



REPUBLIC OF TÜRKİYE

ALTINBAŞ UNIVERSITY

Institute of Graduate Studies

Electrical and Computer Engineering

**CYBERSECURITY MEASURES FOR ATTACKS
DETECTION IN IOT WITH 5G BASED ON
ARTIFICIAL INTELLIGENCE TECHNOLOGIES**

Reem Talal Abdulhameed AL-DULAIMI

Doctor of Philosophy

Supervisor

Asst. Prof. Dr. Ayça KURNAZ TÜRK BEN

İstanbul, 2025

**CYBERSECURITY MEASURES FOR ATTACKS DETECTION IN
IOT WITH 5G BASED ON ARTIFICIAL INTELLIGENCE
TECHNOLOGIES**

Reem Talal Abdulhameed AL-DULAIMI

Department of Electrical and Computer Engineering

Doctor of Philosophy

ALTINBAŞ UNIVERSITY

2025

The dissertation titled CYBERSECURITY MEASURES FOR ATTACKS DETECTION IN IOT WITH 5G BASED ON ARTIFICIAL INTELLIGENCE TECHNOLOGIES prepared by REEM TALAL ABDULHAMEED AL-DULAIMI and submitted on 23/06/2025 has been **accepted unanimously** for the degree of PhD in Electrical and Computer Engineering

Asst. Prof. Dr. Ayça KURNAZ TÜRKBEN
Supervisor

Thesis Defence Committee Members:

Asst. Prof. Dr. Ayça KURNAZ TÜRKBEN	Department of Software Engineering, Altınbaş University	_____
Prof. Dr. Osman Nuri UÇAN	Department of Electrical and Electronic Engineering, Altınbaş University	_____
Asst. Prof. Dr. Hameed Mutlag Farhan FARHAN	Department of Electrical and Computer engineering, Altınbaş University	_____
Asst. Prof. Dr. Serdar KARGIN	Department of Biomedical Engineering, İstanbul Arel University	_____
Asst. Prof. Dr. Zeynep ALTAN	Department of Software Engineering, İstanbul Beykent University	_____

I hereby declare that this dissertation meets all format and submission requirements for a PhD dissertation.

I hereby declare that all information/data presented in this graduation project has been obtained in full accordance with academic rules and ethical conduct. I also declare all unoriginal materials and conclusions have been cited in the text and all references mentioned in the Reference List have been cited in the text, and vice versa as required by the abovementioned rules and conduct.

Reem Talal Abdulhameed AL-DULAIMI

Signature



ABSTRACT

CYBERSECURITY MEASURES FOR ATTACKS DETECTION IN IOT WITH 5G BASED ON ARTIFICIAL INTELLIGENCE TECHNOLOGIES

AL-DULAIMI, Reem Talal Abdulhameed

Ph.D., Department of Electrical and Computer Engineering, Altınbaş University,

Supervisor: Asst. Prof. Dr. Ayça KURNAZ TÜRK BEN

Date: 06/2025

Pages: 155

With the emerging cybersecurity arena, especially in the context of next-generation communication networks like 5G networks, precise identification of Distributed Denial of Service (DDoS) attacks is an urgent challenge. The large dimensionality, high-speed data transmission, and high network diversity typical of 5G environments have a propensity to make conventional machine learning models inadequate. To address these problems, this research suggests a new deep learning approach that uses symmetry to help detect unusual activities in today's fast and complex networks. The system architecture under examination is a Tree Convolutional Neural Network (Tree-CNN) that is particularly capable of understanding hierarchical and symmetrical interdependencies among network traffic, prevalent in 5G communications, as their architecture is distributed and layered. Supporting it is a deep autoencoder module that is employed to be capable of extracting latent symmetrical patterns, noise reduction, and improving the discriminative representation of anomaly behaviour. Model learning performance is further enhanced by the addition of a leader-influenced velocity-based spiral optimization algorithm, a novel metaheuristic, which achieves an effective exploration-exploitation trade-off. The aim here is to optimize Tree-CNN parameters, deep autoencoders, and classification thresholds, which results in improved detection accuracy but at the cost of computational practicality. In the scenario of 5G networks where there is a sense of urgency in real-time processing and adaptive threat

response the necessity for an accuracy-speed trade-off arises. Performance measurements were performed on three benchmark datasets: UNSW-NB15, CIC-IDS 2017, and CIC-IDS 2018, which represent various traffic patterns as one would see in 5G networks, such as bursty rates of traffic and multi-modal inputs. The novel framework achieved exceptional accuracy levels of 96.02% in UNSW-NB15, 99.99% in CIC-IDS 2017, and 99.96% in CIC-IDS 2018, along with nearly perfect precision and recall rates. These results justify the capability of the system to well-detect threats with very low false negatives and positives. While design imposes a medium computational overhead, this is offset by a dramatic enhancement in detection resilience as well as scalability. What is important here is that the model demonstrates its effectiveness in 5G-supported infrastructures, where high-data-rate streams, edge processing, and latency-prone services require high-performance but guaranteed security mechanisms. The proposed symmetry-aware hybrid detection model not only promotes state-of-the-art future 5G networks. It emphasizes the significance of symmetrical pattern detection, hierarchical feature learning, and adaptive optimization as essential components in the construction of next-generation smart security systems.

Keywords: 5G, Cyber Security, Deep Autoencoders, Tree Convolutional Neural Network, Distributed Denial of Service.

TABLE OF CONTENTS

	<u>Pages</u>
ABSTRACT	v
LIST OF TABLES	XI
LIST OF FIGURES.....	xii
ABBREVIATIONS.....	xvii
LIST OF SYMBOLS.....	xix
1. INTRODUCTION.....	1
1.1 BACKGROUND.....	1
1.2 PROBLEM STATEMENT.....	2
1.3 RESEARCH OBJECTIVES.....	4
1.4 RESEARCH CONTRIBUTION	5
1.5 PROPOSED FRAMEWORK.....	6
1.6 THESIS OUTLINE	8
2. LITERATURE REVIEW.....	10
2.1 INTRODUCTION.....	10
2.2 OVERVIEW OF NETWORK INTRUSION DETECTION SYSTEM (NIDS).....	11
2.2.1 Evolution of Intrusion Detection Systems (IDS).....	13
2.2.1.1 Mathematical Foundations of IDS	14
2.2.2 Importance and Relevance of IDS Detection.....	16
2.2.2.1 Mathematical Foundations of IDS Detection	16
2.2.3 Objectives of IDS Detection	17
2.3 IDS DETECTION TECHNIQUES	18
2.3.1 Signature-Based Detection.....	21
2.3.2 Anomaly-Based Detection	22
2.4 MACHINE LEARNING AND DATA MINING IN IDS DETECTION	24
2.4.1 Supervised Learning Models	26

2.4.1.1 Support Vector Machine (SVM).....	26
2.4.1.2 Decision Trees in IDS Detection.....	26
2.4.1.3 Artificial Neural Networks (ANN) in IDS Detection	28
2.4.1.4 Naïve Bayes Classifier in IDS Detection	29
2.4.2 Unsupervised Learning IN IDS	30
2.5 DEEP LEARNING TECHNIQUES IN IDS	31
2.5.1 Convolutional Neural Networks (CNN) In IDS	31
2.5.2 Recurrent Neural Networks (RNN) IN IDS.....	32
2.5.3 Autoencoders (AEs) In IDS	32
2.5.4 Long Short-Term Memory (LSTM) IN IDS.....	33
2.6 HYBRID COMPLEX MULTILAYER PERCEPTRON NETWORK	34
2.6.1 Dynamic Feature Adaption Scheme IN IDS.....	37
2.6.2 Feature Fusion by Dynamic Attention-Weighted Feature Fusion In IDS	39
2.7 CYBERSECURITY ATTACKS: A COMPREHENSIVE ANALYSIS.....	41
2.7.1 Malware Attacks	41
2.7.2 Denial-of-Service (DoS) and Distributed Denial-of-Service (DDoS) Attacks....	42
2.7.3 Man-in-the-Middle (MitM) Attacks	43
2.7.4 Social Engineering Attacks.....	43
2.7.5 IoT-Based Attacks	44
2.7.6 AI-Driven Attacks.....	44
2.8 RELATED WORK.....	45
2.9 RESEARCH GAPS IN CYBERSECURITY: AN INTEGRATED PERSPECTIVE...	46
3. PROPOSED METHODOLOGY	51
3.1 INTRODUCTION	51
3.2 DATA COLLECTION	52
3.2.1 CIC-IDS2018 Dataset	52
3.2.2 UNSW-NB15 Dataset.....	53
3.2.3 CIC-IDS-2017 Dataset.....	54

3.3 PREPRESSING DATASET.....	54
3.3.1 Log Transformation	55
3.3.2 Data Cleaning.....	56
3.3.3 Data Standardization	56
3.3.4 Data Normalization.....	57
3.4 ARCHITECTURE OF THE PROPOSED TREE-BASED CNN (T-CNN).....	57
3.5 VARIATIONAL WAVELET DENOISING AUTOENCODER (VWDA)	62
3.6 LEADER-GUIDED SPIRAL OPTIMIZATION INTEGRATION	66
3.6.1 Particle Swarm Global Search and Derivative Flow	67
3.6.2 Dynamic Leader-Guided Spiral Refinement in Hybrid Metaheuristics	69
3.7 OBJECTIVE FUNCTION FORMULATION	71
3.8 META-ENSEMBLE CLASSIFICATION USING HYBRID VOTING STRATEGY	76
3.8.1 Random Forest (RF) Classifier	77
3.8.2 Probabilistic Classification Using Naive Bayes for Intrusion Detection.....	79
3.8.3 Support Vector Machines (SVMs) Classifier	81
3.9 PERFORMANCE EVALUATION METRICS FOR CLASSIFICATION IN IDS	83
3.9.1 Confusion Matrix	84
3.9.2 Recall	84
3.9.3 F1-Score (F-Measure).....	84
3.9.4 Precision.....	85
3.9.5 Receiver Operating Characteristic (ROC) Curve and AUC	85
4. RESULTS AND DISCUSSION	86
4.1 EXPERIMENTAL ENVIRONMENT AND SETUP	86
4.2 CROSS-VALIDATION STRATEGY AND DATASET DISTRIBUTION	87
4.3 ARCHITECTURE DESIGN AND HYPERPARAMETER OPTIMIZATION	89
4.4 PERFORMANCE RESULTS OF THREE DATASETS.....	92
4.4.1 Performance Analysis on the CICIDS-2018 Dataset.....	92
4.4.2 Performance Analysis on the CICIDS-2017 Dataset.....	95

4.4.3 Performance Analysis on the UNSW-NB15 Dataset.....	99
4.5 COMPARISON WITH PREVIOUS WORK.....	102
4.5.1 Comparative Evaluation of the Proposed Model on CICIDS-2018 Dataset.....	104
4.5.2 Comparative Evaluation of the Proposed Model on CICIDS-2017 Dataset.....	109
4.5.3 Comparative Evaluation of the Proposed Model on UNSW-NB15 Dataset	113
4.6 TIME COMPLEXITY COMPARISON	117
4.6.1 Time Complexity on CIC-IDS-2018 dataset	117
4.6.2 Time Complexity On CIC-IDS-2017 Dataset.....	118
4.6.3 Time Complexity On UNSW-NB15 Dataset.....	119
4.7 DISSCUSION.....	120
4.7.1 Discussion on Execution Time Performance Across Datasets	120
4.7.2 Discussion on Accuracy Performance Across Datasets.....	122
4.7.3 Discussion on Precision Performance Across Datasets	123
4.7.4 Discussion on Recall Performance Across Datasets.....	125
4.7.5 Discussion on F-Score Performance Across Datasets	126
4.8 THESIS LIMITATIONS	127
5. CONCLUSION AND FUTURE WORK	129
5.1 CONCLUSION	129
5.2 FUTURE WORK	129
REFERENCES	131

LIST OF TABLES

	<u>Pages</u>
Table 2.1: Shows the Layer-Wise Parametric Architecture [45].....	37
Table 2.2: Comparison of IDS Detection Methods in Literature	45
Table 4.1: Depicts the Summary of The Experiential Setup for the Proposed Methodology (By Author).	87
Table 4.2: Optimized Hyperparameter Configuration for Tree-Based CNN (T-CNN) (By Author)	91
Table 4.3: Hyperparameter Layout for Variational Wavelet Autoencoder (By Author)	92

LIST OF FIGURES

	<u>Pages</u>
Figure 1.1: Working with Botnets to Initiate a DDoS Attack [6].....	2
Figure 2.1: Shows the Layered Approach for Network Security Monitoring (by Author) .	12
Figure 2.2: The timeline of IDS Development (By Author)	13
Figure 2.3: The IDS Architecture Comprises Six Interdependent Modules [14].....	14
Figure 2.4: Flowchart illustrating in the Signature-Based IDS (By Author).....	15
Figure 2.5: Shows Threat Identification and Anomaly Detection Are Two Classification Processes in The Suggested Method, Which Uses the Cascade Model [20].....	18
Figure 2.6: Shows How the ROC Curve Assesses the Trade-Off Between False Alarm Rate and Detection Probability [22].	19
Figure 2.7: Illustrates Intrusion Detection Systems (IDS) Can Be Systematically Classified Based on Five General Criteria [23].....	20
Figure 2.8: Shows the Signature-Based Detection Process [26].	22
Figure 2.9: Shows Anomaly-Based Intrusion Detection System Architecture [34]	23
Figure 2.10: Shows Classification of Machine Learning Methods for IDS Detection [36]	25
Figure 2.11: Shows SVM Model for IDS (By Author)	26
Figure 2.12: Depicts A Decision Tree-Based IDS For Discriminating Between Normal and Anomalous Network Queries (By Author).....	28
Figure 2.13: Shows ANN-based Model for IDS (By Author).....	29
Figure 2.14: Displays A Naive Bayes-Based IDS For Distinguishing Between Typical and Unusual Network Requests (By Author).....	30
Figure 2.15: Displays A CNN-Based IDS For Distinguishing Between Typical and Unusual Network Requests [42].	32

Figure 2.16: Displays Autoencoders (AES) -Based IDS For Distinguishing Between Typical and Unusual Network Requests [43].	33
Figure 2.17: Depicts A Typical Long Short-Term Memory (LSTM) Architecture Particularly Well-Adapted to Intrusion Detection [44].	34
Figure 2.18: The Architecture of The Hybrid Complex Multilayer Perceptron (MHHPN) Network [45]	36
Figure 2.19: The Malware Attacks Taxonomy [63]	42
Figure 2.20: Workflow of DDoS Attacks [64].	42
Figure 2.21: Workflow of Man-in-the-Middle (MitM) Attacks [65].	43
Figure 2.22: Illustrates A Network Social Engineering Attacks Taxonomy [66].	44
Figure 3.1: Workflow of the Proposed Methodology (By Author)	52
Figure 3.2: Visualize of CIC-IDS2018 dataset (By Author)	53
Figure 3.3: Visualize the UNSW-NB15 Dataset (By Author)	53
Figure 3.4: Visualize The CIC-IDS-2017 Dataset (By Author)	54
Figure 3.5: The Figure Displays the Histogram of the Positively Skewed Dataset Along with its Logarithmic Transformation. The Skewness of the Non-transformed Data (left) is 0.9, Whereas That of the Log-transformed Data (right) is 0.2 (By Author)	55
Figure 3.6: Figure Shows How the Input Data Are Processed First in A Convolutional Layer and Then Gradually Enhanced in A Number of Blocks with Parallel Convolutions, Dense Concatenation, And Residual Connections (By Author)	61
Figure 3.7: Provides the Overall Structure of Variational Wavelet Denoising Autoencoder (VWDA) (By Author)	66
Figure 3.8: The Flow Chart Illustrates a Standard PSO Algorithm Procedure (By Author)	69

