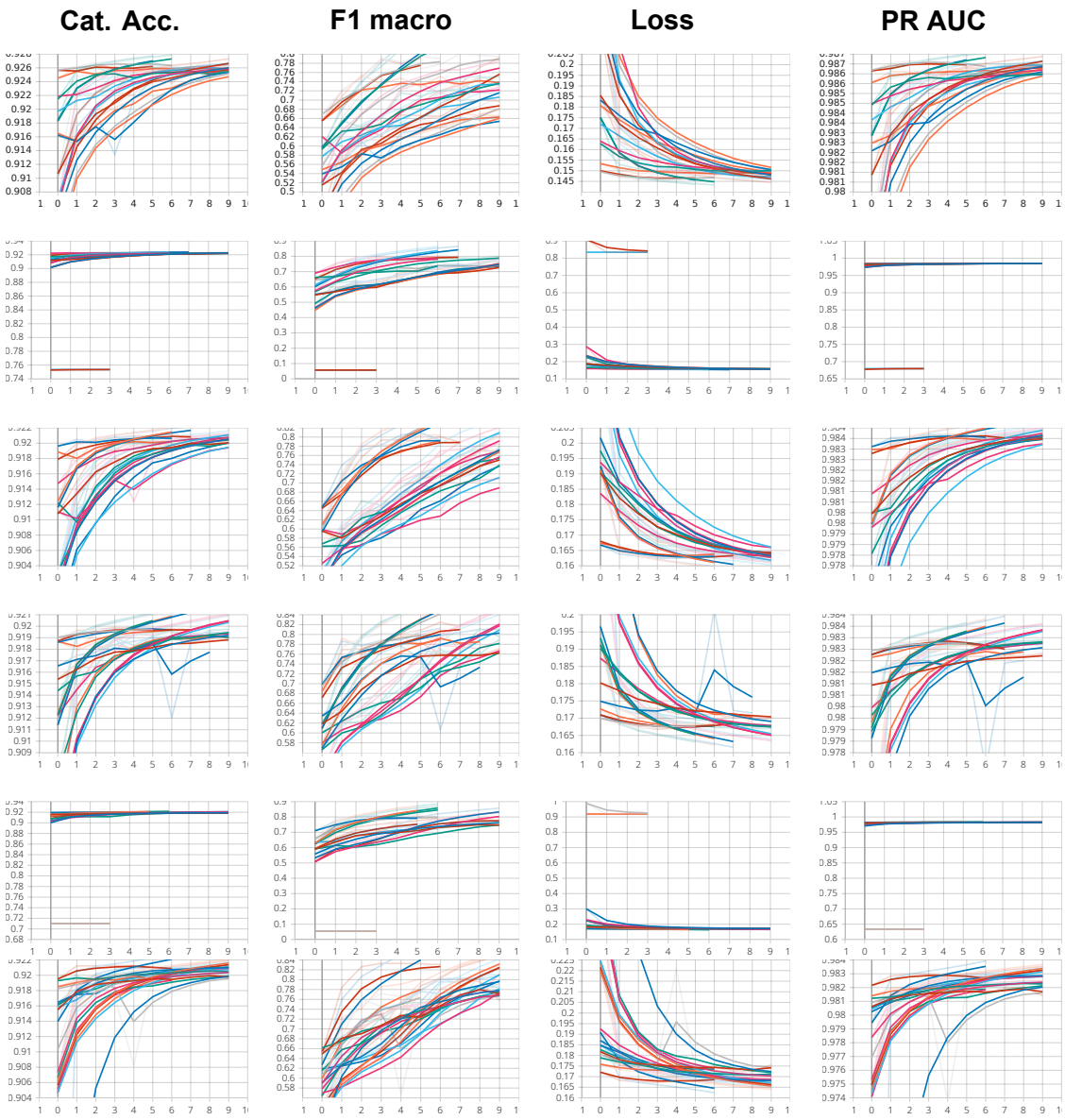


Figure 8.1: Training and validation performance metrics over 10 epochs for the CNN model hyperparameter tuning. Columns from left to right display Categorical Accuracy, F1 Macro score, Categorical Crossentropy Loss, and PR-AUC. Rows correspond to successive dataset versions, beginning with $N = 3$ and ending with $N = 8$.

Table 8.3: Detailed classification metrics for each class and overall performance from the best CNN models across the different dataset versions.

Version	Metric	BENIGN	Bot	DDoS	DoS Gold.	DoS Hulk	DoS Slow.	DoS slowl.	FTP-Pat.	Heartbl.	Infil.	PortScan	SSH-Pat.	Web-Brute	SQL Inj.	Web-XSS
N=3	Precision	0.9900	1.0000	0.6502	0.9570	0.8803	0.7267	0.8864	0.6511	0.6116	1.0000	0.6667	0.7809	0.7841	1.0000	0.5098
	Recall	0.9148	0.9577	0.9318	0.9927	0.9937	0.9806	0.9994	0.9978	0.8315	0.4286	0.1728	0.8840	0.4792	0.6000	0.3355
	F1-Score	0.9509	0.9784	0.7659	0.9745	0.9336	0.8348	0.9395	0.7880	0.7048	0.6000	0.2745	0.8293	0.5948	0.7500	0.4047
	Support	602434	331	64513	3270	88798	1543	1562	9970	534	7	81	1742	144	5	155
	Overall	**Accuracy:** 0.9266 | **Macro Avg (P/R/F1):** 0.8063 / 0.7667 / 0.7549 | **Weighted Avg:** 0.9431 / 0.9266 / 0.9306														
N=4	Precision	0.9969	1.0000	0.6481	0.9721	0.8847	0.7223	0.9408	0.6624	0.6063	1.0000	0.5806	0.8385	0.6860	1.0000	0.5000
	Recall	0.9009	0.9500	0.9924	0.9874	0.9991	0.9752	0.9327	0.9975	0.5048	0.3333	0.3673	0.8328	0.7474	1.0000	0.3171
	F1-Score	0.9464	0.9744	0.7841	0.9797	0.9384	0.8299	0.9368	0.7961	0.5509	0.5000	0.4500	0.8356	0.7154	1.0000	0.3881
	Support	691699	440	83040	3957	118244	1856	1858	13372	729	9	98	2320	190	7	205
	Overall	**Accuracy:** 0.9231 | **Macro Avg (P/R/F1):** 0.8026 / 0.7892 / 0.7751 | **Weighted Avg:** 0.9443 / 0.9231 / 0.9276														
N=5	Precision	0.9942	0.9977	0.6536	0.9621	0.8931	0.7572	0.9418	0.6711	0.6553	1.0000	0.8148	0.8345	0.7000	0.4286	0.4986
	Recall	0.8963	0.9562	0.9848	0.9862	0.9939	0.8894	0.9551	0.9986	0.5595	0.7500	0.2037	0.8189	0.4585	0.4286	0.7121
	F1-Score	0.9427	0.9765	0.7857	0.9740	0.9408	0.8180	0.9484	0.8027	0.6036	0.8571	0.3259	0.8266	0.5541	0.4286	0.5865
	Support	764916	457	99348	4350	147133	2125	2136	16681	924	12	108	2894	229	7	257
	Overall	**Accuracy:** 0.9199 | **Macro Avg (P/R/F1):** 0.7868 / 0.7728 / 0.7581 | **Weighted Avg:** 0.9406 / 0.9199 / 0.9243														
N=6	Precision	0.9954	0.9979	0.6618	0.9654	0.8944	0.7515	0.9181	0.6808	0.6394	1.0000	0.8654	0.8401	0.6424	0.5455	0.4802
	Recall	0.8925	0.9937	0.9871	0.9851	0.9988	0.8449	0.9805	0.9780	0.7964	0.6154	0.3846	0.8540	0.7376	0.7500	0.8697
	F1-Score	0.9411	0.9958	0.7924	0.9752	0.9437	0.7955	0.9483	0.8028	0.7093	0.7619	0.5325	0.8470	0.6867	0.6316	0.6188
	Support	835168	474	114000	4701	175559	2373	2413	19877	1120	13	117	3465	263	8	307
	Overall	**Accuracy:** 0.9195 | **Macro Avg (P/R/F1):** 0.7919 / 0.8446 / 0.7988 | **Weighted Avg:** 0.9401 / 0.9195 / 0.9237														
N=7	Precision	0.9927	0.9979	0.6713	0.9683	0.8989	0.7412	0.9333	0.6887	0.6458	1.0000	0.7101	0.8446	0.6940	1.0000	0.5351
	Recall	0.8947	0.9771	0.9704	0.9889	0.9981	0.9226	0.9736	0.9727	0.9173	0.9375	0.3889	0.8695	0.5421	0.7500	0.6854
	F1-Score	0.9411	0.9874	0.7936	0.9785	0.9459	0.8220	0.9530	0.8064	0.7580	0.9677	0.5026	0.8569	0.6087	0.8571	0.6010
	Support	899686	481	124410	4877	202990	2611	2689	23005	1318	16	126	4031	297	8	356
	Overall	**Accuracy:** 0.9205 | **Macro Avg (P/R/F1):** 0.8215 / 0.8526 / 0.8253 | **Weighted Avg:** 0.9388 / 0.9205 / 0.9242														
N=8	Precision	0.9928	1.0000	0.6731	0.9669	0.9069	0.7359	0.9367	0.6876	0.6448	1.0000	0.8158	0.8777	0.6820	0.8571	0.5044
	Recall	0.8943	0.9713	0.9815	0.9883	0.9938	0.9444	0.9733	0.9680	0.8986	0.8333	0.4559	0.8149	0.6246	0.7500	0.8378
	F1-Score	0.9409	0.9854	0.7986	0.9775	0.9484	0.8272	0.9547	0.8041	0.7508	0.9091	0.5849	0.8452	0.6520	0.8000	0.6297
	Support	961535	488	133787	5047	229969	2841	2964	26038	1519	18	136	4598	333	8	407
	Overall	**Accuracy:** 0.9212 | **Macro Avg (P/R/F1):** 0.8188 / 0.8620 / 0.8272 | **Weighted Avg:** 0.9396 / 0.9212 / 0.9249														

8.2.2 AE

Similar to the supervised approach, the hyperparameter tuning process for the AE involved executing 10 trials for each version of the image dataset. The specific configurations explored during these tuning sessions are detailed in Table 7.5. For conciseness, the hyperparameter values in each cell are presented as a 3-tuple (x,y,w) and are formatted as follows:

- x indicates the number of filters for the outer layers in both the encoder and decoder (layers 4 and 11).
- y indicates the number of filters for the inner layers surrounding the bottleneck/latent space (layers 6 and 9).
- w denotes the learning rate used by the Adam optimizer.

The most effective hyperparameter configurations are also highlighted in bold in Table 8.4. For additional details on the fixed and variable architectural components of the AE, refer to Table 7.5.