Figure 3.9: The Flowchart Depicts the Graphical Representation of the Procedural Model of The Leader-Guided Velocity-Based Spiral Optimization Algorithm (By Author)	74
Figure 3.10: Depicts the Ensemble Classification Model (By Author).....	76
Figure 3.11: Illustrates A Structured Process of a Random Forest-Based Intrusion Detection System (IDS) To Classify Normal and Anomalous Network Traffic (By Author).....	79
Figure 3.12: Shows Naïve Bayes-Based Intrusion Detection System (IDS) Architecture for Classifying Network Traffic as Normal or Anomalous (By Author).....	81
Figure 3.13: Describes A Sequential Process for An Intrusion Detection System (IDS) (By Author)	83
Figure 3.14: Confusion Matrix Layout for the Fours Classes (By Author)	84
Figure 4.1: Illustrates the Five Folds for Training and Testing Distributions (By Author)	88
Figure 4.2: Shows the Deep Neural Network Stages (By Author).....	90
Figure 4.3: The Graph Reveals the Hyperparameter Values Over the Three Main Constituents of the Neural Network Architecture (By Author)	90
Figure 4.4: Performance of CICIDS-2018 dataset (a) Accuracy Model over Epochs (b) Loss Model over Epochs (By Author)	93
Figure 4.5: Confusion Matrix on CICIDS-2018 dataset Results (By Author)	94
Figure 4.6: The Performance of The Classifier on ROC (By Author)	95
Figure 4.7: Performance of CICIDS-2017 Dataset (a) Accuracy Model over Epochs (b) Loss Model over Epochs (By Author)	97
Figure 4.8: Confusion matrix on CICIDS-2017 dataset Results (By Author).....	98
Figure 4.9: The Performance of The Classifier on ROC (By Author)	98
Figure 4.10: Performance of UNSW-NB15 Dataset (a) Accuracy Model over Epochs (b) Loss Model over Epochs (By Author).....	100

Figure 4.11: Confusion Matrix on UNSW-NB15 Dataset Results (By Author).....	101
Figure 4.12: The Performance of The Classifier on ROC (By Author)	102
Figure 4.13: Performance Metrics of the Proposed Work Compared to the State of the Art Method (By Author)	107
Figure 4.14: Performance Metrics of the Proposed Work Compared to the State of the Art Method (By Author)	107
Figure 4.15: Performance False Rate Analysis of The Proposed Method Compared to The State-of-The-Art Method (By Author)	108
Figure 4.16: Performance of Detection Rate of The Proposed Method Compared to The State of The Art Method (By Author)	108
Figure 4.17: Performance Metrics of The Proposed Work Compared to The State of The Art Method (By Author).....	111
Figure 4.18: Performance Metrics of The Proposed Work Compared to The State of The Art Method (By Author).....	111
Figure 4.19: Performance of Detection Rate of The Proposed Method Compared to The State of The Art Method (By Author)	112
Figure 4.20: Performance False Rate Analysis of The Proposed Method Compared to The State-of-The-Art Method (By Author)	112
Figure 4.21: Performance Analysis on UNSW-NB15 (a) Precision Results (b) Recall Results (c) F1 Score (d) Accuracy Results (By Author)	115
Figure 4.22: Performance Metrics of The Proposed Work Compared to the State of the Art Method (By Author)	115
Figure 4.23: Performance of Detection Rate of The Proposed Method Compared to the State of the Art Method (By Author).....	116
Figure 4.24: Performance False Rate Analysis of the Proposed Method Compared to the State-of-the-Art Method (By Author)	116

Figure 4.25: Provide A Comparison Between the Suggested Method's Temporal Complexity and That of Various Baseline Models (By Author).....	117
Figure 4.26: Provide A Comparison Between the Suggested Method's Temporal Complexity and That of Various Baseline Models (By Author).....	118
Figure 4.27: Provide a Comparison Between the Suggested Method's Temporal Complexity and that of Various Baseline Models (By Author)	120
Figure 4.28: Gives Useful Insights into The Computational Capability of Various Intrusion Detection Frameworks (By Author)	122
Figure 4.29: Displays the Comparative Comparison of Classification Accuracy on Three Benchmark Data Sets: CIC-IDS-2018, CIC-IDS-2017, And UNSW-NB15 (By Author) 123	
Figure 4.30: Shows the Comparison of Precision Values, Which Were Tracked by The New Model Over Three Popular Intrusion Detection Datasets (By Author).....	124
Figure 4.31: Indicated Over Every Bar and The Points Linked with A Dotted Line to Easily Illustrate the Relative Variation in Performance (By Author)	126
Figure 4.32: Illustrates The F-Score Performance of The Proposed Intrusion Detection Framework on Three Benchmark Datasets: CIC-IDS-2018, CIC-IDS-2017, And UNSW-NB15 (By Author).....	127

ABBREVIATIONS

5G	:	Fifth Generation
DDoS	:	Distributed Denial of Service
IT	:	Information Technology
CNN	:	Convolution Neural Network
SVM	:	Support Vector Machine
PSO	:	Particle Swarm Optimization
AI	:	Artificial Intelligence
LGSO	:	Leader-Guided Spiral Optimization
TCNNs	:	Tree-Based Convolutional Neural Networks
NAD	:	Network Anomaly Detection
AUC	:	Area Under Curve
SOC	:	Security Operations Centre
IOC	:	Indicator of Compromise
EDR	:	Endpoint Detection and Response
IOC	:	Indicator of Compromise
SIEM	:	Security Information and Event Management
MITM	:	Man-In-The-Middle
GAN	:	Generative Adversarial Network
DNN	:	Deep Neural Network
DBN	:	Deep Belief Network
LSTM	:	Long Short-Term Memory

RBM : Restricted Boltzmann Machine

GRU : Gated Recurrent Unit

NAC : Network Access Control

HIPS : Host Intrusion Prevention System

IOC : Indicator of Compromise

ZTA : Zero Trust Architecture



LIST OF SYMBOLS

X	:	Random Variable Representing Network Traffic Features
$P(X)$	:	Probability Distribution Function of X
μ	:	Mean (expected value) of Normal Traffic Data
σ	:	Standard Deviation of Normal Traffic
σ^2	:	Variance of Normal Traffic
Z	:	Z-score (Normalized Deviation From Mean)
$P(Z)$	:	Probability Density Function of Z-score
α	:	Significance Level for Anomaly Detection
$Z_{\alpha/2}$	:	Critical Threshold From Standard Normal Distribution
$\sum$	:	Summation Operator
$\frac{a}{b}$	:	Division (fraction)
$\log(x)$	:	Logarithm Function
e^x	:	Exponential Function
Λ	:	Regularization Parameter to Avoid Overfitting
D	:	Input Dataset Used for Training and Evaluation
TP	:	True Positives
TN	:	True Negatives
FP	:	False Positives
FN	:	False Negatives
w_i	:	Weight Assigned to the $i - th$ A classifier in Ensemble Learning

1. INTRODUCTION

1.1 BACKGROUND

Distributed Denial of Service (DDoS) attacks are a network security attack that is characterized by their ability to overwhelm network resources and render network services inaccessible [1]. DDoS attacks overwhelm a victim network with an unusual amount of traffic, rendering authorized services inaccessible and causing severe operational and financial damage [2]. Growing complexity and occurrence of DDoS attacks pose ultimate difficulties in their detection and management, and hence require the development of improved mechanisms for their efficient classification and protection [3].

One of the largest problems with the classification of DDoS attacks is the scale and diversity of data that is generated by an attack. Networks that are targeted by DDoS receive unimaginably large amounts of traffic, and it is not simple for traditional security systems to examine and scan the data within a timely fashion [4]. These gargantuan amounts of data not only make real-time monitoring challenging but also make processing overheads increase exponentially, calling for more agile and scalable solutions. Furthermore, the adaptive and dynamic nature of the DDoS attacks poses another challenge. The attackers continuously refine their tactics and methods to avoid detection, using tactics like spoofing and traffic obfuscation [5]. The classifiers thus have to be capable of adapting to the changing trends, with frequent updates and revisions necessary in the detection models. This evolving nature makes static detection approaches futile, indicating the need for adaptive and pliable classification systems. The majority of the attackers or BOT masters use botnets (zombie computers) to initiate DDoS attacks and control thousands of devices simultaneously (Figure 1). Another critical problem is class imbalance. In normal network traffic data, normal traffic vastly outnumbers attack traffic, forming an imbalanced distribution that can negatively impact classification model performance. Models learned from imbalanced data can be biased to the majority class and produce low detection rates for less common types of attacks. To properly identify and classify the DDoS attack types, the model must be developed to compensate for this skewness. Confounding factors are intentional attacks, making the classification of DDoS more complicated [6].

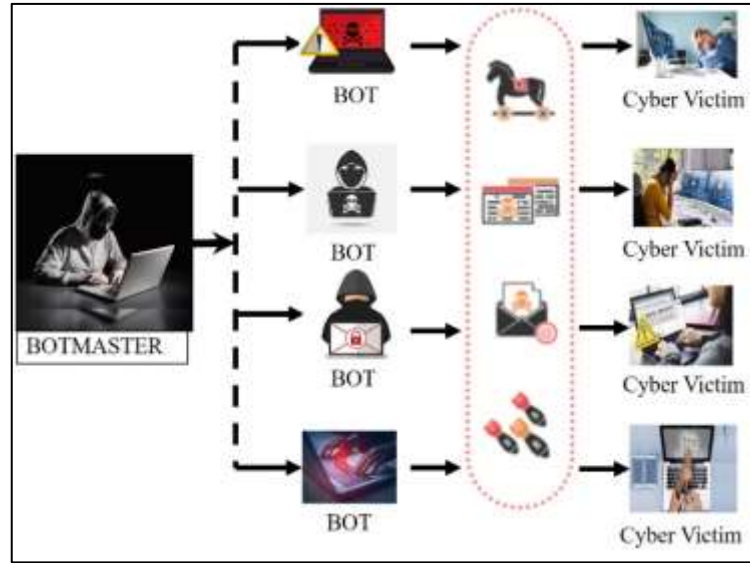


Figure 1.1: Working with Botnets to Initiate a DDoS Attack [6].

Attacks can evade detection or trick the detection algorithm by exploiting loopholes in the model [8]. In order to address adversarial attacks, better models must be created, which may increase false negatives or reduce detection accuracy [7]. The dependability and accuracy of classification systems are determined by how well the detection and remediation of adversarial attacks are executed. Another very important factor is real-time processing in the area of DDoS detection. Due to the speed and, at times, disastrous impacts of their execution, the control and reliability of the network are dependent on the quick and effective identification and response to DDoS attacks. These models need to process the incoming attack data very carefully and very quickly so that an attack can be detected and the defense mechanism activated as fast as possible. In this regard, highly sophisticated models that can process enormous amounts of data traffic and carry out complex operations in real-time using instantaneous processing capabilities are all that is required.

1.2 PROBLEM STATEMENT

Diverse Denial of Service (DDoS) cyberattacks pose an urgent threat to network infrastructure because of their ability to mangle network services and resource functionality. These interrupted services are a result of the massive amount of DDoS traffic needed to flood the system's bandwidth, computer resources, or both. Given the adoption of interconnected

and complex network structures, mitigating and detecting these offenses is exceedingly more difficult this day and age.

The problem that this research seeks to address is the accurate and efficient classification and detection of DDoS cyber-attacks, as DDoS attacks create a huge amount of heterogeneous network traffic. Remaining issues are:

- i. **High Traffic Volume and Diversity:** DDoS cyber-attacks are probably the most vicious and powerful attacks on computer systems as these attacks create flows of data that need to be contained, and these flows have to be filtered through to look for patterns that would aid in distinguishing the real attack flows from the legitimate ones. The volume of data with such diversity and complexity requires novel algorithms for timely processing and classification, scaling machine learning.
- ii. **Dynamic and Evolving Attack Patterns:** Attack strategies and patterns evolve rapidly, with attackers continuously modifying their methods to evade detection. Traditional static classification models struggle to adapt to these dynamic changes, necessitating models that can learn and adjust in real time.
- iii. **Class Imbalance:** Methods of attack as well as their sequences become more sophisticated on a daily basis because the defenders of a system approach them in a way that differs from the attacker's thinking. Therefore, old-style static classification approaches cannot fit modern-day dynamic Information technology (IT) networks, which are always changing, and one must have approaches that are adaptive and self-learning.
- iv. **Adversarial Attacks on Classifiers:** Attackers are able to exploit vulnerabilities in classification models using adversarial techniques to achieve misclassification or non-detection of attacks. This weakness underscores the need for robust models that can withstand adversarial manipulations and maintain high detection rates.
- v. **Real-Time Detection Requirements:** Since DDoS attacks can have an immediate impact, the detection systems need to function in real-time to reduce harm and launch timely countermeasures. It is a significant challenge to make models capable of processing and classifying traffic rapidly without affecting accuracy.

1.3 RESEARCH OBJECTIVES

The research develops an advanced anomaly detection system by using hybrid optimization techniques, deep learning-based feature extraction, and ensemble classification methods. Specifically, the objectives of this study are as follows.

- i. To design a hybrid optimization algorithm: Combine the strengths of Particle Swarm Optimization with Pelican Optimization (PSO-Pelican); this forms the basis for a spiral optimization approach with guiding leadership. Hence, this helps to enhance the exploration and exploitation capabilities in optimizing the deep learning model's hyperparameters and architectural components.
- ii. To develop a robust feature learning framework, use Deep Autoencoders to carry out resilient feature learning. It learns compact and meaningful latent features from high-dimensional data, like noise reduction, and enhances the representation of data for anomaly detection tasks.
- iii. Applying the Hierarchical Feature Extraction: To seize multi-scale, hierarchical features from the compressed auto-encoder representations, a Tree-Convolution Neural Network (CNN) architecture is integrated in order to enhance the detection capability of complex anomalies.
- iv. To improve the accuracy of detection via feature fusion: Deep auto-encoders and Tree-CNN models are fused in feature extraction order, in which a wide scope of anomalies is found with high precision.
- v. Applying an Ensemble Meta-classifier to make decisions: Integrate an ensemble of classifiers, for instance, Random Forest, Gradient Boosting, and Support Vector Machine (SVM), using a meta-classification approach in order to enhance the overall performance, robustness, and generalization of the hybrid anomaly detection framework.
- vi. To validate the proposed framework: Validate the proposed framework by applying the presented hybrid anomaly detection system with multiple benchmarks to existing state-of-the-art models with accuracy, precision, recall, F1-score, and Area Under Curve (AUC-ROC) measures as performance metrics.

1.4 RESEARCH CONTRIBUTION

This research provides many new contributions toward the anomaly detection technique, in particular, that can be related to intelligent systems and network security. Overall contribution:

- i. Hybrid PSO-Pelican Optimization Approach: A leader-guided spiral optimization technique is introduced by combining PSO [8] with the Pelican Optimization Algorithm [9]. This hybrid approach allows a strong balance between explorations and exploitations of the capabilities of the optimization process and thus achieves efficient tuning of deep learning model parameters. Therefore, in contrast to traditional approaches, the algorithm really thrives in navigating through complex search spaces to find the optimized model architectures and hyperparameters.
- ii. Deep Autoencoder-Based Feature Learning: We shall use Deep Autoencoders [10], which learn unsupervised feature representations. Using the bottleneck layer of an autoencoder extracts compact, high-quality features from raw, high-dimensional data. It makes for a more powerful anomaly-detecting system, especially when dealing with noisy and unstructured data.
- iii. Tree-CNN Hierarchical Feature Extraction: We present our approach by integrating a Tree-CNN hierarchical architecture [11] to enrich the feature extraction. It is a model that extracts multi-scale features from the compressed representation learned by an autoencoder. This strategy of hierarchical feature extraction allows for identifying fine-grained and high-level patterns in the data, enabling the system to be more sensitive to different kinds of anomalies.
- iv. Feature Fusion for Enhanced Detection: We propose a feature fusion approach that incorporates features from both Deep Autoencoder and Tree-CNN. Feature fusion strengthens the ability of the model to detect anomalies at various scales with diverse patterns, hence achieving a more holistic detection.
- v. Ensemble Meta-classifier for Robust Classification: We ultimately train an ensemble meta-classifier that combines predictions from a number of models like Random Forest, Gradient Boosting, and Support Vector Machine [12] to enhance general performance in

the classification task. In doing so, boosting for the ensemble method is accompanied by diminishing overfitting and thus assured to be higher in generalization across different forms of anomalies.

- vi. Exhaustive Benchmark Dataset Validation: We conducted extensive experiments on a variety of benchmark anomaly detection datasets, demonstrating the robustness of the proposed framework. Our results show that the hybrid optimization and deep learning approach is efficient in achieving state-of-the-art accuracy, precision, recall, F1-score, and AUC-ROC as compared to other existing state-of-the-art methods.

1.5 PROPOSED FRAMEWORK

In the proposed architecture, we integrated Hierarchical Feature Extraction (Tree-CNN) before the Hybrid Optimization (PSO-Pelican) step. In this context, the output from the Tree-CNN is passed to the Hybrid Optimization, thus optimizing the features extracted by Tree-CNN. This new architecture has been devised to detect anomalies in a timely manner by first capturing hierarchical features and then refining them to optimize anomaly detection. The key steps are discussed below:

- i. Data Pre-processing. This stage of the pre-processing step ensures that input data is normalized and prepared for training and testing purposes. This guarantees the stability of a consistent input during training and quality data input.
 - a. Normalization: Scale input features to a consistent range and improve model training and convergence.
 - b. Data Splitting: Split the dataset into training, validation, and testing subsets in order to evaluate the model and perform cross-validation.
- ii. Hierarchical Feature Extraction (Tree-CNN)

In this step, the Tree-CNN architecture is utilized to extract hierarchical features from the raw input data. The Tree-CNN model comprises several branches of convolutional layers that capture different aspects and scales of the data, allowing the model to understand the low as well as the high-level features.

- iii. **Multi-Scale Feature Extraction:** The input data is passed through multiple scales within each branch of the Tree-CNN, thus capturing both fine-grained details and global patterns.
- iv. **Feature Fusion:** The outputs from all the branches are concatenated. Hence, rich feature maps are obtained, comprising diversified information from varied levels of abstraction. The desired feature from the Tree-CNN is sent into the next step for optimization.
- vii. **Hybrid Optimization (PSO-Pelican) for Feature Refinement:** After hierarchical feature extraction, Hybrid PSO-Pelican Optimization is used to optimize the features extracted by Tree-CNN. This process further refines the features after hyperparameter fine-tuning on either the Deep Autoencoder, if necessary, and on the architecture of the Tree-CNN. Optimization is often in parameters that define layers and sizes, learning rates, and even the choice of architectures for maximizing model operating efficiency.
- viii. **Hybrid PSO-Pelican Optimization:** PSO, Particle Swarm Optimization. The component analyses the solution space depending on a simulation of particle movement toward an optimum solution. Actually, the search in the Pelican Optimization Algorithm is boosted by using a spiral approach that a leader leads to promising searching and exploiting the search space.
- ix. **Optimization Objectives:** The optimization adjusts the number of layers, layer sizes, activation functions, and learning rates so that the feature representations are optimal for the subsequent anomaly detection task.
- x. **Feature Representation Learning (Deep Autoencoder):** Once the features are optimized using Hybrid PSO-Pelican Optimization, the subsequent step includes Feature Representation Learning by the use of a Deep Autoencoder. Such a deep autoencoder can learn by compressing the given input features into a compact representation and reconstructing these features with minimal loss.
 - a. **Encoder:** It compresses the optimized features into the lower-dimensional space of the latent while capturing the most relevant information.

- b. Decoder: The original features are reconstructed from the compressed representation while preserving crucial details.

These hidden features are of the most significant feature information learned by the bottleneck layers of the autoencoder. So, these become the final inputs for the next stage in the architecture.

- xi. Anomaly Detection (Ensemble Meta-Classifier): The last stage of the architecture uses an Ensemble Meta-Classifier for Anomaly Detection. It relies on aggregating predictions from one or more base classifiers that have been trained on features learned and optimized through previous stages in the model. Examples of those include:
 - a. Random Forest: aggregates predictions from many decision trees to improve robustness.
 - b. Gradient Boosting: iteratively improves the performance of the classifier through learning from the error of the preceding tree.
 - c. Support Vector Machines (SVM): Classifies high-dimensional data by selecting the optimum hyperplane that best separates the classes.

The ensemble meta-classifier adds up the outputs of the base classifiers by stacking, voting, or boosting to derive the final, robust decision of whether the input data are normal or abnormal.

- xii. Output Layer (Results of Anomaly Detection): The output from the meta-classifier ensemble is a classification result indicating whether the input data is normal or abnormal. It can be represented in the following ways:
 - a. Anomaly Score: It is a continuous value describing the degree of anomaly.
 - b. Binary Classification: This refers to a decision between the input and being classified as normal or abnormal.

1.6 THESIS OUTLINE

The thesis is properly structured to develop in-depth knowledge of the detection of cyber-attacks on fifth-generation (5G) networks using Artificial Intelligence (AI) techniques,

progressing step by step from fundamentals to advanced methods and experimental studies. All the chapters are sequential to each other, and the research and findings are also well structured in a logical and sequential manner. Chapter 1 sets the background by presenting cybersecurity and the central role played by AI, specifically deep learning, in cyber-attack detection. It emphasizes the utmost need to counter cyber threats under the cover of 5G networks and noting the complexity formed by their rising complexity and the attack behaviour. Chapter 2 provides an extensive critique of advanced cyber-attack detection techniques such as conventional approaches, machine learning-based approaches, and deep learning-based approaches. It covers new developments such as multi-modal CNN fusion and Ensemble Meta-Classifiers for attack classification, and references seminal research gaps that the thesis aims to address. Chapter 3 presents the suggested hybrid model that aims to efficiently detect and distinguish real attacks from false alarms. Chapter 3 describes the architecture of the suggested model and its novel method of improving detection accuracy, especially in the high-speed and dynamic 5G network environment. Chapter 4 is also focused on experimental setup and testing, where test datasets, pre-processing approaches, and benchmarking procedures are described. It elaborates on an extensive analysis of the model's performance compared to available state-of-the-art methods and how the model can surpass them in fulfilling the identified research voids. Lastly, Chapter 5 summarizes the thesis with the main findings and contributions. It states the shortcomings of the suggested models and directions in which it can be further improved. Additionally, it also describes possible future work directions, like techniques for real-time deployment and enhancing robustness to adversarial attacks, such that the research is still feasible and beneficial for real-world applications. This systematic process not only makes the thesis respond to the present challenges of 5G network cybersecurity attack detection, but also lays a good foundation for future development in the area of study.

2. LITERATURE REVIEW

2.1 INTRODUCTION

The rapid rate of networked system development, along with the advanced nature of cyberattacks, has rendered traditional anomaly detection mechanisms ineffective. With organizations and infrastructures increasingly relying on interconnected technology, strong, adaptive, and scalable methods for real-time malicious behavior detection have become a cornerstone in cybersecurity research. This chapter weaves together four decades of scholarly research to establish a preliminary understanding of the methodologies, innovations, and challenges that have shaped the Network Anomaly Detection (NAD) field. Through a discussion of the progress from rule-based systems to the contemporary deep learning models and metaheuristic optimization schemes, this overview places the problem in perspective toward the imperative of new paradigms capable of addressing the dynamic and adversarial nature of present-day networked environments.

The chapter begins by describing the range of network anomalies, noting their variety, inherent detectability, and shortcomings of classical approaches. Early statistical and signature-based systems, while revolutionary, struggled to cope with zero-day attacks and polymorphic malware. The subsequent transition to machine learning introduced data-driven techniques that improved detection precision with feature engineering and supervised classification. However, the flood of high-dimensional network traffic data along with the need to capture hierarchical and temporal patterns exposed inherent shortcomings in conventional ML approaches. This led to the adoption of deep learning (DL) models, which automatically learned feature extraction and performed better to conventional methods in identifying complex patterns in spatial, temporal, and graph-structured data.

Much of this review is focused on talking about advances in Tree-Based Convolutional Neural Networks (TCNNs), which extend traditional convolution operations to hierarchical data representations. TCNNs have emerged as a viable solution to protocol-conscious anomaly detection, particularly where network traffic exhibits nested or tree-like dependencies (e.g., DNS hierarchies, API call sequences). But their effectiveness relies on the ability to transcend scalability, interpretability, and hyperparameter tuning difficulties in non-convex loss landscapes. It is amenable to an in-depth study of metaheuristic search

optimization algorithms, and now specifically of Leader-Guided Spiral Optimization (LGSO). By merging spiral dynamics with hierarchical population steering, LGSO offers a solid method for obtaining exploration and exploitation trade-offs in training models, correcting convergence shortfalls related to gradient-based methods.

The chapter also discusses synergies between TCNNs and optimization schools, with recent efforts on the co-optimization of model architectures and training parameters for enhanced robustness to adversarial evasion and concept drift. Comparative analyses of benchmark databases and performance metrics in the literature reveal ongoing deficits in real-time adaptability, cross-domain generalization, and computational efficiency—deficits that underscore the need for new hybrid solutions. By weaving together these multi-disciplinary strands, this overview not only sketches the research development of NAD throughout history but also identifies existing challenges to be addressed that propel the suggested integration of LGSO and TCNN. What follows in the following sections of this thesis will build upon this foundation by providing a new framework aimed at advancing the art in network anomaly detection.

2.2 OVERVIEW OF NETWORK INTRUSION DETECTION SYSTEM (NIDS)

Network Intrusion Detection Systems (NIDS) are commercial security systems that are configured to monitor, audit, and respond automatically to policy-breaking or unauthorized network system activity. NIDS employ passive or active monitoring of packet headers, payload information, and traffic flow attributes, as opposed to perimeter security systems such as firewalls that impose access control based on preconfigured rules, to detect intrusions in real time [13]. This chapter mathematizes the architectural principles, methodological paradigms, and operational workflows of NIDS, supported by mathematical rigor and algorithmic specificity. Figure 2.1: Layered Approach to Network Security Monitoring .The stratified network security monitoring is a hierarchical system of purposeful total observation and threat detection throughout the different strata of a computer network. This system decomposes network monitoring capability into separate but interconnected strata, each conducting specific types of data and corresponding attack vectors.

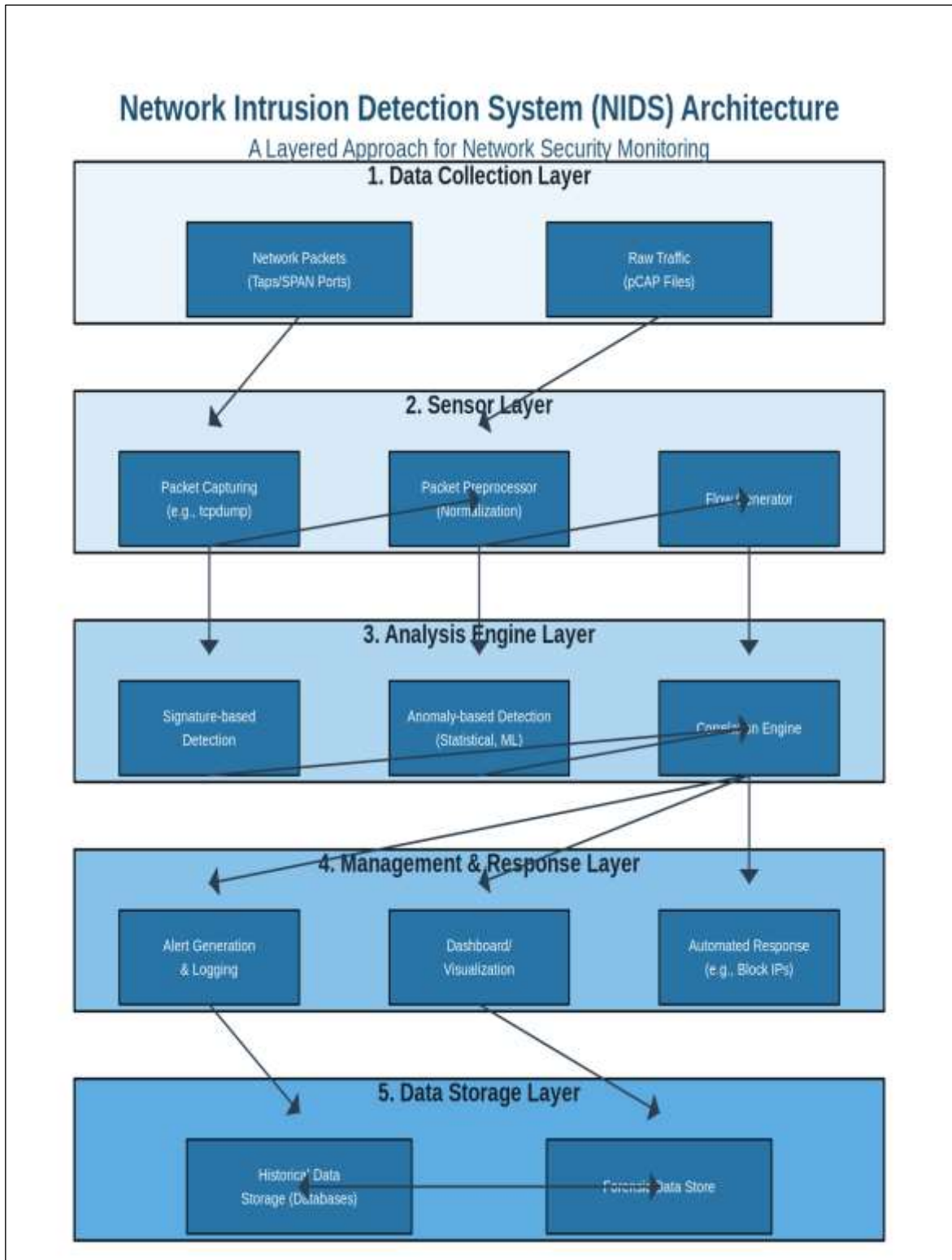


Figure 2.1: Shows the Layered Approach for Network Security Monitoring (by Author).