Figure 8.2 illustrates the training process of the AE during hyperparameter tuning. Unlike the CNN, the AE is an unsupervised model trained only on benign network traffic. Consequently, instead of multi-class classification metrics, the primary metrics tracked across epochs are reconstruction losses such as Binary Crossentropy, MAE, and MSE. The plots show a general trend of rapid, steep decreases in all three error metrics during the initial epochs across different dataset versions. However, similarly to what was observed with the CNN, there are a few notable outliers representing trials that failed to converge. Following this initial decline, the learning curves tend to stabilize and flatten, indicating that the models have potentially learned to compress and reconstruct benign traffic patterns. Additionally, the early termination of several trial lines before reaching the 15-epoch limit visually confirms the early stopping mechanism, which successfully halted training when validation loss stopped improving, preventing over-compression or memorization of the training data.

After selecting the most effective hyperparameter configurations, the top AE candidate for each dataset version were tested on the unseen test split, with detailed metrics for this evaluation provided in Table 8.5. Since the AE functions as an anomaly detector, it does not directly produce class probabilities. Instead, it uses a reconstruction error threshold:

instances with a reconstruction loss above this threshold are marked as anomalous (attacks), while those below are considered normal (benign). Table 8.5 displays the binary classification metrics Precision, Recall, F1-Score, and Support that result from applying this anomaly detection threshold, showcasing the model's baseline ability to differentiate benign traffic from malicious traffic.

A closer look at the binary classification results in Table 8.5 reveals a distinct difference from the outcomes with the supervised CNN. The overall accuracy of the AE starts at approximately 0.7176 for $N = 3$ but gradually decreases as the number of packets per image increases, falling to 0.6341 at $N = 8$. Nonetheless, as previously noted, overall accuracy can be misleading when assessing NIDS performance because the predominance of benign traffic may mask deficiencies in detecting malicious activity. Therefore, analyzing detailed metrics, particularly the Macro F1-Score and individual metrics for the 'ATTACK' class, uncovers significant detection failures. For $N = 3$, the AE exhibits a low Recall of 0.0812 and an F1-Score of 0.1135 for malicious traffic. These metrics decline further as N increases, reaching a low Recall of 0.0077 and an F1-Score of 0.0123 at $N = 8$, which indicates the AE largely fails to detect malicious instances and often misclassifies them as normal traffic, leading to a high rate of False Negatives.

This clear downward trend highlights a key limitation of using this AE architecture for transition-based image datasets, as the poor performance is linked to how the unsupervised reconstruction error functions. In an image with N packets, a malicious attack might appear in just a single packet or a small part of the sequence. As N increases, the fraction of malicious data compared to normal data in the image drops significantly. Since the AE computes a single reconstruction error for the entire image, the small anomaly caused by the malicious packet becomes "diluted" among the larger amount of normal traffic. As a result, the localized malicious data doesn't cause enough disruption to surpass the anomaly detection threshold, causing the model to mistakenly classify the sample as benign.

	Image Dataset Version					
Trial	**N=3**	**N=4**	**N=5**	**N=6**	**N=7**	**N=8**
1	(128,32,1e-3)	(64,32,1e-4)	(96,16,1e-3)	(96,16,1e-4)	(96,16,1e-4)	(128,16,1e-3)
2	(128,16,1e-3)	(128,32,1e-2)	**(128,32,1e-3)**	(64,16,1e-2)	(96,32,1e-2)	(64,16,1e-4)
3	(128,16,1e-2)	(128,16,1e-2)	(128,16,1e-4)	(96,32,1e-2)	(128,16,1e-4)	**(96,32,1e-3)**
4	(128,32,1e-4)	**(64,32,1e-3)**	(64,16,1e-2)	(64,32,1e-4)	(64,32,1e-4)	(96,32,1e-2)
5	(96,16,1e-4)	(96,16,1e-2)	(128,32,1e-4)	(96,16,1e-3)	(64,32,1e-2)	(96,16,1e-2)
6	(64,16,1e-3)	(128,16,1e-4)	(96,16,1e-4)	(128,16,1e-3)	**(96,32,1e-3)**	(128,16,1e-4)
7	(128,32,1e-2)	(128,32,1e-4)	(64,16,1e-3)	(128,32,1e-4)	(128,16,1e-2)	(128,16,1e-2)
8	**(96,32,1e-3)**	(64,16,1e-3)	(96,32,1e-4)	(128,32,1e-2)	(128,32,1e-2)	(128,32,1e-4)
9	(64,16,1e-2)	(64,16,1e-2)	(96,32,1e-2)	(64,16,1e-3)	(96,16,1e-3)	(96,32,1e-4)
10	(96,16,1e-2)	(96,32,1e-3)	(64,32,1e-4)	**(128,32,1e-3)**	(96,16,1e-2)	(64,32,1e-4)

Table 8.4: AE hyperparameter configurations for each trial across multiple dataset versions.

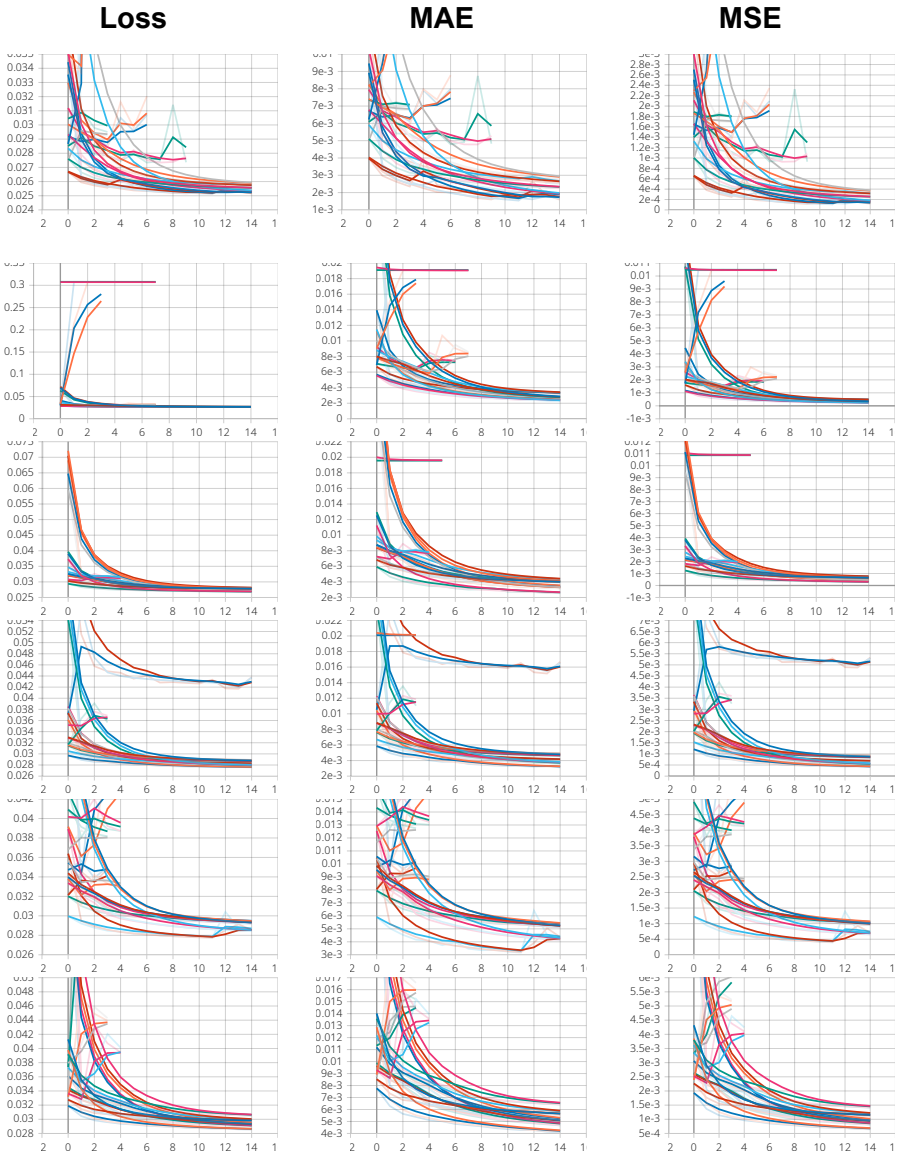


Figure 8.2: Training and validation performance metrics over 15 epochs for the AE model hyperparameter tuning. Columns from left to right display Binary crossentropy loss, MAE, and MSE. Rows correspond to successive dataset versions, beginning with $N = 3$ and ending with $N = 8$.

Version	Metric	BENIGN (0)	ATTACK (1)
N=3	Precision	0.7736	0.1887
	Recall	0.9000	0.0812
	F1-Score	0.8321	0.1135
	Support	602434	172655
	Overall	**Acc:** 0.7176 | **Macro F1:** 0.4728 | **Weighted F1:** 0.6720	
N=4	Precision	0.7434	0.1417
	Recall	0.9000	0.0505
	F1-Score	0.8142	0.0744
	Support	691699	226325
	Overall	**Acc:** 0.6906 | **Macro F1:** 0.4443 | **Weighted F1:** 0.6318	
N=5	Precision	0.7195	0.0980
	Recall	0.9000	0.0300
	F1-Score	0.7997	0.0460
	Support	764916	276661
	Overall	**Acc:** 0.6689 | **Macro F1:** 0.4228 | **Weighted F1:** 0.5995	
N=6	Precision	0.7020	0.0634
	Recall	0.9000	0.0174
	F1-Score	0.7888	0.0273
	Support	835168	324690
	Overall	**Acc:** 0.6529 | **Macro F1:** 0.4080 | **Weighted F1:** 0.5756	
N=7	Precision	0.6906	0.0466
	Recall	0.9000	0.0120
	F1-Score	0.7815	0.0191
	Support	899686	367215
	Overall	**Acc:** 0.6426 | **Macro F1:** 0.4003 | **Weighted F1:** 0.5605	
N=8	Precision	0.6812	0.0315
	Recall	0.9000	0.0077
	F1-Score	0.7755	0.0123
	Support	961535	408153
	Overall	**Acc:** 0.6341 | **Macro F1:** 0.3939 | **Weighted F1:** 0.5481	

Table 8.5: Detailed binary classification metrics and overall performance from the best AE models across different dataset versions.

8.3. Adversarial Robustness (before and after AT)

The following section represents the final evaluation step outlined in the proposed methodology (see chapter 7). It aims to test the candidate models obtained after hyperparameter tuning against adversarial samples, which are generated based on the perturbations described in Table 7.8. After analyzing and discussing the results, the next step is to apply AT to determine whether the models robustness has improved and to assess the impact of AT on predictions made on clean data. These three phases are divided into the three subsections bellow.

8.3.1 Robustness before AT

To assess the robustness of each candidate model across different dataset versions, an equal number of adversarial samples, generated using the previously discussed perturbations. Evasion rates are illustrated with line charts, with Figure 8.3 showing results for all CNN candidate models and Figure 8.4 for all AE models. The evasion rates in the line charts clearly demonstrate that CNN robustness improves with increasing N, indicating that images with more packets are more resistant to adversarial attacks. Conversely, for the AE, the evasion rate does not increase with N, indicating an opposite trend. Similar to the poor performance observed on the test split, this issue likely arises because, as the number of packets per image increases, the proportion of perturbed data within the benign sequence decreases. As a result, localized adversarial perturbations become less apparent and fail to generate enough overall reconstruction error to surpass the anomaly threshold. This leads the model to misclassify the sample as benign traffic. Given this, and since the AE model is trained only on benign traffic, AT was applied exclusively to the CNN.