2.2.1 Evolution of Intrusion Detection Systems (IDS)

The need for Intrusion Detection Systems (IDS) was necessitated by the increased level of sophistication of cyberattacks. Denning (1987) proposed the first IDS, which was rule-based. IDS evolved over the years from signature-based approaches to anomaly detection through statistical models and machine learning [14]. Figure 2.2 indicates the timeline of the development of the IDS. A standard Intrusion Detection System (IDS) can be defined mathematically as a function

$D(x)$, which determines if an input event x is malicious or normal:

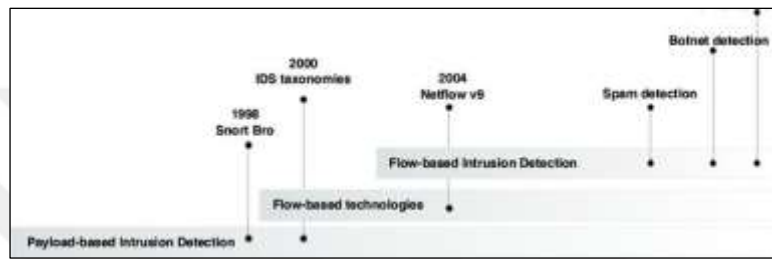


Figure 2.2: The Timeline of IDS Development (By Author)

$$D(x) = \begin{cases} 1, & x \text{ is an attack} \\ 0, & x \text{ is normal} \end{cases} \quad (2.1)$$

This function evolved over time to create four broad categories:

- i. Signature-Based IDS – Functions on the basis of known attack signatures.
- ii. Statistical Anomaly-Based IDS – It detects anomalies using probability models.
- iii. Entropy-Based IDS – Uses information theory for measuring the randomness of traffic.
- iv. Deep Learning-Based IDS – Makes use of artificial neural networks to detect anomalies.

An Intrusion Detection System (IDS) is a critical security feature that operates by methodically scanning, analysing, and responding to malicious or unauthorized activity within a networked environment. As illustrated in Figure 2.3, the IDS architecture comprises six interdependent modules, each serving a distinct function in the intrusion detection chain. Below is a straightforward decomposition of these components and their interdependencies:

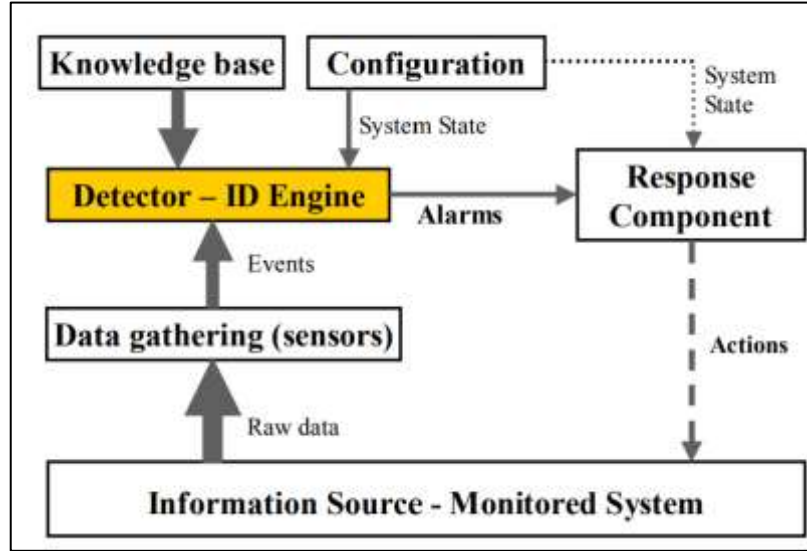


Figure 2.3: The IDS Architecture Comprises Six Interdependent Modules [14]

2.2.1.1 Mathematical Foundations of IDS

i. Signature-Based IDS: Rule-Based Matching

A signature-based IDS maintains a collection of known attacks, represented by the set S [15], Below is a flowchart illustrating in the figure 2.3 of the Signature-Based IDS: Rule-Based Matching process for your thesis. It graphically illustrates the step-by-step logic of how the IDS detects attacks based on pre-defined signatures. The detection function is:

$$D(x) = \begin{cases} 1, & \text{if } x \in S \\ 0, & \text{otherwise} \end{cases} \quad (2.2)$$

where x is an event in a network (e.g., a packet, a connection, or a log message).

Since looking up an attack signature is a lookup operation, the detection time complexity is:

$$T(n) = O(1) \quad (2.3)$$

ii. Statistical Anomaly-Based IDS: Probabilistic Detection

Anomaly-based IDS models normal traffic behavior using probability distributions. Let $X = \{x_1, x_2, \dots, x_n\}$ be a set of normal network feature values [15]. The following flowchart provides a clear visual representation of the IDS detection process, as shown in Figure 2.4. Normal traffic is commonly assumed to follow a Gaussian (Normal) Distribution:

where:

The normal network traffic, μ is the mean of:

$$\mu = \frac{1}{n} \sum_{i=1}^n x_i \quad (2.4)$$

σ^2 is the variance, given by:

$$\sigma^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2 \quad (2.5)$$

To determine whether an event x is an anomaly, we compute the Z-score:

$$Z = \frac{x - \mu}{\sigma} \quad (2.6)$$

A traffic event is classified as anomalous if:

$$|Z| > Z_{\alpha/2} \quad (2.7)$$

where $Z_{\alpha/2}$ is the critical threshold from the standard normal table.

iii. Entropy-Based IDS: Information Theory Approach

The use of Shannon's entropy to measure randomness in network traffic, which is expressed as:

$$H(X) = - \sum_{i=1}^n P(x_i) \log P(x_i) \quad (2.8)$$

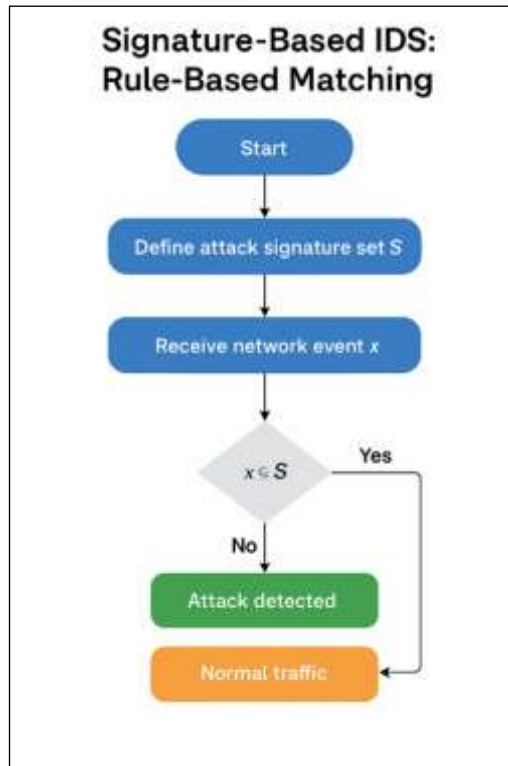


Figure 2.4: Flowchart illustrating in the Signature-Based IDS (By Author)

where:

$P(x_i)$ is the probability of the occurrence of a network feature x_i .

Greater entropy signifies regular traffic, while lesser entropy implies organized (possibly malicious) behaviour.

2.2.2 Importance and Relevance of IDS Detection

Intelligent Decision Systems (IDS) play a very significant role in situations where decisions are to be made timely, correct, and based on data. They are extremely applied in several areas such as cybersecurity, healthcare, financial markets, and autonomous systems, where decision-making is advanced and perilous. The detection capability of such systems helps in identifying all abnormal or possibly risky scenarios in real-time and resolving them accordingly, which is the secret of the reliability and security of modern intelligent systems [16]. In order to be able to accurately interpret detection value through the application of IDS, the detection process, the mathematical underpinning of the same, and how it supports decision-making in real time will be covered.

2.2.2.1 Mathematical Foundations of IDS Detection

IDS detection relies to a great extent on statistical models and machine learning algorithms for the identification of data patterns that are normally or abnormally behaving [17]. Common mathematical models IDS detection borrows include the following:

- i. Statistical Decision Theory: IDS applies decision theory to classify observations into pre-specified classes (e.g., normal or anomalous). The most basic method is the likelihood ratio test, where one makes a decision based on the probability of a data point x belonging to one class or another. The likelihood ratio is:

$$\Lambda(x) = \frac{P(x|C_1)}{P(x|C_2)} \theta \quad (2.9)$$

Where:

$P(x | C_1)$ and $P(x | C_2)$ are the likelihoods that data point x is in class C_1 (normal) or class C_2 (anomalous).

θ is a threshold that differentiates whether x is considered normal or anomalous.

- ii. Classification and Detection: Real systems utilize machine learning algorithms (i.e., Support Vector Machines, Decision Trees, and Neural Networks) to classify data into these classes [18]. The goal is to reduce the classification errors, which can be expressed mathematically as:

$$P(\text{"Error"}) = P(\text{"False Positive"}) + P(\text{"False Negative"}) \quad (2.10)$$

Where a False Positive is an event that has been wrongly characterized as anomalous, and a False Negative is an event that has been wrongly characterized as normal.

- iii. Detection Neural Networks: Neural networks work best in the detection of complicated patterns that lesser models would have overlooked [19]. Neural networks are tasked with adjusting the system weights to their optimum values for reducing classification error. The neural network's training error function is usually best expressed as:

$$E(w) = \sum_{i=1}^N (y_i - f(x_i, w))^2 \quad (2.11)$$

Where:

y_i is the true class label for the sample i , $f(x_i, w)$ is the output of the neural network a forecast for the input data x_i , the weights of the neural network.

2.2.3 Objectives of IDS Detection

The major objective of Intelligent Decision System (IDS) detection is to deliver timely, precise, and autonomous anomaly detection, inconsistencies, or threats within data-driven dynamic systems. As systems in use become increasingly data-intensive and interdependent, reliance on intelligent processes with the ability to handle massive volumes of data and make decisions with minimal human intervention has grown exponentially. IDS detection processes intend to enhance such intelligent systems' credibility, security, and overall decision-making capability.

- i. Accurate Anomaly and Threat Identification

One of the main goals of IDS detection is to successfully distinguish between benign and suspicious patterns in input data streams. It includes detection of malicious activity (e.g., cyber-attacks), faults (e.g., system or sensor faults), or out-of-pattern occurrences (e.g.,

medical anomalies). Owing to the nature of real systems, the generalization of traditional rule-based detection models becomes almost impossible. IDSs utilize advanced learning algorithms that learn to improve pattern detection and classification accuracy with time [20]. In Figure 2.4 shows that Threat identification and anomaly detection are two classification processes in the suggested method, which uses the cascade model.

For example, in cyber defenses, IDS detection systems try to uncover hidden attack patterns by learning past patterns of network traffic. The systems comprise low false positives with a capability to react very rapidly to actual threats, which is crucial for maintaining system resilience in real-time [21].

2.3 IDS DETECTION TECHNIQUES

Intrusion Detection Systems (IDS) are essential network security systems that monitor and examine network traffic for unusual activity. These vary from unauthorized access to complex cyberattacks. IDS detection methods can be divided into a plethora of methods, such as traditional, machine learning, statistical, hybrid, and real-time adaptive systems. In this research, we have provided elaborate explanations of all detection methods, including flowcharts to illustrate the process in Figure 2.5.

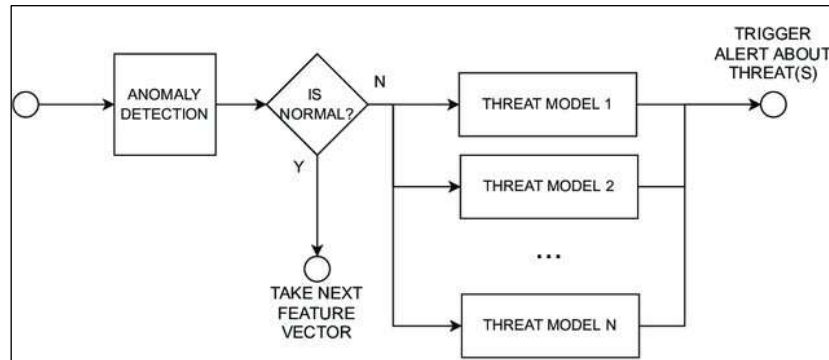


Figure 2.5: Shows Threat Identification and Anomaly Detection Are Two Classification Processes in The Suggested Method, Which Uses the Cascade Model [20].

Figure 2.6 shows how the ROC curve assesses the trade-off between false alarm rate and detection probability. An improved IDS is shown by an ROC that is closer to the left upper corner of the graph point which represents 0% false alarm and 100% detection.

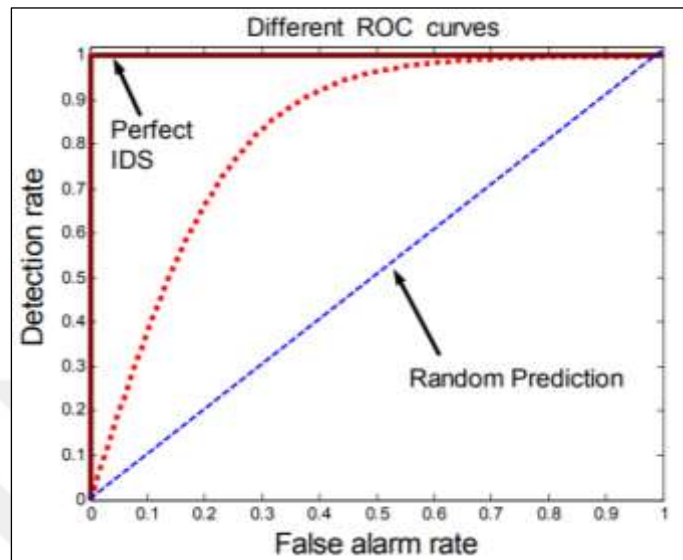


Figure 2.6: Shows How the ROC Curve Assesses the Trade-Off Between False Alarm Rate and Detection Probability [22].

As illustrated in Figure 2.7, Intrusion Detection Systems (IDS) can be systematically classified based on five general criteria. The first and most basic of these is the information source, which determines the type of data being monitored by the IDS. This is a classification that is basic in the sense that it directly affects the system's potential for detection, method of deployment, and range of operation. Intrusion Detection Systems intrinsically classify according to the origin of information that controls their monitoring and detection scope [22]. Host-based IDS solutions control system information in the shape of security event logs, such as Windows Event Logs and auth.log on Linux, system audit logs, such as audited, and file integrity through the use of checksum checking agents like Tripwire. They are good at identifying privilege escalation and insider attacks through monitoring of process run-time behaviour and registry modifications, including the improper use of command shells by security software. But as host-based solutions, they are oblivious to network-based attacks such as port scanning and can be a burdensome performance load on monitored systems. Alternatively, network-based IDS deployments inspect unprocessed packet contents captured by libraries like libpcap, communication endpoint-based NetFlow data, and

protocol-level problems in HTTP or DNS. This approach is highly effective against volumetric threats like DDoS campaigns and worm spread, and is passive and does not require host installation. These products do not have good performance in handling encrypted traffic and cannot support man-in-the-middle attacks and produce false positives in high network conditions. Application log analysis IDS solutions are great at analyzing structured representations of web server access logs, database audit trails, and proprietary application events, such as low-level detection of session hijacking and business logic abuse, as a compromise for being unable to detect exploits at lower levels of the operating system or at the network level. Hybrid approaches overcome these limitations by correlating activity across application, network, and host planes with products such as Splunk in order to support advanced pattern-of-attack detection such as the correlation of brute force authentication probes with the follow-up command-and-control behavior, though this end-to-end visibility

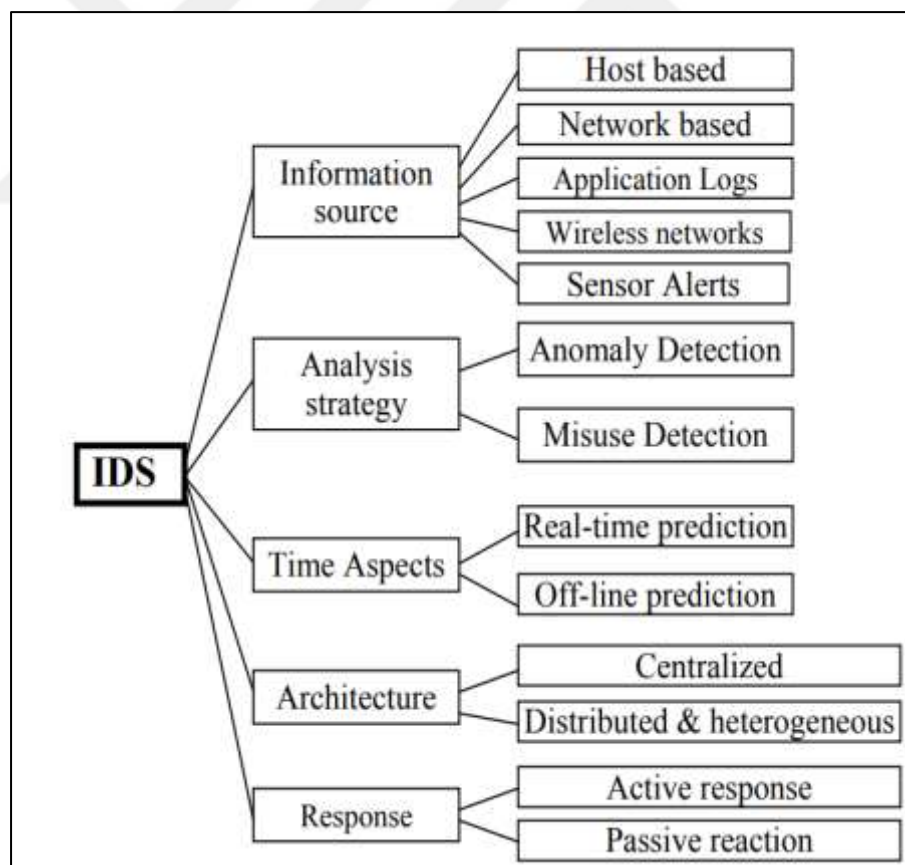


Figure 2.7: Illustrates Intrusion Detection Systems (IDS) Can Be Systematically Classified Based on Five General Criteria [23].

comes at the cost of increased deployment complexity and tuning that can overwhelm operations personnel [23].

2.3.1 Signature-Based Detection

Intrusion detection systems naturally divide by the source of information used to define their monitoring and detection scope. Host-based IDS products monitor system data by looking at security event logs like Windows Event Logs and `auth.log` on Linux, system audit logs like `audit`, and file integrity through checksum checking tools like Tripwire. They are effective in the detection of privilege escalation and insider attacks through the monitoring of process run-time behavior monitoring and registry modifications, like the abuse of command shells by security software. However, as they are host-based systems, they cannot detect network-based attacks like port scanning, which impose significant performance loads on the monitored systems. Instead, network-based IDS solutions analyse raw packet data collected with tools like libpcap, endpoint communication data like NetFlow, and Hypertext Transfer Protocol (HTTP) or Domain Name System (DNS) protocol-level issues [24]. This approach is highly effective for volumetric attacks like Distributed Denial-of-Service (DDoS) Attack campaigns and worm propagation; it is also passive and installation-free on hosts. These solutions have poor performance in handling encrypted traffic and cannot handle man-in-the-middle attacks, and produce false positives in heavy network conditions. Application log analysis IDS solutions are best at analysing structured information in web server access logs, database audit logs, and some application events, including detecting session hijacking and business logic misuse, but cannot detect problems at lower operating system or network levels. Hybrid techniques can address such issues by associating activity among applications, networks, and hosts with tools like Splunk that can identify patterns of advanced attacks, such as correlating brute force login attempts with subsequent command-and-control activities, but at the cost of making setup and tuning more difficult for operations staff. Hybrid solutions respond to these limitations by correlating activity across application, network, and host planes with products such as Splunk to support advanced pattern-of-attack detection such as the correlation of brute force authentication probes with ensuing command-and-control activity, but one that sacrifices end-to-end visibility for the costs of additional deployment complexity and tuning that overwhelm operations personnel [25]. The

signature-based detection procedure is a clearly defined process [26], Capture and Monitor Network Traffic – The IDS continues to monitor and capture system logs or network packets.

- i. Extract Features from Captured Data – The features of interest (e.g., payloads, packet headers) are extracted to be analysed.
- ii. Compare with Known Signatures – The features are compared with the signature database.
- iii. Detection Decision
 - a. If a match is found, an alert is issued, indicating a potential breach.
 - b. If a match is not created, the system continues to monitor.

2.3.2 Anomaly-Based Detection

The increasing complexity of cyberattacks necessitated the creation of a paradigm change from traditional signature-based intrusion detection systems. Anomaly-based intrusion detection systems (A-IDS) are now an inevitable choice so far as new and complex cyberattacks detection is concerned, on the basis of the detection of any anomaly from usual patterns of usual system behaviour [27]. In contrast to signature-based systems that rely solely on known attack signatures, A-IDS employs advanced statistical and machine learning techniques to identify potentially malicious activity, and is therefore better at identifying

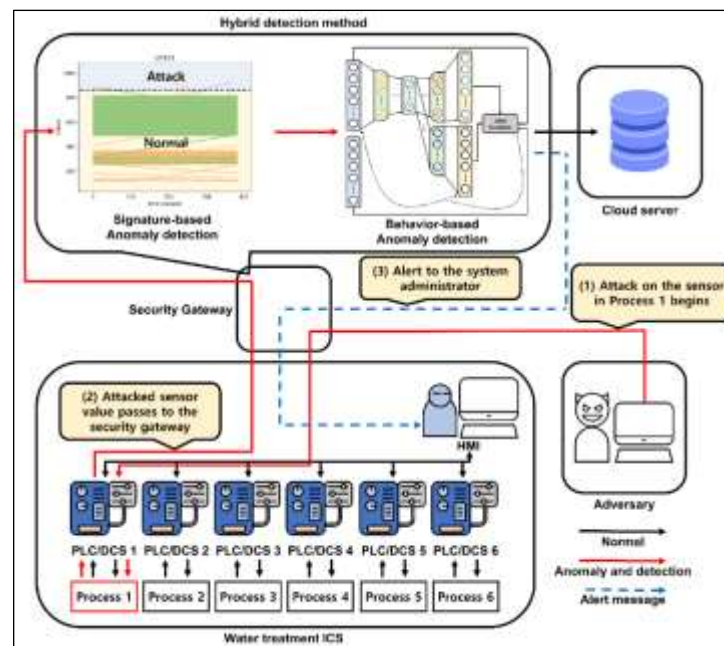


Figure 2.8: Shows the Signature-Based Detection Process [26].

zero-day attacks and APTs [28]. As illustrated in Figure 2.8 shows the Signature-Based Detection Process.

Advanced A-IDS implementations are generally comprised of three dominant methodological frameworks: statistical models, machine learning techniques, and deep learning paradigms. Statistical methods such as Gaussian Mixture Models and Kolmogorov-Smirnov tests establish probability distributions of normal system activity as baselines against which anomalies are identified [29]. Machine learning methods such as Isolation Forests and One-Class Support Vector Machines enable more sophisticated detection in high-dimensional feature spaces [30]. Recent advances in deep learning, such as applying autoencoders and long short-term memory (LSTM) networks, have provided significant help toward the discovery of intricate temporal patterns and anomalous behavior [31]. Use of effective A-IDS solutions involves a range of technical challenges to be solved by continuing research efforts. These issues include high-dimensional feature space, overall false alarms in complicated environments, and computationally expensive analysis in real-time [32]. In recent years, solutions including learning algorithms in online learning, automatic feature selection schemes, and threshold adaptation mechanisms have been promoted as panaceas to overcome such constraints [33]. In addition, the introduction of explainable AI techniques allowed improving the interpretability of A-IDS decisions, which is a major SOC analyst's requirement [34]. Figure 2.9, shows Anomaly-Based Intrusion Detection System Architecture; the architecture has four basic components: (1) a data ingestion layer that merges network traffic, system logs, and application events; (2) a preprocessing module for feature extraction and normalization; (3) an anomaly detection engine based on statistical

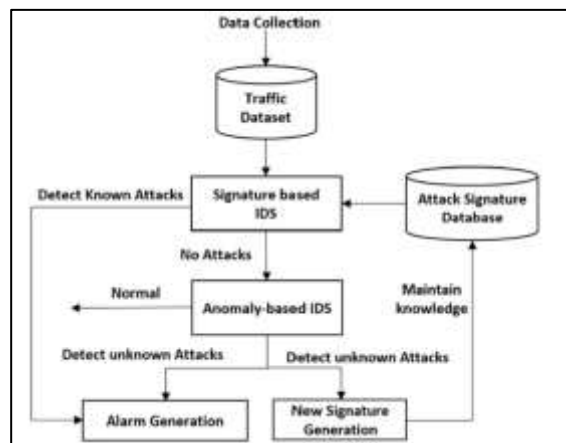


Figure 2.9: Shows Anomaly-Based Intrusion Detection System Architecture [34]

models or machine learning to identify deviations; and (4) an alerting/response system that triggers mitigations. The system operates iteratively, with behavioural baselines refreshed continuously to stay ahead of emerging threats while minimizing false positives through adaptive thresholding.

The task of anomaly detection can be mathematically formulated as follows:

Let:

$X = \{x_1, x_2, \dots, x_n\}$ represent the dataset of observed system behavior.

μ and σ denote the mean and standard deviation of normal behavior.

T be a threshold that determines the deviation limit.

Anomaly detection can be modeled using the Gaussian (Normal) Distribution:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (2.12)$$

An observation x is considered an anomaly if:

$$|x - \mu| > T\sigma \quad (2.13)$$

Otherwise, using Machine Learning, a classifier for anomaly detection can be trained to optimize the anomaly score function:

$$A(x) = \sum_{i=1}^n |x_i - \mu|^2 \quad (2.14)$$

Whereas the larger value for $A(x)$ indicates the greater likelihood of the anomaly

2.4 MACHINE LEARNING AND DATA MINING IN IDS DETECTION

A core aspect of modern digital infrastructure security, especially of cloud computing, Internet of Things (IoT), and industrial control systems (ICS), is intrusion detection systems (IDS). As cyber-attacks evolve on a daily basis, conventional signature-based Intrusion Detection Systems, even though skilled in identifying known threats, are of no use in detecting emerging or zero-day attacks. Consequently, Data Mining (DM) and Machine Learning (ML) have emerged as foundational technologies for the development of adaptive, intelligent, and scalable Intrusion Detection Systems (IDS) that learn from extensive data and generate novel intrusion patterns [35]. This section addresses the widespread implementation of machine learning and data mining methodologies in intrusion detection system design. This section addresses theoretical foundations, algorithmic approaches,

mathematical formulations, performance assessment criteria, and related challenges in the implementation of intrusion detection systems utilizing machine learning.

Machine Learning is an algorithm that are able to find patterns in data and make informed guesses or decisions. ML models used in IDS are typically statistical learning theory and optimization. IDS problems are generally formulated as a classification problem such that the system labels network connections as benign or malicious.