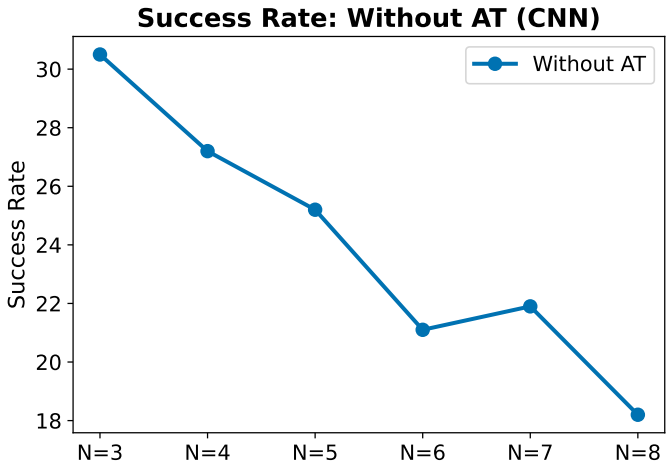


Figure 8.3: Evasion rate of adversarial attacks against the candidate CNN models across different dataset versions.

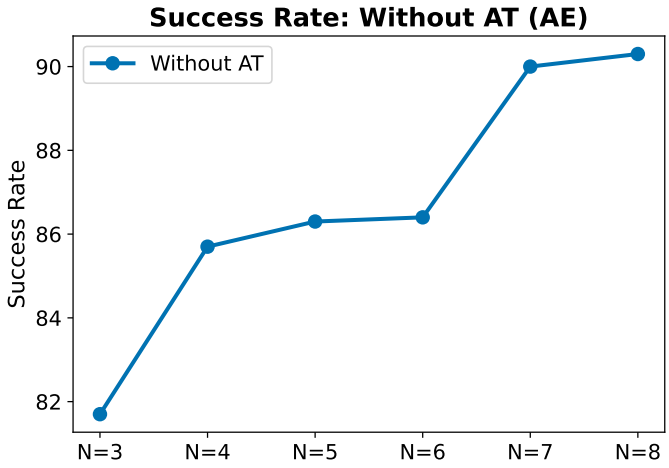


Figure 8.4: Evasion rate of adversarial attacks against the candidate AE models across different dataset versions.

8.3.2 Robustness after AT

To thoroughly assess the impact of AT on CNN, the retrained candidate models were tested against a new set of adversarial attacks. It's crucial to note that the adversarial samples used for testing differed entirely from those used during retraining. During AT, models were retrained with samples created using static, minimal perturbations, such as increasing the TTL or delta time by exactly 1. However, to properly evaluate the models' actual ability to generalize against dynamic threats, the evasion rates in this section were calculated using adversarial samples generated with pseudo-random perturbation values up to the maximum thresholds specified in Table 7.8. This approach ensures that the network learned the fundamental structural patterns of the attacks instead of merely memorizing specific, static adversarial signatures.

Figure 8.5 shows the updated evasion rates for CNN models with AT across all image dataset versions from $N = 3$ to $N = 8$. Compared with the pre-AT baseline in Figure 8.3, these results clearly demonstrate enhanced adversarial robustness. Before applying the defense, CNNs were vulnerable to adversarial attacks, with evasion success rates between 18.2% and 30.5%. After implementing AT, the success rate decreased significantly, with evasion rates falling to approximately 1.4% to 3.1% across all dataset versions, as shown in Figure 8.5. As a result, the percentage of failed evasions, where the model correctly identified malicious payloads, increased to over 96.9%, indicating that the model detects attacks most of the time. These results demonstrate the effectiveness of AT against the applied adversarial perturbations in this image-based architecture. The CNN

not only reduced vulnerabilities caused by TTL, delta time, and payload padding modifications but also showed strong generalization under randomized testing perturbations, resisting various attack intensities.

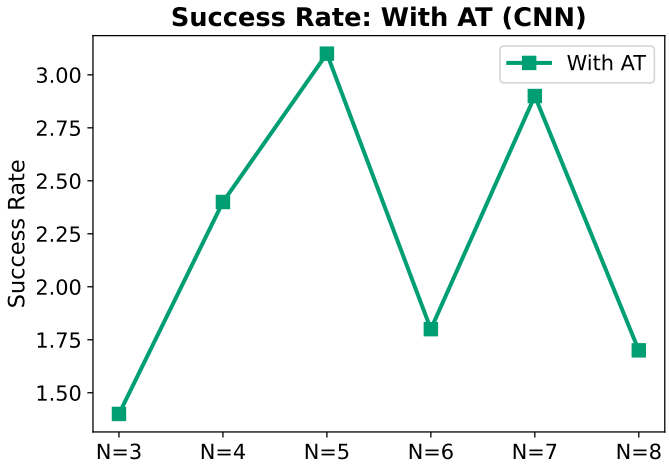


Figure 8.5: Evasion rate of adversarial attacks against the candidate CNN models (with AT) across different dataset versions.

8.3.3 Impact of AT on clean data

While AT effectively shields the CNN from evasion attacks, it's important to examine how this defense affects the model's performance on standard, unperturbed network traffic. In many ML fields, enhancing adversarial robustness often results in reduced baseline accuracy, a trade-off known as the "robustness cost" [110]. To thoroughly analyze this effect the CNN models (both before and after AT) were evaluated on the same clean test set used to make Table 8.3. To offer a more detailed view of the model's operation beyond basic classification metrics, three particular error categories were defined and monitored across different dataset versions:

- **Missed Threats**: Malicious packets wrongly classified as 'BENIGN' (False Negatives). In a NIDS environment, this is the most serious failure because it lets threats enter the network undetected.
- **Misidentified Threats**: Malicious packets that the model correctly detected as an anomaly or attack but were misclassified into the wrong specific attack category (for example, labeling a 'DDoS' payload as a 'PortScan'). Although the threat is identified, incident response may be temporarily misdirected.

- **False Alarms**: Normal 'BENIGN' traffic incorrectly flagged as an attack (False Positives). High rates of false alarms significantly diminish the usability of a NIDS by causing alert fatigue among security administrators.

Figures 8.6 and 8.7 illustrate the percentage of total samples in these three error categories before and after applying AT. The comparison shows a significant shift in the network's behavior: after AT, the Missed Threats rate drops sharply from around 0.2%–0.7% to nearly zero (below 0.005%). This suggests that AT made the model highly vigilant, nearly catching all malicious payloads. However, this increased vigilance has a downside. To minimize misses, the model's decision boundaries become overly aggressive, resulting in a rise in False Alarms (from about 7% to roughly 8.5%) and a sharp increase in Misidentified Threats, which escalate from near-zero to substantially higher percentages.

This operational trade-off is reflected in the multi-class classification metrics presented in Figure 8.8 which offers a detailed comparison of the Macro and Weighted performance averages across all values of N before and after AT, with each subplot showing the results. Examining the Macro metrics shown in the top row of Figure 8.8, which treat all 15 network traffic categories equally regardless of their support size, the model's behavioral shift is very evident:

- **Macro Recall**: Without AT, the model's Macro Recall gradually improves as N increases, from about 0.76 at $N = 3$ to 0.86 at $N = 8$. When AT is applied, the Recall jumps to nearly 0.98 across all dataset versions, showing a flat, near-perfect curve. This indicates that "Missed Threats" are nearly eliminated, and the retrained model is highly sensitive, successfully detecting almost all minority attack cases.

- **Macro Precision**: This increased sensitivity significantly reduces Macro Precision. Initially, before AT, precision was stable around 0.80. After AT, it drops to between 0.48 and 0.58. This decline results directly from the model's near-perfect recall. By focusing on detecting adversarially altered samples, the model's decision boundaries for minority attack classes have become broader and more permissive. As a result, when facing borderline or unclear traffic patterns, the network tends to classify them as anomalies to avoid missing actual threats (False Negatives). This widened net inevitably captures a high volume of benign or misclassified packets, leading to more False Positives and lowering the overall Macro Precision.

- **Macro F1-Score**: Consequently, the Macro F1-Score curve, which balances Precision and Recall, drops significantly after AT. While the original model showed a steady upward trend reaching approximately 0.83, the defensively trained model displays a lower and more volatile trajectory oscillating between 0.58 and 0.67. The large number of Misidentified Threats and False Alarms pulls the harmonic mean down, despite the perfect Recall.

However, looking at the Weighted metrics (shown in the bottom row of Figure 8.8) offers essential insight into the model's overall stability. Since weighted metrics reflect the significant class imbalance in the dataset, they are mainly affected by the dominant 'BENIGN' class, which makes up over 90% of the traffic:

- **Weighted Precision**: Notably, the Weighted Precision stays highly competitive after AT, consistently ranging from 0.940 to 0.946 and even surpassing the non-AT model at higher N values. This suggests that the model is almost always correct when predicting 'BENIGN'.
- **Weighted Recall & F1-Score**: The Weighted Recall and Weighted F1-Score experience only marginal, fractional declines after AT.

Ultimately, the detailed plots in Figure 8.8 quantify the fundamental trade-off of applying AT in an imbalanced NIDS environment. The defense successfully secures the network against evasion and nearly eliminates undetected intrusions, maximizing Macro Recall. However, the runtime cost of this robustness results in a severe decrease in minority-class precision. The system becomes significantly safer from an intrusion perspective, but the increased volume of false alarms and misidentified threats means security analysts must triage more alerts.

To further contextualize these macro-level shifts, a direct comparison between the class-by-class metrics before AT (Table 8.3) and after AT (Table 8.6) reveals how the model's predictive behavior changed in detail. Before AT, the CNN exhibited high precision but struggled markedly with the recall of extremely rare minority classes. For instance, for $N = 3$ in Table 8.3, classes such as Infiltration and SQL Inj. achieved perfect precision (1.0000) but poor recall (0.4286 and 0.6000, respectively), indicating that the model frequently failed to detect these attacks. After AT, however, Table 8.6 shows a radical inversion. The recall for almost every attack class, regardless of its support size, rises to nearly

1.0000 across all dataset versions. The defensively trained model successfully detects virtually every instance of Bot, Infiltration, SQL Inj., and Web-Brute attacks. The direct consequence of AT, however, is a catastrophic collapse in the precision of these same minority classes. For example, the precision for Infiltration falls to the 0.0036–0.0206 range, while PortScan drops sharply to approximately 0.01–0.03. This class-by-class breakdown clarifies the precise mechanism underlying the robustness cost: in order to minimize False Negatives (Missed Threats), the retrained network adopts a much wider and more permissive detection strategy. Consequently, when processing ambiguous traffic, it systematically biases predictions toward rare attack categories rather than risking a missed threat, directly causing the observed spike in Misidentified Threats.