Let $X = \{x_1, x_2, \dots, x_n\}$ be a collection of network observations, and each $x_i \in R^d$ be a d-dimensional feature vector of source IP, port number, protocol type, and packet size. In supervised learning, every observation x_i is accompanied by a label $y_i \in Y$, $Y = \{0,1\}$, representing normal and anomalous traffic, respectively. The learning algorithm tries to discover a hypothesis function $f: R^d \rightarrow \{0,1\}$ that generalizes to predict unseen data with low classification error. As shown in Figure 2.10, Classification of Machine Learning Methods for IDS Detection

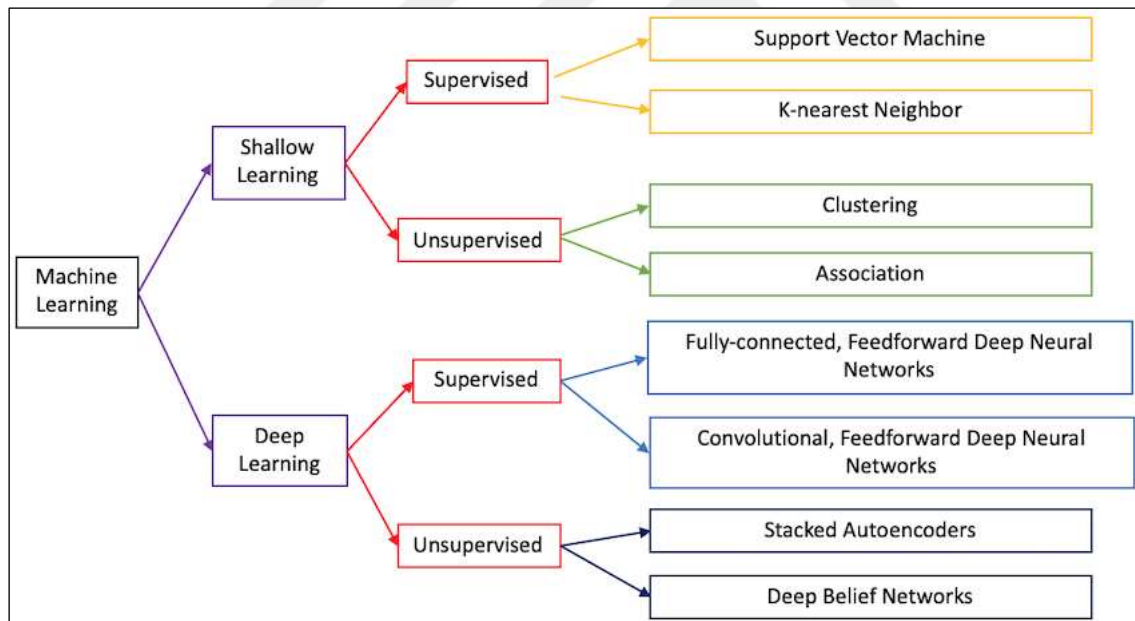


Figure 2.10: Shows Classification of Machine Learning Methods for IDS Detection [36]

2.4.1 Supervised Learning Models

Supervised learning is the popular pattern of machine learning where models are trained on tagged data to foresee the outcome or classify data. Supervised learning models, mathematical concepts, and applications are discussed in this chapter.

2.4.1.1 Support Vector Machine (SVM)

Support Vector Machines (SVMs) are effective supervised learning algorithms widely applied to classification issues in Intrusion Detection Systems (IDS). They are most valued for their efficiency in handling high-dimensional data and for the strong theoretical framework within statistical learning theory [37]. Figure 2.11 illustrates an SVM-based IDS to classify normal and anomalous network queries. The SVM constructs the most optimal possible hyperplane to maximally distinguish data into two classes. For the given data (x_i, y_i) , the SVM solves the following optimization problem:

$$\min_{w,b,\xi} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i \text{ subject to: } y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \quad (2.15)$$

Here, ξ_i are slack variables and C is the penalty parameter.

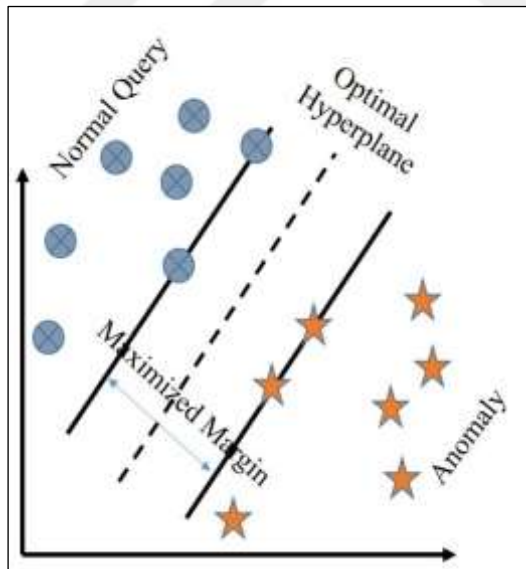


Figure 2.11: Shows SVM Model for IDS (By Author)

2.4.1.2 Decision Trees in IDS Detection

Decision trees are a supervised learning model, which in the majority of cases is applied for classification in intrusion detection systems (IDS). They recursively divide the feature space into disjoint subsets by employing a split criterion that maximally differentiates the classes.

The tree-form structure that they yield is one in which each internal node is a test on an attribute, each branch is the outcome of the test, and each leaf node is a class label.

Mathematically, the decision tree construction is the choice of those features that partition the data in the best way. The most popular attribute selection measure is Information Gain (IG), which is based on the concept of entropy. Entropy is used to express the impurity or uncertainty of a data set. Let a data set D with k classes to be represented. Let the entropy $H(D)$ be given as

$$H(D) = - \sum_{i=1}^k p_i \log_2 p_i \quad (2.16)$$

where p_i is the relative frequency of examples in D belonging to the class i . If the set D is divided into subsets $D_1, D_2, \dots, D_v$ based on an attribute A with v possible values, then the expected value of the entropy after division is:

$$H(D, A) = \sum_{j=1}^v \frac{|D_j|}{|D|} H(D_j) \quad (2.17)$$

The information gain from using the attribute A is then:

$$IG(D, A) = H(D) - H(D, A) \quad (2.18)$$

The resulting model can be represented as a collection of understandable and computationally basic decision rules, so decision trees are suitable for real-time Intrusion Detection Systems (IDS). Pruning and ensemble techniques, e.g., random forests, can reduce overfitting vulnerability. Decision trees are used in IDS against known and unknown attacks due to their transparency and efficacy. Decision trees can be learned with labeled training sets for network traffic to classify events into malicious and benign categories, and therefore, they are crucial components of most intelligent detection mechanisms [1]. Figure 2.12 depicts a decision tree-based IDS for discriminating between normal and anomalous network queries.

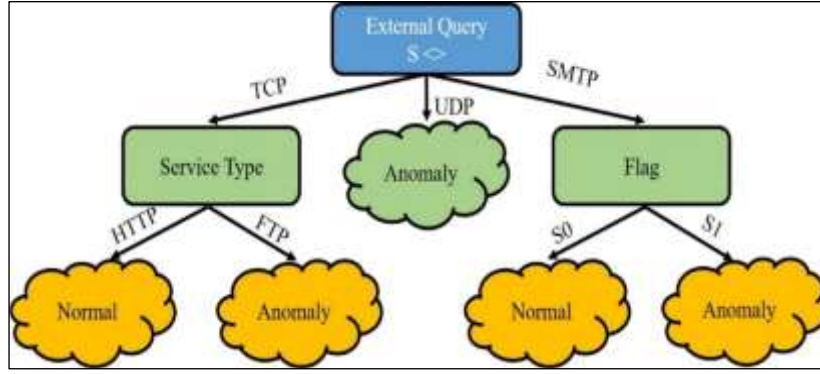


Figure 2.12: Depicts A Decision Tree-Based IDS For Discriminating Between Normal and Anomalous Network Queries (By Author)

2.4.1.3 Artificial Neural Networks (ANN) in IDS Detection

The function and organization of biological brain operations can inspire a class of data-driven models known as artificial neural networks (ANNs). ANNs are good learning models to learn complex, non-linear patterns from patterns of network traffic to support Intrusion Detection Systems (IDS) in identifying normal and abnormal behavior.

An input layer, several hidden layers, and an output layer form an artificial neural network (ANN), which is an interconnected set of nodes (neurons). Each neuron implements a non-linear activation function, applies a bias, and implements a weighted sum of inputs. The output y of some neuron in an output or hidden layer can be quantitatively expressed as:

$$y = \phi(\sum_{i=1}^n w_i x_i + b) \quad (2.19)$$

where x_i is input features, w_i input feature weights, b is the bias term, and $\phi(\cdot)$ is an activation function such as the sigmoid, hyperbolic tangent, or Rectified Linear Unit (ReLU). For instance, the sigmoid function is given by:

$$\phi(z) = \frac{1}{1+e^{-z}} \quad (2.20)$$

During training, the network learns optimum weights and bias by minimizing a loss function L , often the mean squared error (MSE) for regression problems or cross-entropy for classification problems. Cross-entropy loss for any training set of m examples with target values $y^{(i)}$ and predicted values $\hat{y}^{(i)}$ is defined as:

$$\mathcal{L} = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})] \quad (2.21)$$

The parameters are tuned using backpropagation and gradient descent. Backpropagation is a technique for calculating the gradient of the loss function with respect to each parameter w_{ij} by chaining the rule of calculus layer-wise, and updates are as:

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial \mathcal{L}}{\partial w_{ij}} \quad (2.22)$$

where η is the learning rate.

ANNs are particularly suitable for IDS since they are capable of generalizing from high-dimensional, large input spaces and uncovering complex hidden relations between the features. They are therefore particularly well-suited to signature-based detection of known attack signatures and zero-day attacks. But they require much training data as well as processing power. In actual deployment of IDS in the real world, ANN models such as MLPs as well as CNN, have demonstrated good detection rates on a variety of intrusion datasets [39]. An IDS based on KNN is shown in Figure 2.13 to distinguish between typical and unusual network requests.

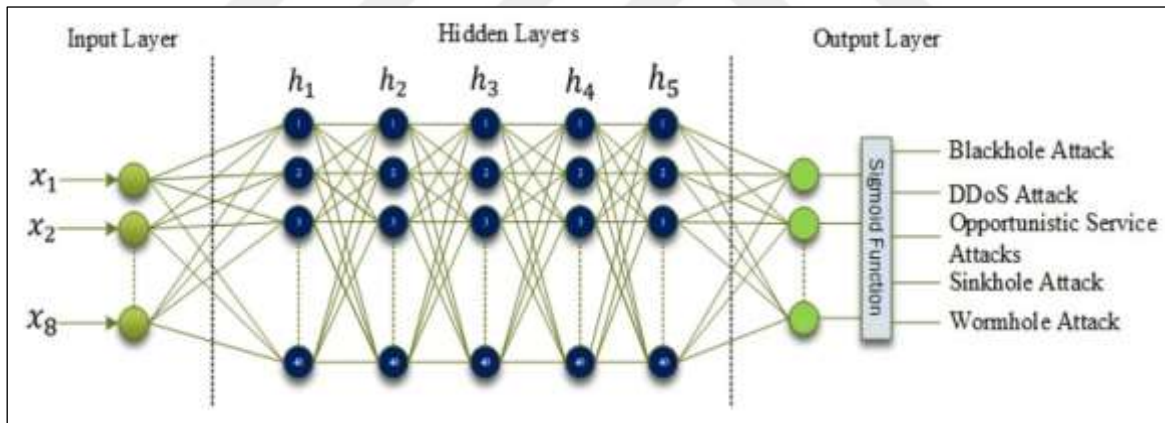


Figure 2.13: Shows ANN-based Model for IDS (By Author)

2.4.1.4 Naïve Bayes Classifier in IDS Detection

Naïve Bayes classifier is an algorithm for learning Bayes' theorem probabilistically. Naïve Bayes classifier performs superbly in terms of intrusion detection because it is extremely lightweight, scalable, and stable when working with high-dimensional data. The model makes an assumption of feature conditional independence under the class label and is computation-wise light, and thus enables incredibly rapid classification of network traffic

[41]. Figure 2.14 displays a Naive Bayes-based IDS for distinguishing between typical and unusual network requests.

Bayes' theorem provides the foundation of the Naïve Bayes approach, expressing the posterior probability $P(C_k | x)$ of a class C_k Given an input vector $x = (x_1, x_2, \dots, x_n)$ as:

$$P(C_k | x) = \frac{P(x|C_k)P(C_k)}{P(x)} \quad (2.23)$$

$$\hat{C} = \operatorname{argmax}_{C_k} P(C_k) \prod_{i=1}^n P(x_i | C_k) \quad (2.24)$$

where $\mu_{(k,i)}$ and $\sigma_{(k,i)}^2$ are the mean and variance of the feature x_i for class C_k , estimated from training data.

$$P(x_i | C_k) = \frac{1}{\sqrt{2\pi\sigma_{k,i}^2}} \exp\left(-\frac{(x_i - \mu_{k,i})^2}{2\sigma_{k,i}^2}\right) \quad (2.25)$$

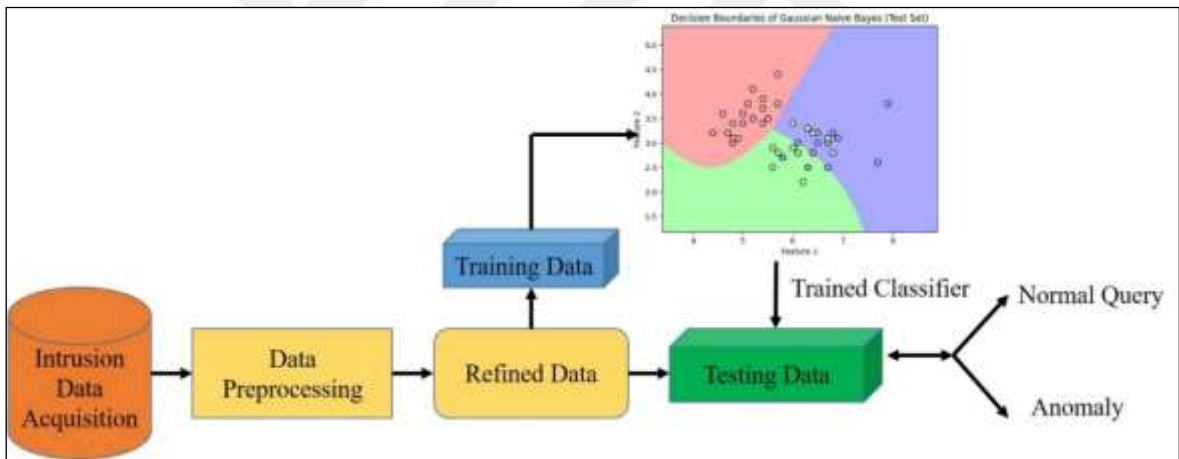


Figure 2.14: Displays A Naive Bayes-Based IDS For Distinguishing Between Typical and Unusual Network Requests (By Author)

2.4.2 Unsupervised Learning IN IDS

The unsupervised learning approach presumes no labeled data. Unsupervised learning approaches, like K-means and DBSCAN, cluster the traffic pattern based on similarity [41]. This approach may be able to identify anomalies in traffic, possibly a symptom of a DDoS attack. In contrast, unsupervised methods are generally ineffective on high-dimensional data and sometimes present difficult-to-interpret clusters. Apart from these, the success of

unsupervised learning depends very much on selecting its features, which vary quite a lot for different network environments.

2.5 DEEP LEARNING TECHNIQUES IN IDS

Deep learning models, such as CNNs and RNNs, have attracted particular interest in DDoS detection since these models can extract features from raw data automatically.

2.5.1 Convolutional Neural Networks (CNN) In IDS

The CNNs can process structured data and look into network traffic patterns. The CNNs can learn the hierarchies of features automatically, thereby making it possible to detect intricate patterns linked to DDoS attacks. Moreover, CNNs process large volumes of data without affecting speed, an essential issue during a DDoS event because traffic volume is massively large. They can easily differentiate between benign and malicious traffic since they can identify intricacies in the data [42]. Figure 2.15 displays a CNN-based IDS for distinguishing between typical and unusual network requests.

CNNs effectively extract local spatial relationships in structured network data. Let $X \in R^{(n \times d)}$ Note the input matrix, and n denotes the number of flows or packets, and d denotes the number of features. A 1D convolution with a kernel $W \in R^{(k \times d)}$ and bias b gives:

$$H_i = \phi\left(\sum_{j=0}^{k-1} W_j \cdot X_{i+j} + b\right), \quad \forall i \in \{1, \dots, n - k + 1\} \quad (2.26)$$

In which $\phi(\cdot)$ is an example of non-linear activation, e.g., *ReLU*. The pooling function comes after the convolution:

$$P_i = \max_{j=1}^p H_{i+j} \quad (2.27)$$

This dimensionality reduction is achieved without compromising important features. Features output are subsequently fed into dense layers for the purpose of classification.

2.5.2 Recurrent Neural Networks (RNN) IN IDS

RNNs, particularly Long Short-Term Memory networks, find their most proper application in handling sequential data and, more particularly, handling temporal dependencies. LSTMs can model time series data to detect DDoS attacks. It is pretty helpful in better distinguishing unusual patterns over time relative to the evolution of attack behavior. It would enable an early alert mechanism, as trends and deviations in traffic could be analyzed for RNNs [42].

RNNs are particularly well-suited for time data such as traffic accident sequences. For a sequence $\{x_1, x_2, \dots, x_T\}$, the hidden state is recursively updated as:

$$h_t = \sigma(W_x x_t + W_h h_{t-1} + b) \quad (2.28)$$

$$\hat{y}_t = \text{softmax}(W_y h_t + c) \quad (2.29)$$

Where $\sigma(\cdot)$ is an activation function such as tanh, and $\hat{y}_t$ is the probability of a certain class at time t .

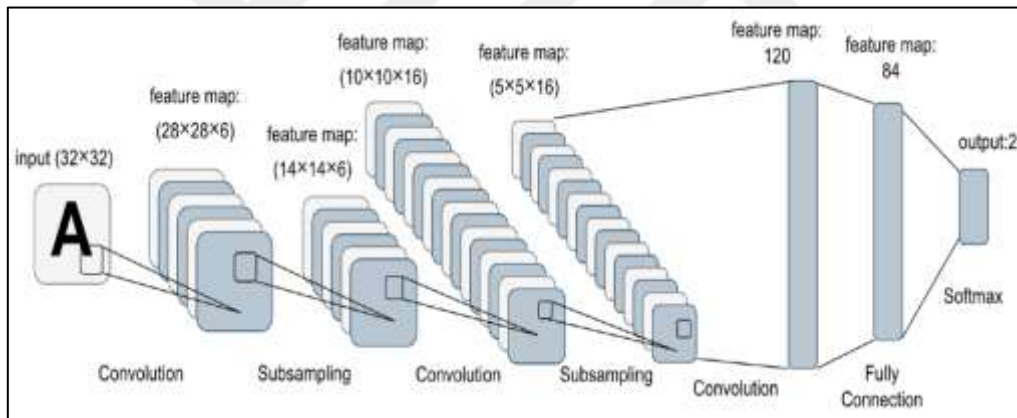


Figure 2.15: Displays A CNN-Based IDS For Distinguishing Between Typical and Unusual Network Requests [42].

2.5.3 Autoencoders (AEs) In IDS

The anomalies present in the DDoS traffic can be detected using autoencoders. Autoencoders are trained on known standard patterns of traffic so that they can learn how to reconstruct input data. Deviations from the reconstructed one might represent anomalies that could represent a DDoS attack. It uses an unsupervised approach, allowing it to adapt to changes in the traffic pattern. Hence, it becomes a dynamic way of discovering new strategies for

attacks [43]. Figure 2.16 displays Autoencoders (AEs) -based IDS for distinguishing between typical and unusual network requests.

Autoencoders provide a robust unsupervised approach to anomaly detection in IDS. With the input $x \in \mathbb{R}^d$, the encoder projects it to the latent representation $z \in \mathbb{R}^k$: $z = \phi(W_e x + b_e)$

The decoder reconstructs the original input as:

$$\hat{x} = \phi(W_d z + b_d) \quad (2.30)$$

The model minimizes the reconstruction loss:

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^n \|x^{(i)} - \hat{x}^{(i)}\|^2 \quad (2.31)$$

A threshold θ is put on the reconstruction error for network traffic anomaly detection

$$S(x) = \|x - \hat{x}\|_2^2 \quad \text{with} \quad \text{Label}(x) = \begin{cases} \text{Attack,} & \text{if } S(x) > \theta \\ \text{Normal,} & \text{otherwise} \end{cases} \quad (2.32)$$

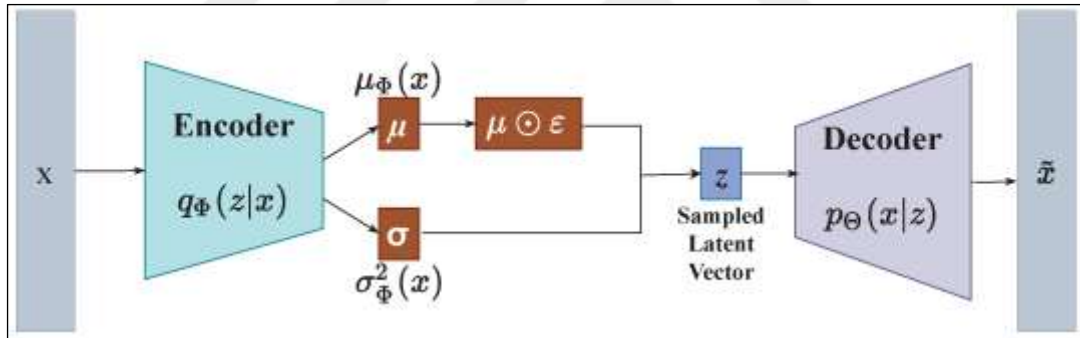


Figure 2.16: Displays Autoencoders (AES) -Based IDS For Distinguishing Between Typical and Unusual Network Requests [43].

2.5.4 Long Short-Term Memory (LSTM) IN IDS

LSTMs are designed to overcome the pitfalls of standard RNNs. As mentioned above, they often suffer from vanishing and exploding gradient problems. The memory cell in LSTMs is the innovation that allows the network to remember information over long sequences while selectively forgetting irrelevant information [44]. Figure 2.17 depicts a typical Long Short-Term Memory (LSTM) architecture, particularly well-adapted to intrusion detection. The encoding is typically seeded by an input layer that processes sequential network traffic data each input instance is composed of features such as packet size, protocol, and timing

features. They are passed into one or more stacked LSTM layers, in each of which the LSTM cell contains three primary gates:

- i. Forget Gate: This gate determines what to forget about the information reaching the memory cell. It will calculate the previous hidden state and the input present to yield a value between 0 and 1, where 0 indicates complete forgetting of information, and 1 indicates total information retention.
- ii. Input Gate: It controls what new input in the memory cell should be added. As with the forget gate, it also computes the current input and previous hidden state.
- iii. Output Gate: This gate determines what information from the memory cell passes to succeeding layers. It is achieved by filtering the contents of the memory cell, whereby only relevant ones can affect the succeeding computations.

This architecture enables LSTMs to model sequences effectively, making them particularly adept at analysing time-dependent data such as network traffic.

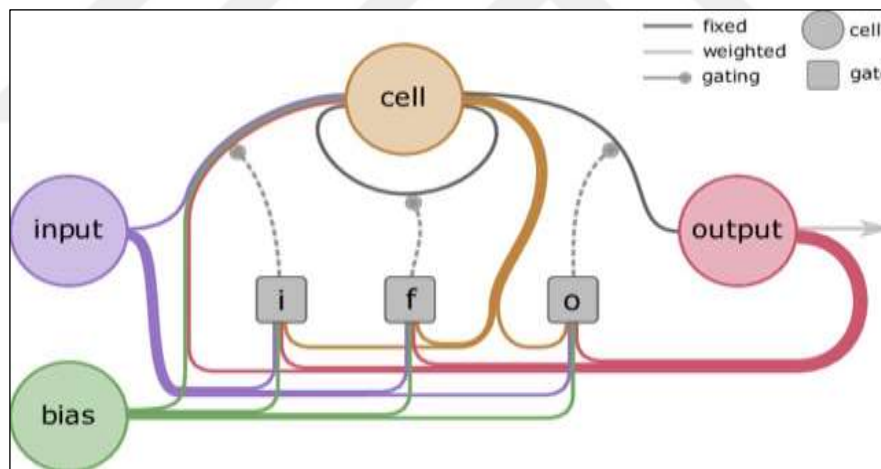


Figure 2.17: Depicts A Typical Long Short-Term Memory (LSTM) Architecture Particularly Well-Adapted to Intrusion Detection [44]

2.6 HYBRID COMPLEX MULTILAYER PERCEPTRON NETWORK

The architecture of the hybrid complex multilayer perceptron (MHHPN) network involves integrating three distinct branches: A standard Multilayer Perceptron (MLP) branch, the Dense Net-like branch, and the Resnet-like branch. All branches operate on the input data

differently, which advances the feature extraction process. The architecture of the MHHPN network is shown in Figure 2.18.

The basic structure of the MLP branch is simple, and it contains three fully connected layers with the neuron number decreasing gradually (128, 64, and 32), and the ReLU activation function is used. This branch obtains simple kinship information from the input data, such as the basic hierarchy. The MLP has been organized so that neuron counts are reduced step by step to lower the dimensionality space, but in a way that important information is retained. This organization captures more superficial patterns of kinship or relationship within the data by successively abstracting away secondary features while concentrating on primary characteristics. As the number of neurons in an MLP is reduced from one layer to the next, it compresses data into essential global features while ignoring noise details. The process is relatively straightforward and guarantees that vital attributes are identified without introducing many computations [45]. This step can be calculated as:

$$F_{MLP} = Dense_{Relu}(128, Dense_{Relu}(64, Dense_{Relu}(32, X))) \quad (2.33)$$

Where X is the original feature vector. The Dense Net-like branch is also initiated with an FC layer of 128 neurons, using concatenations of previous layer outputs as the Dense block. This branch has several cardinally packed layers with concatenations intended to advance feature reclusion and improve the gradient flow that boosts the learning. The Dense Net-like branch, which concatenates the feature maps of the previous layers, is supposed to learn detailed patterns and dependencies in the data, making it more useful for feature extraction. This step can be calculated as:

$$F_{DenseNet} = Concat([X, Dense_{Relu}(128, Concat([X, Dense_{Relu}(64, Concat([X, Dense_{Relu}(32, X)])))]))] \quad (2.34)$$

Finally, the primary Res Net-like branch incorporates a skip connection, allowing the network to learn residual mappings. Beginning with the list of parameters with a fully connected layer of 128 neurons, it also includes additions of input transformations before each dense layer (128, 64, and 32 neurons). These skip connections also tackle the vanishing gradient problem and training rate enhancement. This step can be calculated as:

$$F_{ResNet} = Add([X, Dense_{Relu}(128, Add([X,$$

$$Dense_{Relu} (64, Add ([X, Dense_{Relu} (32, X)])))])) \quad (2.35)$$

The layer-by-layer parametric framework described in Table 2.1 encapsulates the general overall picture of the deep learning model in question's structure composition. Each layer is properly planned with premeditated parameters like layer type (e.g., convolutional, pooling, dense, activation), filters or number of neurons, kernel size, stride, padding type, activation function, and regularization techniques like dropout or batch normalization. This tabular form of architecture supports a clean way of expressing hierarchical transformation of the input data during its passage through the network.

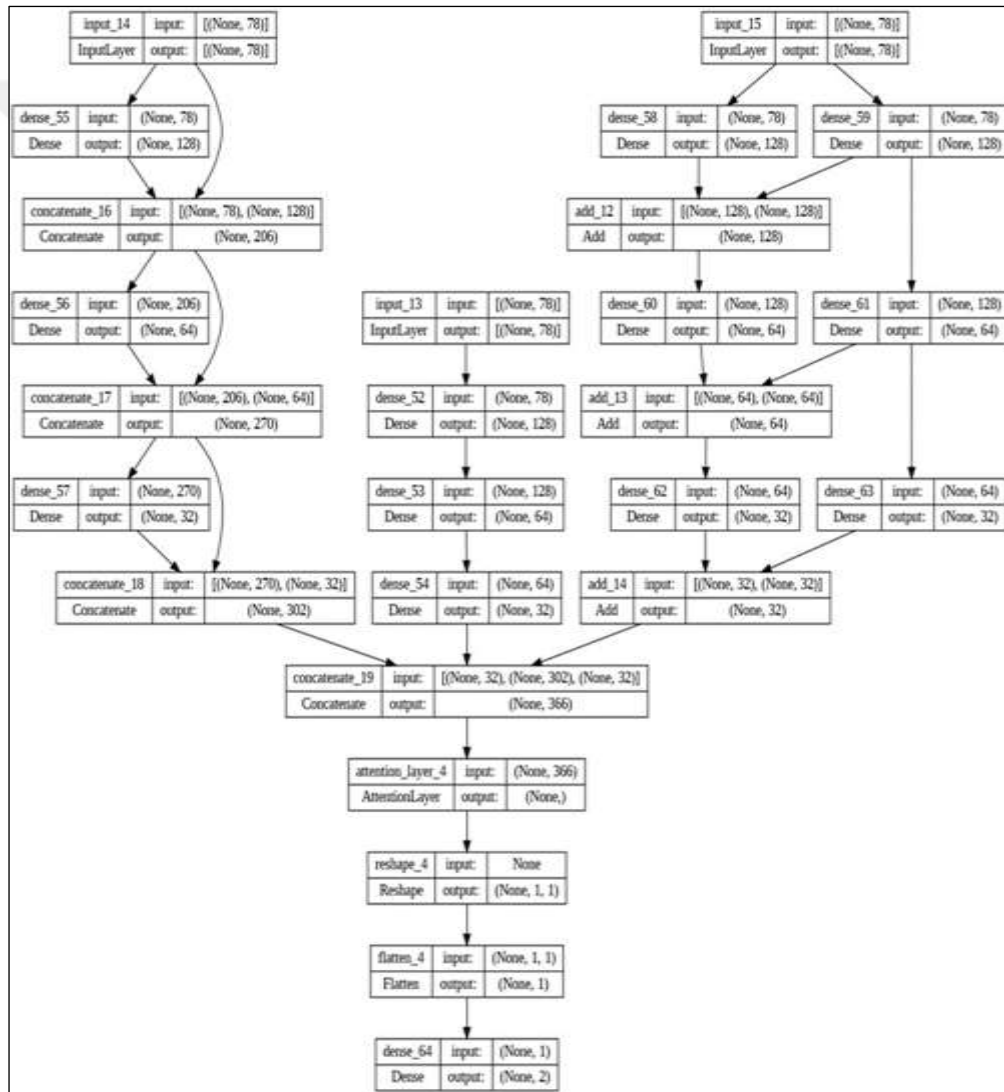


Figure 2.18: The Architecture of The Hybrid Complex Multilayer Perceptron (MHHPN) Network [45].