Version	Metric	BENIGN	Bot	DDoS	DoS Gold.	DoS Hulk	DoS Slow.	DoS slowl.	FTP-Pat.	Heartbl.	Infil.	PortScan	SSH-Pat.	Web-Brute	SQL Inj.	Web-XSS
N=3	Precision	1.0000	0.2953	0.6364	0.8285	0.8763	0.6555	0.8411	0.6226	0.4734	0.0036	0.0162	0.4026	0.1946	0.2083	0.2411
	Recall	0.8921	1.0000	0.9940	1.0000	0.9765	1.0000	1.0000	0.9767	1.0000	1.0000	0.8395	0.9977	0.9931	1.0000	1.0000
	F1-Score	0.9430	0.4559	0.7759	0.9062	0.9237	0.7919	0.9137	0.7605	0.6426	0.0071	0.0318	0.5737	0.3254	0.3448	0.3885
	Support	602434	331	64513	3270	88798	1543	1562	9970	534	7	81	1742	144	5	155
	Overall	**Accuracy:** 0.9126 | **Macro Avg (P/R/F1):** 0.4864 / 0.9780 / 0.5856 | **Weighted Avg:** 0.9466 / 0.9126 / 0.9224														
N=4	Precision	1.0000	0.2261	0.6470	0.8305	0.8843	0.6706	0.8698	0.6576	0.2826	0.1429	0.0110	0.5948	0.4644	0.5385	0.2726
	Recall	0.8902	1.0000	0.9934	0.9992	0.9504	0.9995	0.9995	0.9838	1.0000	1.0000	0.9490	0.9776	0.9947	1.0000	1.0000
	F1-Score	0.9419	0.3688	0.7836	0.9071	0.9162	0.8027	0.9301	0.7883	0.4406	0.2500	0.0217	0.7396	0.6332	0.7000	0.4284
	Support	691699	440	83040	3957	118244	1856	1858	13372	729	9	98	2320	190	7	205
	Overall	**Accuracy:** 0.9099 | **Macro Avg (P/R/F1):** 0.5395 / 0.9825 / 0.6435 | **Weighted Avg:** 0.9441 / 0.9099 / 0.9201														
N=5	Precision	1.0000	0.1900	0.6510	0.9438	0.8890	0.6607	0.8736	0.6653	0.2617	0.0129	0.0155	0.5705	0.4154	0.0737	0.4305
	Recall	0.8812	1.0000	0.9931	0.9998	0.9746	0.9995	0.9995	0.9863	0.9989	1.0000	0.9444	0.9775	0.9869	1.0000	1.0000
	F1-Score	0.9369	0.3194	0.7865	0.9710	0.9299	0.7955	0.9323	0.7946	0.4147	0.0255	0.0305	0.7205	0.5847	0.1373	0.6019
	Support	764916	457	99348	4350	147133	2125	2136	16681	924	12	108	2894	229	7	257
	Overall	**Accuracy:** 0.9082 | **Macro Avg (P/R/F1):** 0.5102 / 0.9828 / 0.5987 | **Weighted Avg:** 0.9419 / 0.9082 / 0.9175														
N=6	Precision	1.0000	0.2970	0.6592	0.9181	0.8939	0.6696	0.8881	0.6688	0.2392	0.0813	0.0113	0.5596	0.1541	0.1270	0.2482
	Recall	0.8762	1.0000	0.9793	0.9989	0.9789	0.9983	0.9996	0.9886	1.0000	1.0000	0.9829	0.9821	0.9962	1.0000	1.0000
	F1-Score	0.9340	0.4580	0.7880	0.9568	0.9345	0.8016	0.9405	0.7979	0.3861	0.1503	0.0223	0.7130	0.2669	0.2254	0.3977
	Support	835168	474	114000	4701	175559	2373	2413	19877	1120	13	117	3465	263	8	307
	Overall	**Accuracy:** 0.9054 | **Macro Avg (P/R/F1):** 0.4944 / 0.9854 / 0.5849 | **Weighted Avg:** 0.9407 / 0.9054 / 0.9154														
N=7	Precision	0.9999	0.6544	0.6650	0.9263	0.8975	0.6822	0.8687	0.6707	0.2998	0.0206	0.0123	0.6069	0.3169	0.6154	0.4317
	Recall	0.8804	1.0000	0.9821	0.9973	0.9755	0.9989	0.9989	0.9903	0.9985	1.0000	0.9841	0.9824	0.9966	1.0000	0.9944
	F1-Score	0.9364	0.7911	0.7930	0.9605	0.9349	0.8107	0.9293	0.7998	0.4612	0.0404	0.0243	0.7503	0.4809	0.7619	0.6020
	Support	899686	481	124410	4877	202990	2611	2689	23005	1318	16	126	4031	297	8	356
	Overall	**Accuracy:** 0.9091 | **Macro Avg (P/R/F1):** 0.5779 / 0.9853 / 0.6718 | **Weighted Avg:** 0.9409 / 0.9091 / 0.9179														
N=8	Precision	1.0000	0.1733	0.6683	0.9042	0.9022	0.6945	0.9022	0.6712	0.4631	0.0115	0.0314	0.5602	0.1740	0.0640	0.4147
	Recall	0.8762	1.0000	1.0000	1.0000	0.9917	0.9993	0.9993	0.9905	1.0000	1.0000	0.9485	0.9872	0.9970	1.0000	0.9975
	F1-Score	0.9340	0.2954	0.8012	0.9497	0.9448	0.8195	0.9483	0.8001	0.6330	0.0228	0.0609	0.7147	0.2963	0.1203	0.5859
	Support	961535	488	133787	5047	229969	2841	2964	26038	1519	18	136	4598	333	8	407
	Overall	**Accuracy:** 0.9114 | **Macro Avg (P/R/F1):** 0.5090 / 0.9858 / 0.5951 | **Weighted Avg:** 0.9409 / 0.9114 / 0.9185														

Table 8.6: Detailed classification metrics for each class, as well as overall performance, of the best CNN models with adversarial training across the different dataset versions.

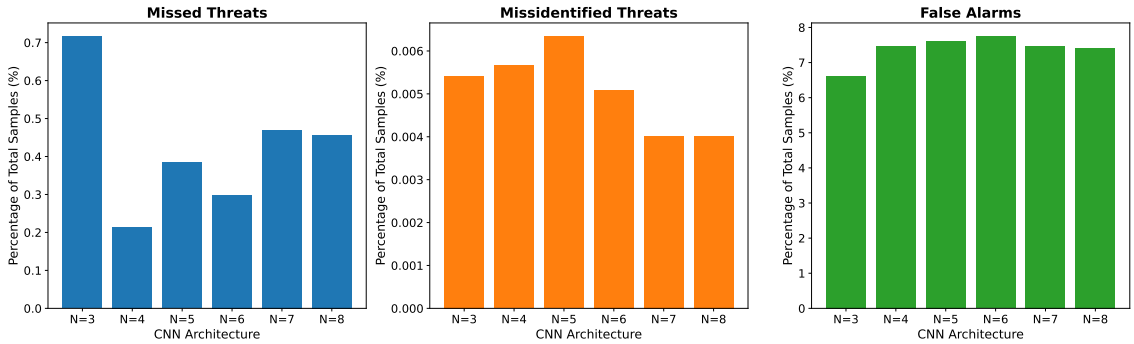


Figure 8.6: CNN performance metrics (Missed Threats, Misidentified Threats, and False Alarms) without AT.

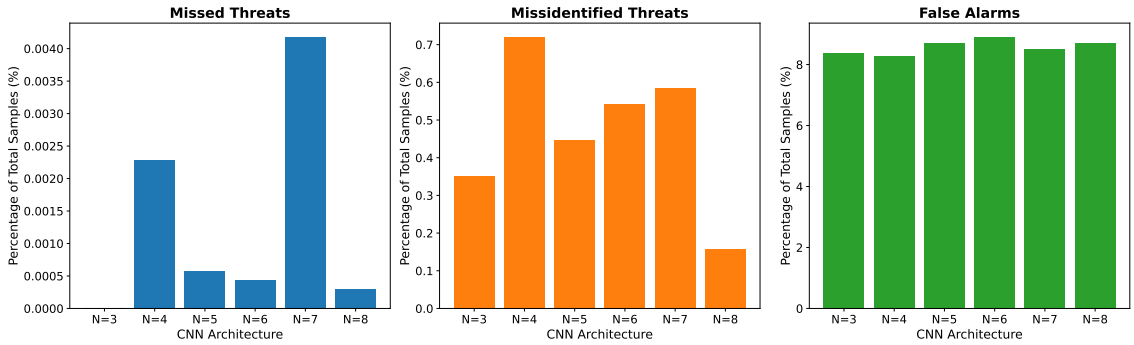


Figure 8.7: CNN performance metrics (Missed Threats, Misidentified Threats, and False Alarms) with AT.

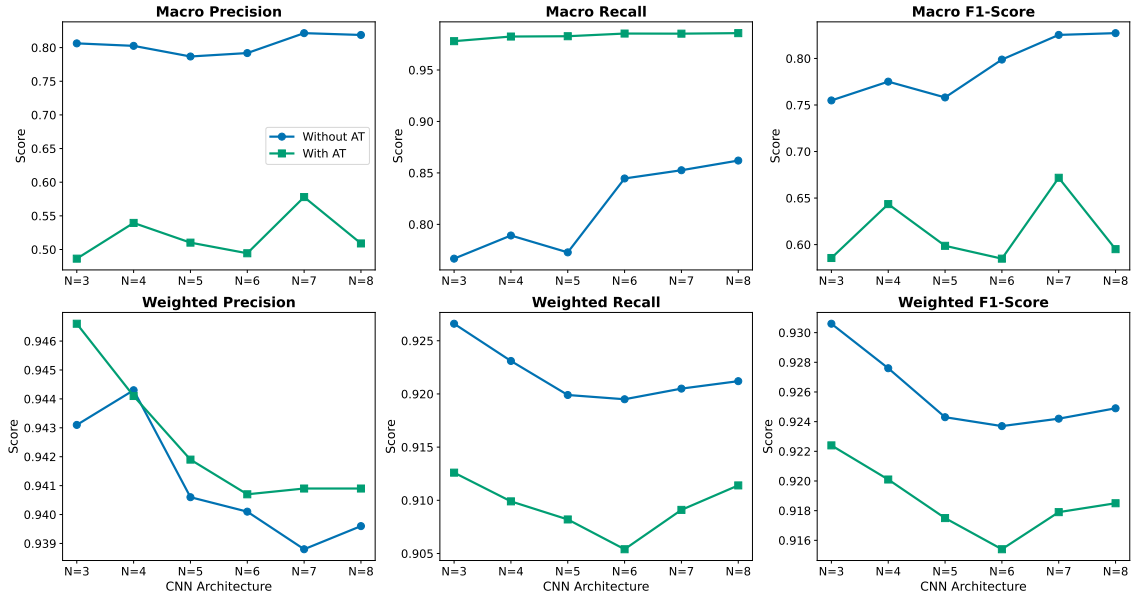


Figure 8.8: Impact of AT on Macro and Weighted performance metrics across CNN architectures.

8.4. Overall Discussion

The experimental phases detailed in this chapter reveal critical insights into the performance, vulnerabilities, and defensive trade-offs of DL models applied to image-based network intrusion detection.

A key methodological choice was to evaluate the CNN as a multi-class classifier rather than limiting it to binary anomaly detection. While binary classification is common in many NIDS studies, a multi-class framework yields far deeper operational insights. In real-world security scenarios, it is essential not only to detect anomalies but also to identify the specific nature of each threat. The multi-class approach enabled detailed analysis of the model's decision boundaries, clarifying which attack types are most challenging to differentiate. For example, it revealed operational errors such as Misidentified Threats, which can temporarily misdirect incident response efforts. This approach also proved essential for assessing the true impact of AT, as it demonstrated how the precision of certain minority classes (such as Infiltration and SQL Injection) declines when the model's overall sensitivity increases.

8.4.1 Detection Performance Insights

During the baseline testing phase (Section 8.2), the results revealed a clear contrast between the two candidate architectures. The supervised CNN leveraged the proposed transition-based image generation method effectively. By combining a packet-based approach with transition-aware aggregation, the methodology captured and enhanced the temporal context of network flows. As the number of packets per image (N) increased, the CNN acquired a broader temporal perspective of network sessions, enabling the detection of subtle, long-duration multi-packet attacks. This led to significant improvements in Macro Recall and Macro F1-Score without compromising precision.

In contrast, the unsupervised AE proved fundamentally unsuited to the packet-aggregation approach. Its inability to detect threats was not simply due to inadequate hyperparameter tuning, but rather reflected deep structural limitations in handling aggregated sequential data. Because the AE computes a single reconstruction error for each image, the influence of a localized malicious packet becomes increasingly diluted as N increases. Consequently, large volumes of benign traffic mask anomalies, resulting in severe detection failures and exposing the AE's vulnerability.

8.4.2 Adversarial Robustness Analysis

The adversarial testing phase (Section 8.3) highlighted the need for strong defense mechanisms in ML-based NIDS. Initially, the CNN models were highly vulnerable to evasion attacks that used minimal protocol-valid changes, such as altering the TTL or delta time, achieving evasion success rates of up to 30.5%. Implementing AT proved very effective. Retraining the CNN on static adversarial samples enabled the model to learn the structural patterns of threats, reducing evasion rates against dynamic, pseudo-random attacks to only 1.4% to 3.1%.

8.4.3 Trade-off: Robustness vs Precision

This defensive improvement came at a clear cost to robustness. AT greatly increased the model's vigilance, pushing Macro Recall to near-perfect levels and virtually eliminating Missed Threats. At the same time, it required the network to adopt much wider, more permissive decision boundaries. To avoid missing intrusions, the model flagged more ambiguous traffic, which led to a significant drop in Macro Precision and a surge in False Alarms and Misidentified Threats.

8.4.4 Operational Impact (SOCs)

In conclusion, the experimental findings demonstrate that the proposed image-based CNN, particularly when combined with Adversarial Training (AT), forms a robust intrusion detection framework. This defensive approach significantly reduces evasion attempts and minimizes undetected threats, but it comes with notable operational trade-offs. Unlike previous studies that emphasize only theoretical robustness, this work provides a concrete quantification of the operational costs associated with deploying Adversarial Training in real-world NIDS environments. Achieving heightened sensitivity requires the model to expand its classification boundaries, which inevitably leads to more false positives. By directly measuring the resulting increase in false alarms and misidentified threats, this research offers Security Operations Centers (SOCs) a realistic and quantifiable understanding of the true cost of adversarial defenses. Consequently, while theoretical security is substantially improved, the practical effectiveness of this approach ultimately relies on implementing robust alert management strategies to handle the increased volume of warnings and prevent alert fatigue.

8.5. Operational Overhead Analysis

To assess the real-world feasibility of the candidate models, an inference latency analysis was conducted using a stratified subset of approximately 50,000 test samples. Table 8.7 summarizes the mean inference times alongside their sample standard deviations across 10 independent trials, comparing CPU and single-GPU execution both before and after applying AT. This analysis yields three key conclusions:

1. **Hardware Acceleration is Essential**: In all scenarios evaluated, GPU inference consistently outperformed CPU execution by a factor of 4 to 6. For instance, processing 50,000 samples at $N = 3$ required an average of 29.3 ± 2.07 seconds on the CPU, but only 5.27 ± 1.30 seconds on the GPU. These results clearly demonstrate that dedicated hardware acceleration is crucial for enabling near real-time threat detection in high-throughput operational settings.

2. **Minimal Runtime Overhead from Adversarial Defenses**: Mean inference times and their associated standard deviations show no meaningful difference before and after applying AT. Since AT is performed exclusively during training, where model weights are adjusted using augmented data without introducing new structural layers, the network achieves adversarial robustness without degrading inference speed. These findings confirm that the defense's "robustness cost" appears solely as an operational penalty in the form of reduced precision, rather than as a computational bottleneck during live inference.

3. **Architectural Complexity vs. Input Dimensionality**: Inference times do not increase proportionally with the number of packets per image (N). The sharp runtime spike at $N = 6$ (for example, 39.77 ± 4.66 seconds on CPU), followed by a notable decrease at $N = 8$ (11.81 ± 1.08 seconds on CPU), demonstrates this non-linear trend. These results suggest that inference latency is influenced more by the model architecture, particularly the number of filters and dense units optimized during hyperparameter tuning for each dataset version, than by the raw spatial dimensions of the input image.

	PRE-AT		POST-AT	
Dataset	**CPU**	**GPU**	**CPU**	**GPU**
N=3	29.3 ± 2.07	5.27 ± 1.3	28.33 ± 3.73	5.53 ± 6.39
N=4	23.12 ± 1.38	4.01 ± 0.26	20.28 ± 2.68	4.25 ± 5.17
N=5	15.29 ± 1.34	2.60 ± 0.24	13.69 ± 1.76	2.51 ± 3.66
N=6	39.77 ± 4.66	6.13 ± 0.37	37.81 ± 6.07	5.95 ± 11.98
N=7	22.61 ± 1.77	4.37 ± 0.23	21.90 ± 2.25	4.32 ± 4.96
N=8	11.81 ± 1.08	3.70 ± 2.74	13.15 ± 2.99	3.77 ± 3.55

Table 8.7: Impact of adversarial training on CPU and GPU inference latency: Mean $\pm$ sample standard deviation across varying dataset versions.

9. Conclusion

This thesis investigated the robustness of Deep Learning-based Network Intrusion Detection Systems (NIDS) against adversarial evasion attacks in packet-based environments, with the goal of developing more resilient and practically deployable detection mechanisms.

To address this goal, four main objectives were defined. First, this work analyzed adversarial threats targeting ML- and DL-based NIDS, with a particular focus on protocol-compliant evasion attacks. The results confirmed that even minimal and constrained perturbations, such as modifications to TTL values, timing characteristics, and payload structure, can significantly impact detection performance, highlighting the practical risk posed by adversarial manipulation in real-world network environments.

Second, this thesis proposed a transition-based packet-to-image representation methodology designed to preserve both temporal and directional characteristics of network traffic. By aggregating packets into sessions using bidirectional 5-tuple identifiers and generating image representations triggered by traffic transitions, the proposed approach enables a richer contextual understanding of network behavior. Experimental results demonstrated that this representation improves the ability of Deep Learning models, particularly Convolutional Neural Networks (CNNs), to detect complex and multi-stage attacks, while maintaining suitability for near real-time applications.

Third, this work evaluated the performance of Deep Learning models under both normal and adversarial conditions, comparing supervised and unsupervised methods. Results showed a clear difference in model capabilities. Convolutional Neural Networks (CNNs) maintained strong, with a consistent performance across various dataset configurations, especially as the number of packets per image increased, leveraging more temporal context. Conversely, Autoencoders, while effective at modeling benign traffic, were not suitable for detecting malicious or adversarial patterns. Their dependence on global reconstruction error makes them vulnerable to localized changes, which are often not significant enough to trigger an anomaly detection.

Fourth, the effectiveness and operational impact of Adversarial Training (AT) were systematically assessed. The experimental results demonstrated that AT significantly improves model robustness, reducing evasion rates from 30.5% to below 3.1% and nearly

eliminating undetected attacks. However, these gains come at a considerable cost. The introduction of adversarial samples during training alters the model's decision boundaries, leading to a substantial reduction in Macro Precision and a corresponding increase in false positives and misclassified threats.

Taken together, these findings highlight a fundamental trade-off between robustness and operational efficiency in adversarially resilient NIDS. While Adversarial Training enhances security by minimizing undetected attacks, it simultaneously increases the volume of low-confidence alerts, which may overwhelm analysts and reduce the overall efficiency of Security Operations Centers (SOCs). This trade-off represents a critical challenge for the deployment of robust, data-driven intrusion detection systems in real-world environments.

In conclusion, this thesis demonstrates that improving adversarial robustness in packet-based NIDS is both feasible and effective, particularly when using transition-aware image representations combined with CNN models. Achieving this, however, demands careful consideration of operational implications, as adversarial robustness in cybersecurity is not just a technical challenge but also an operational one. A balanced approach is essential to reconcile detection capability with practical deployment constraints.

9.1. Limitations

While the proposed framework demonstrated strong detection capabilities, this study encountered practical limitations. Severe storage constraints and the computational demands of processing large-scale image data restricted the generation and evaluation of the transition-based image dataset to $N = 8$ packets per image. As a result, it remains unclear how much additional temporal context could improve detection accuracy or at what point performance might level off or decline, since these upper limits were not explored.

9.2. Future Work

Drawing on the findings and limitations of this thesis, several compelling directions for future research arise:

- **Scaling the Image Representation ($N > 8$)**: Future research should prioritize generating and evaluating image datasets with longer packet sequences per image (such as $N = 10$ or $N = 15$). Examining these extended temporal windows will clarify the optimal sequence length for detection.

- **Mitigating False Positives Post-AT**: In light of the notable decrease in Macro Precision and rise in false alarms following AT, future research should investigate additional defensive strategies. Approaches like confidence-based thresholding, ensemble defense frameworks, or hybrid anomaly-signature systems may better balance threat sensitivity with operational precision.

- **Alternative Unsupervised Architectures**: Because the standard Auto Encoder (AE) struggled to detect localized anomalies in aggregated packet images, considering other architectures like sequence-based Long Short-Term Memory (LSTM) could offer more effective baseline anomaly detection without needing extensive labels.

- **Evaluation on Diverse Datasets**: Extending this methodology to additional modern, high-volume datasets, such as CIC-DDoS2019 and LITNET-2020, will provide stronger evidence for the generalizability of the transition-based image builder and its associated defensive strategies across diverse network topologies and attack scenarios.

- **Real-Time Network Deployment and Validation**: Migrating the proposed models and defensive mechanisms from offline simulations to a live, real-time network environment represents a crucial step toward real-world validation. Static datasets fail to capture the unpredictable traffic bursts, ongoing concept drift, and adaptive evasion tactics encountered in operational networks. By deploying in a live environment, it is possible to rigorously assess operational overhead, latency, and mitigation effectiveness, thereby bridging the gap between theoretical adversarial robustness and practical applicability.

References

[1] S. L. Jacob and P. S. Habibullah, “A systematic analysis and review on intrusion detection systems using machine learning and deep learning algorithms,” pp. 108–120, 5 2025. [Online]. Available: https://doi.org/10.47852/bonviewJCCE42023249 [Cited on pages 3, 6, and 7.]

[2] S. Ennaji, F. D. Gaspari, D. Hitaj, A. Kbidi, and L. V. Mancini, “Adversarial challenges in network intrusion detection systems: Research insights and future prospects,” *IEEE Access*, vol. 13, pp. 148 613–148 645, 2025. [Online]. Available: https://doi.org/10.1109/ACCESS.2025.3600984 [Cited on pages viii, ix, 4, 16, 17, 18, 19, 20, 21, 24, 25, 26, 27, 28, and 54.]

[3] J. Ghadermazi, A. Shah, and N. D. Bastian, “Towards real-time network intrusion detection with image-based sequential packets representation,” *IEEE Transactions on Big Data*, vol. 11, pp. 157–173, 2025. [Online]. Available: https://doi.org/10.1109/TBDATA.2024.3403394 [Cited on pages viii, 4, 31, 33, 34, 41, 43, 45, 46, and 47.]

[4] M. Handley, V. Paxson, and C. Kreibich, “Network intrusion detection: Evasion, traffic normalization, and End-to-End protocol semantics,” in *10th USENIX Security Symposium (USENIX Security 01)*. Washington, D.C.: USENIX Association, Aug. 2001. [Online]. Available: https://www.usenix.org/conference/10th-usenix-security-symposium/network-intrusion-detection-evasion-traffic-normalization [Cited on pages 4 and 41.]

[5] S. Peri, N. Vadla, Y. Panuganti, and K. Ramana, “A survey on real-time adversarial attack detection and robustness for real-time systems,” *ICCK Journal of Image Analysis and Processing*, vol. 2, no. 2, pp. 104–120, 2026. [Online]. Available: https://www.icck.org/article/abs/JIAP.2026.481078 [Cited on page 4.]

[6] M. Stamp, *Information security: principles and practice*. John Wiley & Sons, 2011. [Cited on pages 6 and 7.]

[7] P. Vanin, T. Newe, L. L. Dhirani, E. O’connell, D. O’shea, B. Lee, and M. Rao, “A study of network intrusion detection systems using artificial intelligence/machine

learning," 2022. [Online]. Available: https://doi.org/10.3390/app122211752 [Cited on pages viii, 6, 10, 11, and 16.]

[8] Z. Azam, M. M. Islam, and M. N. Huda, "Comparative analysis of intrusion detection systems and machine learning-based model analysis through decision tree," *IEEE Access*, vol. 11, pp. 80 348–80 391, 2023. [Online]. Available: https://doi.org/10.1109/ACCESS.2023.3296444 [Cited on page 7.]

[9] D. Bergmann, "What is machine learning?" https://www.ibm.com/think/topics/machine-learning, IBM, 2024, accessed: 2026-05-17. [Cited on page 8.]

[10] R. Mokkapati and V. L. Dasari, "A comprehensive review on areas and applications of artificial intelligence, machine learning, deep learning, and data science," in *2023 3rd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*, 2023, pp. 427–435. [Online]. Available: https://doi.org/10.1109/ICIMIA60377.2023.10426237 [Cited on page 8.]

[11] M. N. O. Sadiku, S. A. Ajayi, and J. O. Sadiku, "Machine learning: An overview," Tech. Rep., 2025. [Online]. Available: https://www.ijtsrd.com/papers/ijtsrd97693.pdf [Cited on page 8.]

[12] S. Mohammed, L. Budach, M. Feuerpfeil, N. Ihde, A. Nathansen, N. Noack, H. Patzlaff, F. Naumann, and H. Harmouch, "The effects of data quality on machine learning performance on tabular data," *Information Systems*, vol. 132, p. 102549, 2025. [Online]. Available: https://doi.org/10.1016/j.is.2025.102549 [Cited on page 9.]

[13] Denodo, "Data preprocessing: Definition, importance, and key techniques," https://www.denodo.com/en/glossary/data-preprocessing-definition-importance-key-techniques, 2026, accessed: 2026-05-19. [Cited on page 9.]

[14] Q. Xiang, L. Zi, X. Cong, and Y. Wang, "Concept drift adaptation methods under the deep learning framework: A literature review," *Applied Sciences*, vol. 13, no. 11, 2023. [Online]. Available: https://doi.org/10.3390/app13116515 [Cited on page 9.]

[15] A. Alotaibi and M. A. Rassam, "Adversarial machine learning attacks against intrusion detection systems: A survey on strategies and defense," 2 2023. [Online]. Available: https://doi.org/10.3390/fi15020062 [Cited on page 10.]

[16] C. C. Aggarwal, *Data Mining: The Textbook*. Springer International Publishing, 1 2015. [Online]. Available: https://doi.org/10.1007/978-3-319-14142-8 [Cited on page 10.]

[17] "Supervised vs. Unsupervised Learning: Pros, Cons, and When to Choose — coursera.org," https://www.coursera.org/articles/supervised-vs-unsupervised-learning, [Accessed 24-01-2026]. [Cited on pages viii and 11.]

[18] V. Dutta, M. Choraś, M. Pawlicki, and R. Kozik, "A deep learning ensemble for network anomaly and cyber-attack detection," *Sensors*, vol. 20, no. 16, 2020. [Online]. Available: https://doi.org/10.3390/s20164583 [Cited on page 11.]

[19] T. Neupert, M. H. Fischer, E. Greplova, K. Choo, and M. M. Denner, "Supervised learning with neural networks: Computational neurons," https://ml-lectures.org/docs/supervised_learning_w_NNs/ml_intro_neural.html, 2021, accessed: August 1, 2026. [Cited on pages 11 and 12.]

[20] F. La Vigne, "Artificially intelligent - a closer look at neural networks," https://learn.microsoft.com/en-us/archive/msdn-magazine/2019/february/artificially-intelligent-a-closer-look-at-neural-networks, 2019, accessed: 2026-05-16. [Cited on page 12.]

[21] V. Ghorakavi, "Neural networks: A beginner's guide," https://www.geeksforgeeks.org/deep-learning/neural-networks-a-beginners-guide/, 2024, accessed: 2024-05-23. [Cited on page 12.]

[22] TechTarget, "What is a backpropagation algorithm?" https://www.techtarget.com/searchenterpriseai/definition/backpropagation-algorithm, 2024, accessed: 2026-05-16. [Cited on page 14.]

[23] Great Learning Editorial Team, "Backpropagation algorithm in neural network," https://www.mygreatlearning.com/blog/backpropagation-algorithm/, 2025, accessed: 2026-05-16. [Cited on page 14.]

[24] Sesen AI and zhubarb, "Backpropagation demystified: Neural nets from first principles," https://sesen.ai/blog/backpropagation-neural-nets-from-first-principles, 2024, accessed: 2026-05-16. [Cited on page 14.]

[25] ramlakhan79, "Difference between shallow and deep neural networks," https://www.geeksforgeeks.org/deep-learning/difference-between-shallow-and-deep-neural-networks/, July 2025, accessed: 2026-04-12. [Cited on page 15.]

[26] H. Jmila and M. I. Khedher, "Adversarial machine learning for network intrusion detection: A comparative study," *Computer Networks*, vol. 214, 9 2022. [Online]. Available: https://doi.org/10.1016/j.comnet.2022.109073 [Cited on pages 15, 19, and 25.]

[27] Z. Ahmad, A. S. Khan, C. W. Shiang, J. Abdullah, and F. Ahmad, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," *Transactions on Emerging Telecommunications Technologies*, vol. 32, 1 2021. [Online]. Available: https://doi.org/10.1002/ett.4150 [Cited on page 16.]

[28] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, pp. 41–50, 2 2018. [Online]. Available: https://doi.org/10.1109/TETCI.2017.2772792 [Cited on page 16.]

[29] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic, "Deep learning applications and challenges in big data analytics," *Journal of Big Data*, vol. 2, 12 2015. [Online]. Available: https://doi.org/10.1186/s40537-014-0007-7 [Cited on page 16.]

[30] O. H. Abdulganiyu, T. A. Tchakoucht, and Y. K. Saheed, "A systematic literature review for network intrusion detection system (ids)," *International Journal of Information Security*, vol. 22, pp. 1125–1162, 10 2023. [Online]. Available: https://doi.org/10.1007/s10207-023-00682-2 [Cited on page 16.]

[31] P. Dini, A. Elhanashi, A. Begni, S. Saponara, Q. Zheng, and K. Gasmi, "Overview on intrusion detection systems design exploiting machine learning for networking cybersecurity," 7 2023. [Online]. Available: https://doi.org/10.3390/app13137507 [Cited on page 16.]

[32] H. Qiu, T. Dong, T. Zhang, J. Lu, G. Memmi, and M. Qiu, "Adversarial attacks against network intrusion detection in iot systems," *IEEE Internet of Things Journal*, vol. 8, pp. 10 327–10 335, 7 2021. [Online]. Available: https://doi.org/10.1109/JIOT.2020.3048038 [Cited on page 17.]

[33] E. Alhajjar, P. Maxwell, and N. D. Bastian, “Adversarial machine learning in network intrusion detection systems,” 4 2020. [Online]. Available: http://arxiv.org/abs/2004.11898 [Cited on page 17.]

[34] O. Ibitoye, R. Abou-Khamis, M. ElShehaby, A. Matrawy, and M. O. Shafiq, “The threat of adversarial attacks against machine learning in network security: A survey,” *Journal of Electronics and Electrical Engineering*, 1 2025. [Online]. Available: https://doi.org/10.37256/jeee.4120255738 [Cited on pages ix, 18, 19, and 20.]

[35] L. Muñoz-González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli, “Towards poisoning of deep learning algorithms with back-gradient optimization,” 8 2017. [Online]. Available: http://arxiv.org/abs/1708.08689 [Cited on page 18.]

[36] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Srndic, P. Laskov, G. Giacinto, and F. Roli, “Evasion attacks against machine learning at test time,” 8 2017. [Online]. Available: https://doi.org/10.1007/978-3-642-40994-3_25 [Cited on page 18.]

[37] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, “Practical black-box attacks against machine learning,” in *ASIA CCS 2017 - Proceedings of the 2017 ACM Asia Conference on Computer and Communications Security*. Association for Computing Machinery, Inc, 4 2017, pp. 506–519. [Online]. Available: https://doi.org/10.1145/3052973.3053009 [Cited on pages ix, 19, and 22.]

[38] G. Apruzzese, M. Andreolini, L. Ferretti, M. Marchetti, and M. Colajanni, “Modeling realistic adversarial attacks against network intrusion detection systems,” *Digital Threats: Research and Practice*, vol. 3, 9 2022. [Online]. Available: https://doi.org/10.1145/3469659 [Cited on page 19.]

[39] J. Lin, L. Dang, M. Rahouti, and K. Xiong, “Ml attack models: Adversarial attacks and data poisoning attacks,” Tech. Rep. [Online]. Available: https://doi.org/10.48550/arXiv.2112.02797 [Cited on page 20.]

[40] M. Pawlicki, A. Pawlicka, R. Kozik, and M. Choraś, “A meta-survey of adversarial attacks against artificial intelligence algorithms, including diffusion models,” *Neurocomputing*, vol. 653, 11 2025. [Online]. Available: https://doi.org/10.1016/j.neucom.2025.131231 [Cited on page 21.]

[41] A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial examples in the physical world,” 2 2017. [Online]. Available: http://arxiv.org/abs/1607.02533 [Cited on page 25.]

[42] N. Carlini and D. Wagner, “Towards evaluating the robustness of neural networks,” in *Proceedings - IEEE Symposium on Security and Privacy*. Institute of Electrical and Electronics Engineers Inc., 6 2017, pp. 39–57. [Online]. Available: https://doi.org/10.1109/SP.2017.49 [Cited on page 25.]

[43] S. M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, “Deepfool: A simple and accurate method to fool deep neural networks,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-December. IEEE Computer Society, 12 2016, pp. 2574–2582. [Online]. Available: https://doi.org/10.1109/CVPR.2016.282 [Cited on page 25.]

[44] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” 3 2015. [Online]. Available: http://arxiv.org/abs/1412.6572 [Cited on page 25.]

[45] M. Cisse, Y. Adi, N. Neverova, and J. Keshet, “Houdini: Fooling deep structured visual and speech recognition models with adversarial examples,” Tech. Rep. [Online]. Available: https://doi.org/10.48550/arXiv.1707.05373 [Cited on page 25.]

[46] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, *The Limitations of Deep Learning in Adversarial Settings*. IEEE, 2016. [Online]. Available: https://doi.org/10.1109/EuroSP.2016.36 [Cited on page 25.]

[47] A. M. □. Adry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models resistant to adversarial attacks,” Tech. Rep. [Online]. Available: https://doi.org/10.48550/arXiv.1706.06083 [Cited on page 25.]

[48] Z. Wang, M. Gao, J. Li, J. Zhang, and J. Zhong, “Gray-box shilling attack: An adversarial learning approach,” *ACM Transactions on Intelligent Systems and Technology*, vol. 13, 10 2022. [Online]. Available: https://doi.org/10.1145/3512352 [Cited on page 25.]

[49] Z. Lin, Y. Shi, and Z. Xue, “Idsgan: Generative adversarial networks for attack generation against intrusion detection,” J. Gama, T. Li, Y. Yu, E. Chen, Y. Zheng, and F. Teng, Eds., Cham, 2022. [Online]. Available: https://doi.org/10.1007/978-3-031-05981-0_7 [Cited on page 25.]

[50] R. Chauhan, U. Sabeel, A. Izaddoost, and S. S. Heydari, “Polymorphic adversarial cyberattacks using wgan,” *Journal of Cybersecurity and Privacy*, vol. 1, pp. 767–792, 12 2021. [Online]. Available: https://doi.org/10.3390/jcp1040037 [Cited on page 25.]

[51] J. Vitorino, N. Oliveira, and I. Praça, “Adaptative perturbation patterns: Realistic adversarial learning for robust intrusion detection,” *Future Internet*, vol. 14, 4 2022. [Online]. Available: https://doi.org/10.3390/fi14040108 [Cited on page 25.]

[52] X. Peng, W. Huang, and Z. Shi, “Adversarial attack against dos intrusion detection: An improved boundary-based method,” in *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*, vol. 2019-November. IEEE Computer Society, 11 2019, pp. 1288–1295. [Online]. Available: https://doi.org/10.1109/ICTAI.2019.00179 [Cited on page 25.]

[53] S. Zhang, X. Xie, and Y. Xu, “A brute-force black-box method to attack machine learning-based systems in cybersecurity,” *IEEE Access*, vol. 8, pp. 128 250–128 263, 2020. [Online]. Available: https://doi.org/10.1109/ACCESS.2020.3008433 [Cited on page 25.]

[54] J. Lin, L. Xu, Y. Liu, and X. Zhang, “Black-box adversarial sample generation based on differential evolution,” *Journal of Systems and Software*, vol. 170, 12 2020. [Online]. Available: https://doi.org/10.1016/j.jss.2020.110767 [Cited on page 25.]

[55] J. Su, D. V. Vargas, and K. Sakurai, “One pixel attack for fooling deep neural networks,” *IEEE Transactions on Evolutionary Computation*, vol. 23, pp. 828–841, 10 2019. [Online]. Available: https://doi.org/10.1109/TEVC.2019.2890858 [Cited on page 25.]

[56] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein generative adversarial networks,” Tech. Rep., 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1701.07875 [Cited on page 25.]

[57] P. Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C. J. Hsieh, “Zoo: Zeroth order optimization based black-box atacks to deep neural networks without training substitute models,” in *AISec 2017 - Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, co-located with CCS 2017*. Association for Computing Machinery, Inc, 11 2017, pp. 15–26. [Online]. Available: https://doi.org/10.1145/3128572.3140448 [Cited on page 25.]

[58] W. Brendel, J. Rauber, and M. Bethge, “Decision-based adversarial attacks: Reliable attacks against black-box machine learning models,” 2 2018. [Online]. Available: http://arxiv.org/abs/1712.04248 [Cited on page 25.]

[59] I. Rosenberg, A. Shabtai, L. Rokach, and Y. Elovici, “Generic black-box end-to-end attack against state of the art api call based malware classifiers,” in *Research in Attacks, Intrusions, and Defenses*, M. Bailey, T. Holz, M. Stamatogiannakis, and S. Ioannidis, Eds. Cham: Springer International Publishing, 2018, pp. 490–510. [Online]. Available: https://doi.org/10.1007/978-3-030-00470-5_23 [Cited on page 25.]

[60] J. Chen, M. I. Jordan, and M. J. Wainwright, “Hopskipjumpattack: A query-efficient decision-based attack,” in *Proceedings - IEEE Symposium on Security and Privacy*, vol. 2020-May. Institute of Electrical and Electronics Engineers Inc., 5 2020, pp. 1277–1294. [Online]. Available: https://doi.org/10.1109/SP40000.2020.00045 [Cited on page 25.]

[61] J. Vitorino, I. Praça, and E. Maia, “Sok: Realistic adversarial attacks and defenses for intelligent network intrusion detection,” 11 2023. [Online]. Available: https://doi.org/10.1016/j.cose.2023.103433 [Cited on pages viii and 25.]

[62] M. Elshehaby and A. Matrawy, “Evasion adversarial attacks remain impractical against ml-based network intrusion detection systems, especially dynamic ones,” Tech. Rep., 2025. [Online]. Available: https://arxiv.org/abs/2306.05494 [Cited on pages 26 and 28.]

[63] K. Rani and S. Dhingra, “A comprehensive survey of deep learning-based adversarial network intrusion detection systems,” in *Proceedings of the 6th International Conference on Inventive Research in Computing Applications, ICIRCA 2025*. Institute of Electrical and Electronics Engineers Inc., 2025, pp. 830–837. [Online]. Available: https://doi.org/10.1109/ICIRCA65293.2025.11089572 [Cited on page 26.]

[64] R. A. Khamis, M. O. Shafiq, and A. Matrawy, “Investigating resistance of deep learning-based ids against adversaries using min-max optimization,” in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–7. [Cited on page 26.]

[65] R. A. Khamis and A. Matrawy, “Evaluation of adversarial training on different types of neural networks in deep learning-based idss,” in *2020 International Symposium on Networks, Computers and Communications (ISNCC)*, 2020, pp. 1–6. [Online]. Available: https://doi.org/10.1109/ISNCC49221.2020.9297344 [Cited on page 26.]

[66] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff, “On detecting adversarial perturbations,” 2 2017. [Online]. Available: http://arxiv.org/abs/1702.04267 [Cited on page 26.]

[67] Z. T. M. Ali, A. Mohammed, and I. Ahmad, “Evaluating adversarial robustness of secret key-based defenses,” *IEEE Access*, vol. 10, pp. 34 872–34 882, 2022. [Online]. Available: https://doi.org/10.1109/ACCESS.2022.3162874 [Cited on page 27.]

[68] Y. Chen, M. Zhang, J. Li, and X. Kuang, “Adversarial attacks and defenses in image classification: A practical perspective,” in *2022 7th International Conference on Image, Vision and Computing, ICIVC 2022*. Institute of Electrical and Electronics Engineers Inc., 2022, pp. 424–430. [Online]. Available: https://doi.org/10.1109/ICIVC55077.2022.9886997 [Cited on page 27.]

[69] I. Tolstikhin, O. Bousquet, S. Gelly, and B. Schoelkopf, “Wasserstein auto-encoders,” 12 2019. [Online]. Available: http://arxiv.org/abs/1711.01558 [Cited on page 28.]

[70] K. Roshan, A. Zafar, and S. B. U. Haque, “Untargeted white-box adversarial attack with heuristic defence methods in real-time deep learning based network intrusion detection system,” *Computer Communications*, vol. 218, pp. 97–113, 3 2024. [Online]. Available: https://doi.org/10.1016/j.comcom.2023.09.030 [Cited on pages 29, 33, and 34.]

[71] X. Yuan, S. Han, W. Huang, H. Ye, X. Kong, and F. Zhang, “A simple framework to enhance the adversarial robustness of deep learning-based intrusion detection system,” 12 2023. [Online]. Available: http://dx.doi.org/10.1016/j.cose.2023.103644 [Cited on pages 29, 33, and 34.]

[72] M. K. Roshan and A. Zafar, “Boosting robustness of network intrusion detection systems: A novel two phase defense strategy against untargeted white-box optimization adversarial attack,” *Expert Systems with Applications*, vol. 249, 9

2024. [Online]. Available: https://doi.org/10.1016/j.eswa.2024.123567 [Cited on pages 29, 33, and 34.]

[73] E. Degirmenci, I. Ozcelik, and A. Yazici, “Adversarial attack detection approach for intrusion detection systems,” *IEEE Access*, vol. 12, pp. 195 996–196 009, 2024. [Online]. Available: https://doi.org/10.1109/ACCESS.2024.3520406 [Cited on pages 29, 33, and 34.]

[74] C. Zhang, X. Costa-Perez, and P. Patras, “Tiki-taka: Attacking and defending deep learning-based intrusion detection systems,” in *CCSW 2020 - Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop*. Association for Computing Machinery, Inc, 11 2020, pp. 27–39. [Online]. Available: https://doi.org/10.1145/3411495.3421359 [Cited on pages 29, 33, and 34.]

[75] A. Alotaibi and M. A. Rassam, “Enhancing the sustainability of deep-learning-based network intrusion detection classifiers against adversarial attacks,” *Sustainability (Switzerland)*, vol. 15, 6 2023. [Online]. Available: https://doi.org/10.3390/su15129801 [Cited on pages 30, 33, and 34.]

[76] R. Soleymanzadeh and R. Kashef, “Efficient intrusion detection using multi-player generative adversarial networks (gans): an ensemble-based deep learning architecture,” *Neural Computing and Applications*, vol. 35, pp. 12 545–12 563, 6 2023. [Online]. Available: https://doi.org/10.1007/s00521-023-08398-z [Cited on pages 30, 33, and 34.]

[77] Z. Bolor-Erdene, S. Ryan, M. Patrick, W. M. J., Z. Sencun, Z. Shitong, and K. Srikanth, “Generating practical adversarial network traffic flows using nidsgan,” vol. 2657, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2203.06694 [Cited on pages 30, 33, and 34.]

[78] V. Hnamte, H. Nhung-Nguyen, J. Hussain, and Y. Hwa-Kim, “A novel two-stage deep learning model for network intrusion detection: Lstm-ae,” *IEEE Access*, vol. 11, pp. 37 131–37 148, 2023. [Online]. Available: https://doi.org/10.1109/ACCESS.2023.3266979 [Cited on pages 30, 33, and 34.]

[79] Y. C. Wang, Y. C. Houng, H. X. Chen, and S. M. Tseng, “Network anomaly intrusion detection based on deep learning approach,” *Sensors*, vol. 23, 2 2023. [Online]. Available: https://doi.org/10.3390/s23042171 [Cited on pages 30, 33, and 34.]

[80] G. Karatas, O. Demir, and O. K. Sahingoz, "Increasing the performance of machine learning-based idss on an imbalanced and up-to-date dataset," *IEEE Access*, vol. 8, pp. 32 150–32 162, 2020. [Online]. Available: https://doi.org/10.1109/ACCESS.2020.2973219 [Cited on pages 31, 33, and 34.]

[81] Z. Awad, M. Zakaria, and R. Hassan, "An enhanced ensemble defense framework for boosting adversarial robustness of intrusion detection systems," *Scientific Reports*, vol. 15, 12 2025. [Online]. Available: https://doi.org/10.1038/s41598-025-94023-z [Cited on pages 31, 33, and 34.]

[82] I. Debicha, R. Bauwens, T. Debatty, J. M. Dricot, T. Kenaza, and W. Mees, "Tad: Transfer learning-based multi-adversarial detection of evasion attacks against network intrusion detection systems," *Future Generation Computer Systems*, vol. 138, pp. 185–197, 1 2023. [Online]. Available: https://doi.org/10.1016/j.future.2022.08.011 [Cited on pages 31, 33, and 34.]

[83] K. Roshan, A. Zafar, and S. B. Ul Haque, "A novel deep learning based model to defend network intrusion detection system against adversarial attacks," in *2023 10th International Conference on Computing for Sustainable Global Development (INDIACom)*, 2023, pp. 386–391. [Cited on pages 31, 33, and 34.]

[84] K. He, D. D. Kim, and M. R. Asghar, "Mtd-ad: Moving target defense as adversarial defense," *IEEE Transactions on Dependable and Secure Computing*, vol. 22, pp. 5047–5059, 2025. [Online]. Available: https://doi.org/10.1109/TDSC.2025.3560246 [Cited on pages 32, 33, and 34.]

[85] N. Wang, Y. Chen, Y. Xiao, Y. Hu, W. Lou, and Y. T. Hou, "Manda: On adversarial example detection for network intrusion detection system," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, pp. 1139–1153, 3 2023. [Online]. Available: https://doi.org/10.1109/TDSC.2022.3148990 [Cited on pages 32, 33, and 34.]

[86] H. Zhang, D. Han, S. Zhuang, Z. Wang, J. Sun, Y. Liu, J. Liu, and J. Dong, "Explainable and transferable adversarial attack for ml-based network intrusion detectors," *IEEE Transactions on Dependable and Secure Computing*, vol. 22, pp. 5090–5107, 2025. [Online]. Available: https://doi.org/10.1109/TDSC.2025.3560486 [Cited on pages 32, 33, and 34.]

[87] F. Alotaibi and S. Maffeis, “Mateen: Adaptive ensemble learning for network anomaly detection,” in *ACM International Conference Proceeding Series*. Association for Computing Machinery, 9 2024, pp. 215–234. [Online]. Available: https://doi.org/10.1145/3678890.3678901 [Cited on pages 32, 33, and 34.]

[88] K. Barik and S. Misra, “Ids-anta: An open-source code with a defense mechanism to detect adversarial attacks for intrusion detection system,” *Software Impacts*, vol. 21, 9 2024. [Online]. Available: https://doi.org/10.1016/j.simpa.2024.100664 [Cited on pages 32, 33, and 34.]

[89] N. Moustafa and J. Slay, “Unsw-nb15: A comprehensive data set for network intrusion detection systems (unsw-nb15 network data set),” Tech. Rep. [Online]. Available: https://doi.org/10.1109/MilCIS.2015.7348942 [Cited on pages viii, 35, and 41.]

[90] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, “Toward generating a new intrusion detection dataset and intrusion traffic characterization,” in *ICISSP 2018 - Proceedings of the 4th International Conference on Information Systems Security and Privacy*, vol. 2018-January. SciTePress, 2018, pp. 108–116. [Online]. Available: https://doi.org/10.5220/0006639801080116 [Cited on pages viii, 36, 41, 43, and 45.]

[91] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, “Developing realistic distributed denial of service (ddos) attack dataset and taxonomy,” in *2019 International Carnahan Conference on Security Technology (ICCST)*, 2019, pp. 1–8. [Online]. Available: https://doi.org/10.1109/CCST.2019.8888419 [Cited on pages viii, 36, and 37.]

[92] R. Damasevicius, A. Venckauskas, S. Grigaliunas, J. Toldinas, N. Morkevicius, T. Aleliunas, and P. Smuikys, “Litnet-2020: An annotated real-world network flow dataset for network intrusion detection,” *Electronics (Switzerland)*, vol. 9, 5 2020. [Online]. Available: https://doi.org/10.3390/electronics9050800 [Cited on pages viii and 37.]

[93] GeeksforGeeks, “Precision-recall curve - ML,” https://www.geeksforgeeks.org/machine-learning/precision-recall-curve-ml/, 2024, last updated: July 12, 2024. Accessed: [Insert your date here]. [Cited on page 39.]

[94] Evidently AI, “Accuracy, precision, and recall in multi-class classification,” August 2025, accessed: 2026-04-11. [Online]. Available: https://www.evidentlyai.com/classification-metrics/multi-class-metrics [Cited on pages 39 and 40.]

[95] K. He, D. D. Kim, and M. R. Asghar, “Adversarial machine learning for network intrusion detection systems: A comprehensive survey,” *IEEE Communications Surveys and Tutorials*, vol. 25, pp. 538–566, 2023. [Online]. Available: https://doi.org/10.1109/COMST.2022.3233793 [Cited on page 41.]

[96] W. Lee, S. J. Stolfo, P. K. Chan, E. Eskin, W. Fan, M. Miller, S. Hershkop, and J. Zhang, “Real time data mining-based intrusion detection,” in *Proc. DARPA Inf. Survivability Conf. Expo. II*, 2001, pp. 89–100. [Online]. Available: https://doi.org/10.1109/DISCEX.2001.932195 [Cited on page 41.]

[97] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, “Characterization of encrypted and VPN traffic using time-related features,” in *Proc. 2nd Int. Conf. Inf. Syst. Security Privacy (ICISSP)*, Rome, Italy, 2016, pp. 407–414. [Cited on page 41.]

[98] The Zeek Project, “Zeek: The network security monitor,” https://zeek.org/, 2024, accessed: 2024-05-20. [Cited on page 41.]

[99] P. Cheng, M. Han, and G. Liu, “Desc-ids: Towards an efficient real-time automotive intrusion detection system based on deep evolving stream clustering,” *Future Generation Computer Systems*, vol. 140, pp. 266–281, 3 2023. [Online]. Available: https://doi.org/10.1016/j.future.2022.10.020 [Cited on page 41.]

[100] B. Yang, M. H. Arshad, and Q. Zhao, “Packet-level and flow-level network intrusion detection based on reinforcement learning and adversarial training,” *Algorithms*, vol. 15, 12 2022. [Online]. Available: https://doi.org/10.3390/a15120453 [Cited on page 41.]

[101] Y. A. Farrukh, I. Khan, S. Wali, D. Bierbrauer, J. A. Pavlik, and N. D. Bastian, “Payload-byte: A tool for extracting and labeling packet capture files of modern network intrusion detection datasets,” in *2022 IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (BDCAT)*, 2022, pp. 58–67. [Online]. Available: https://doi.org/10.1109/BDCAT56447.2022.00015 [Cited on pages ix, 43, 45, and 46.]

[102] IBM, “What are convolutional neural networks?” https://www.ibm.com/think/topics/convolutional-neural-networks, accessed: 2026-05-01. [Cited on page 49.]

[103] T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi *et al.*, "Kerastuner," https://keras.io/keras_tuner/, accessed: 2026-05-01. [Cited on page 49.]

[104] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," online; accessed 1 May 2026. [Online]. Available: https://www.tensorflow.org/ [Cited on page 49.]

[105] T. Authors, "Tensorboard: Tensorflow's visualization toolkit," https://www.tensorflow.org/tensorboard, 2026, accessed: 2026-05-01. [Cited on page 49.]

[106] GeeksforGeeks, "Categorical cross entropy in multi-class classification," https://www.geeksforgeeks.org/deep-learning/categorical-cross-entropy-in-multi-class-classification/, 2026, accessed: May 28, 2026. [Cited on page 49.]

[107] ——, "Binary cross entropy/log loss for binary classification," https://www.geeksforgeeks.org/deep-learning/binary-cross-entropy-log-loss-for-binary-classification/, July 2025, accessed: 2026-05-28. [Cited on page 50.]

[108] S. Mondal, "Understanding mae, mse, and rmse: Key metrics in machine learning," https://dev.to/mondal_sabbha/understanding-mae-mse-and-rmse-key-metrics-in-machine-learning-4la2, 2023, accessed: 2026-05-25. [Cited on page 50.]

[109] J. Wilber and B. Werness, "Train, test, and validation sets - mlu-explain," https://mlu-explain.github.io/train-test-validation/, 2022, accessed: 2026-05-01. [Cited on page 50.]

[110] M. Terzi, G. A. Susto, and P. Chaudhari, "Directional adversarial training for cost sensitive deep learning classification applications," *Engineering Applications of Artificial Intelligence*, vol. 91, p. 103550, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0952197620300452 [Cited on page 69.]