Table 2.1: Shows the Layer-Wise Parametric Architecture [45].

Input Layer	MLP Branch	0
Dense layer (128 units)	MLP Branch	16512
Dense layer (64 units)	MLP Branch	8256
Dense layer (32 units)	MLP Branch	2080
Input Layer	DenseNet Branch	0
Dense layer (128 units)	DenseNet Branch	16512
Dense layer (64 units)	DenseNet Branch	14656
Dense layer (32 units)	DenseNet Branch	9376
Input Layer	ResNet Branch	0
Dense layer (128 units)	ResNet Branch	16512
Transformation (128 units)	ResNet Branch	16512
Dense layer (64 units)	ResNet Branch	8256
Transformation (64 units)	ResNet Branch	8256
Dense layer (32 units)	ResNet Branch	2080
Transformation (32 units)	ResNet Branch	2080
Attention Mechanism	Combined Output	147840
Final Dense Layer (10 units)	Final Classifier	3850
Total Parameters		273778

2.6.1 Dynamic Feature Adaption Scheme IN IDS

Dynamic Feature Adaptation [46] is one of the most essential components for improving the performance of deep learning models by highlighting the critical features in each context of the input data. Such flexibility enables the model to pay more attention to essential features of the values, which enhances the model's capability to identify intricate trends and interactions. At this moment, it overcomes some shortcomings in practice, such as feature redundancy and noise, which significantly influence the reliability of predictions. The versatility of the feature representation in situations like the detection of a DDoS attack is essential for such a process since the data variety and richness require a stable and adaptable representation.

Here, the Dynamic Feature Adaptation applied to the individual deep features extracted by Perceptron, Resnet, and Dense Net branches helps fine-tune and optimize each branch's dynamic features according to their importance and relevance. Each branch processes the input data and generates a feature vector, denoted as F_{B_i} where $B_i = \{\text{Perceptron, Dense Net, and Resnet}\}$ for $i \in 1, 2, \text{ and } 3$. Let $F_{B_i} \in \mathbb{R}^{B_i}$ be the feature vector with B_i dimensions.

Further, we applied a dynamic weighting mechanism by assigning different weights to the individual features in F_{B_i} based on their relevance or importance. Here, the dynamic weight vector is computed according to

$$W_P = [W_1, W_2, \dots, W_{B_i}] \quad (2.36)$$

Where weight W_i refers to the weight for i^{th} feature in the feature vector F_{B_i} . These weights are determined using Equation (2.37).

$$W_i = \text{softmax} (W_{att} * F_i + b_{att}) \quad (2.37)$$

Here, W_{att} and b_{att} are learnable parameters of the attention mechanism and the SoftMax function confirms that the weights sum to 1, providing a probabilistic interpretation. The calculated dynamic weights are applied to the feature F_{B_i} to design an adapted feature vector $F_{B_i}^{adapted}$ according to Equation 3.38.

$$F_{B_i}^{adapted} = W_P \odot F_{B_i} \quad (2.38)$$

Here, $\odot$ denotes element-wise multiplication. Each feature in F_{B_i} is scaled by its corresponding weight in W_P . Finally, the adapted feature vector $F_{B_i}^{adapted}$ is passed through additional fully connected layers to further refine the features:

$$\text{Refine} (F_{B_i}^{adapted}) = \sigma (W_{refine} * F_{B_i}^{adapted} + b_{refine}) \quad (2.39)$$

where W_{refine} and b_{refine} are weights and bias parameters of the refinement layer, and σ is the ReLU activation function. This approach addresses feature redundancy and noise in DDoS classification through several key strategies: it computes dynamic weights through an attention mechanism that makes it focus more on relevant features, down-weighting or ignoring redundant ones. The applied attention mechanism mainly focuses on features that recognize abnormal traffic patterns and resource strains, focusing on high-impact indicators such as traffic rate metrics and connection states. In this way, it would be the adaptive selection of which features will be emphasized while cutting down on the noise and redundancy of the feature set. The elementwise feature multiplication by the weights of its corresponding feature component will have fine control over how much each feature can contribute in a very subtle way. Low-priority features will be given low weights, while their effects are negligible in the output. The flexibility is useful in real-time environments, especially when the DDoS attack pattern changes; it filters noise and adapts to data patterns that change in real time. The pseudocode of the dynamic feature adaptation is shown in Algorithm 2.1.

Algorithm 3.1. Pseudocode for Dynamic Feature Adaptation [45]

Step 1: Feature Extraction $F_p = \text{PerceptronBranch}(\text{input}_{data})$ $F_R = \text{ResNet}(\text{input}_{data})$ $F_D = \text{DenseNet}(\text{input}_{data})$ Step 2: Dynamic Feature Adaptation for Perceptron Features $W_p = \text{Compute Dynamic Weights}(F_p)$ $F_{p_adapted} = \text{Adapt Features}(F_p, W_p)$ Step 3: Dynamic Feature Adaptation for ResNet Features $W_R = \text{Compute Dynamic Weights}(F_R)$ $F_{R_adapted} = \text{Adapt Features}(F_R, W_R)$ Step 4: Dynamic Feature Adaptation for DenseNet Features $W_D = \text{Compute Dynamic Weights}(F_D)$ $F_{D_adapted} = \text{Adapt Features}(F_D, W_D)$ Step 5: Function to compute dynamic weights based Step 6: Function to adapt features by applying dynamic weights
--

2.6.2 Feature Fusion by Dynamic Attention-Weighted Feature Fusion In IDS

After extracting adaptive feature representations from these three branches, we introduced a dynamic attention-weighted feature fusion function that fuses these features semantically meaningfully [47]. Here, the mechanism encompasses multi-head attention, self-attention, and gating mechanisms, so the features' importance determines the correct weighting and computing procedure. In the fusion mechanism, for each adaptive feature set F_i (where I represents MLP, Dense Net, or Resnet), we apply multi-head attention to capture different aspects of the features:

$$\text{MultiHead}(F_i) = \text{Concat}(\text{head1}(F_i), \text{head2}(F_i), \dots, \text{headH}(F_i)) \quad (2.40)$$

where H is the number of attention heads. We calculate self-attention scores for each feature branch using a self-attention mechanism defined in Equation 3.13.

$$\text{SelfAttention}(F_i) = \text{softmax}\left(\frac{Q_i * K_i^T}{\sqrt{d_k}}\right) \quad (2.41)$$

where Q_i, K_i and d_k are the query, key, and value matrices derived from F_i , and d_k is the dimension of the key vectors. This mechanism helps in weighing the input features based on their relevance to each other. Further, we applied a gating mechanism to enhance the contribution of each feature branch. The gate for each feature F_i is computed as:

$$G_i = \sigma (W_g F_i + b_g) \quad (2.42)$$

where σ is the sigmoid function, and W_g and b_g are learnable parameters. The gating mechanism dynamically adjusts the influence of each feature based on its importance. During this gating mechanism, the dynamic attention weights are calculated using a SoftMax function to ensure that they sum up to 1:

$$\alpha_i = \frac{\exp(G_i)}{\sum_{i=1}^3 \exp(G_i)} \quad (2.43)$$

where G_i represents the gated attention score for the feature F_i . This step ensures that the features are weighted appropriately before fusion. Finally, the features from different branches are fused using the dynamically calculated attention weights:

$$F_{Fused} = \alpha_1 F_{MLP} + \alpha_2 F_{DenseNet} + \alpha_3 F_{ResNet} \quad (2.44)$$

This final fused feature vector F_{Fused} combines the most relevant aspects of the input features, weighted by their dynamically adjusted importance. By leveraging the strengths of MLP, DenseNet, and ResNet branches and dynamically fusing their features using multi-head attention, self-attention, and gating mechanisms, our MHHPN network effectively captures and combines diverse feature representations, improving performance on various tasks. The pseudocode of the fusion process is given in Algorithm 3.2.

Algorithm 2.2: Pseudocode for Feature Fusion Process by Dynamic Attention-Weighted Mechanism	
Input: Features from each branch: F_p (MLP Branch), F_R (ResNet Branch), F_D (DenseNet Branch) Output: Fused Feature Set	
Step 1: Adapted feature transformation from each branch	
	$F_{p_adapted} = \text{Adapt Features } (F_p, W_p)$ $F_{R_adapted} = \text{Adapt Features } (F_R, W_R)$ $F_{D_adapted} = \text{Adapt Features } (F_D, W_D)$
Step 2: Applying a multi-head attention mechanism for each set of adapted features from the branches using $\text{MultiHead}(F_i) = \text{Concat}(\text{head1}(F_i), \text{head2}(F_i), \dots, \text{headH}(F_i))$	
Step 3: Calculate self-attention scores for each branch's features to understand how much each feature relates to the others within the same branch using	
	$\text{SelfAttention}(F_i) = \text{softmax} \left(\frac{Q_i * K_i^T}{\sqrt{d_k}} \right)$
Step 4: Use a gating mechanism to determine how much importance should be given to each branch's features using	
	$\text{Gate Scores } (F_p) = \text{Gate Mechanism } (F_p)$ $\text{Gate Scores } (F_R) = \text{Gate Mechanism } (F_R)$ $\text{Gate Scores } (F_D) = \text{Gate Mechanism } (F_D)$

Step 5: Using a softmax function, calculate the importance weights for each branch's features using $\text{SoftMax}(\text{GateScores}(F_p), \text{GateScores}(F_R), \text{GateScores}(F_D))$

Step 6: Combine the features from the Perceptron, DenseNet, and ResNet branches by multiplying each by its respective importance weight.

Step 7: Use the fused features for classification by passing through a Softmax layer.

2.7 CYBERSECURITY ATTACKS: A COMPREHENSIVE ANALYSIS

Cyber-attacks are more prevalent and sophisticated, infiltrating information systems, critical infrastructures, and personal data at a gigantic scale. This section provides an overview of eminent types of cyber-attacks, modes, and quantitative models where necessary. With digitalization speeding up and connectivity expanding, attackers have increasingly more chances to take advantage of vulnerabilities in software, hardware, and even human behavior. The objectives of this chapter are to:

- i. Classify and define different types of cybersecurity attacks.
- ii. Talk about technical and mathematical modeling of attacks.
- iii. Assess the impact to organizations and individuals.
- iv. Sweep for cybersecurity future threats and trends.

Good understanding of such attack mechanisms and their mathematical descriptions helps to have proper defense mechanisms and makes critical infrastructures more robust.

2.7.1 Malware Attacks

Malware or malicious code defines computer programs that are developed to cause harm, violate, or gain unauthorized access on systems [63]. As shown in the figure 2.19, the malware attacks taxonomy. Subsequent sections present some prevalent categories of malware and offer mathematical models where needed.

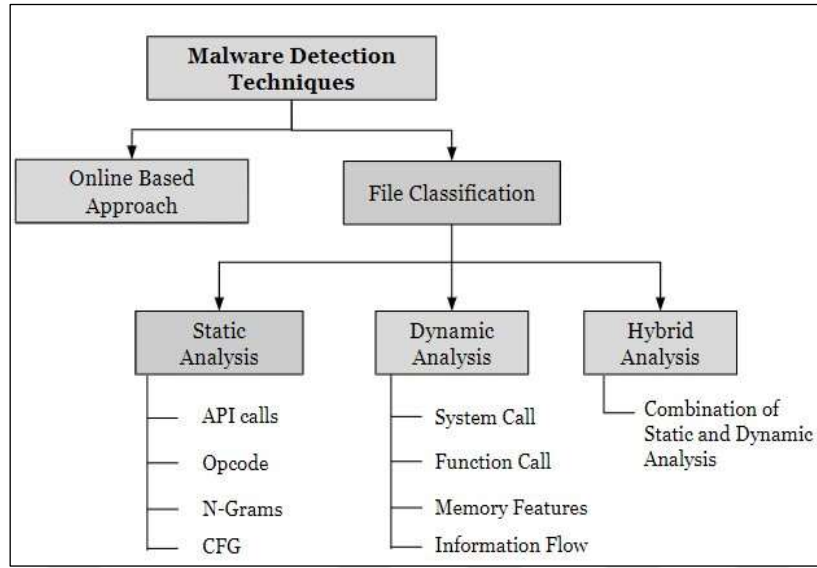


Figure 2.19: The Malware Attacks Taxonomy [63].

2.7.2 Denial-of-Service (DoS) and Distributed Denial-of-Service (DDoS) Attacks

A DoS attack saturates a victim's resources. A packet arrival rate λ of malicious packets can be modeled by a Poisson process [64]. Also, figure 2.20, shows the DDoS attacks workflow. Given the service rate μ of the server and buffer size B , the system overload probability (i.e., queue length Q greater than T) can be approximated as:

$$P(Q > T) = 1 - \sum_{k=0}^T \frac{(\lambda/\mu)^k}{k!} e^{-\lambda/\mu}. \quad (2.45)$$

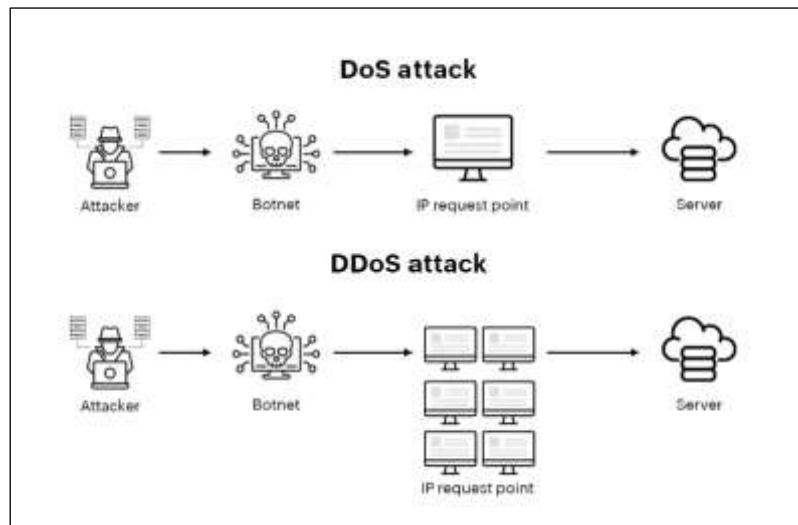


Figure 2.20: Workflow of DDoS Attacks [64].

When it comes to DDoS attacks, the total arrival rate is artificially increased, thus distorting the overload probability. Mathematically, it is complicated to differentiate between legitimate bursts and hostile bursts.

2.7.3 Man-in-the-Middle (MitM) Attacks

MitM attacks do not typically have neatly defined mathematical models, but detection software typically uses statistical models to flag data-flow anomalies. In Figure 2.21 the workflow of Man-in-the-Middle (MitM) Attacks.

For example, under normal operation with mean and variance represented by μ and σ of inter-packet delays, one might flag a delay d as anomalous if

$$\left| \frac{d-\mu}{\sigma} \right| > \tau, \quad (2.46)$$

with τ some threshold parameter. Statistical techniques form the foundation of most intrusion detection algorithms.

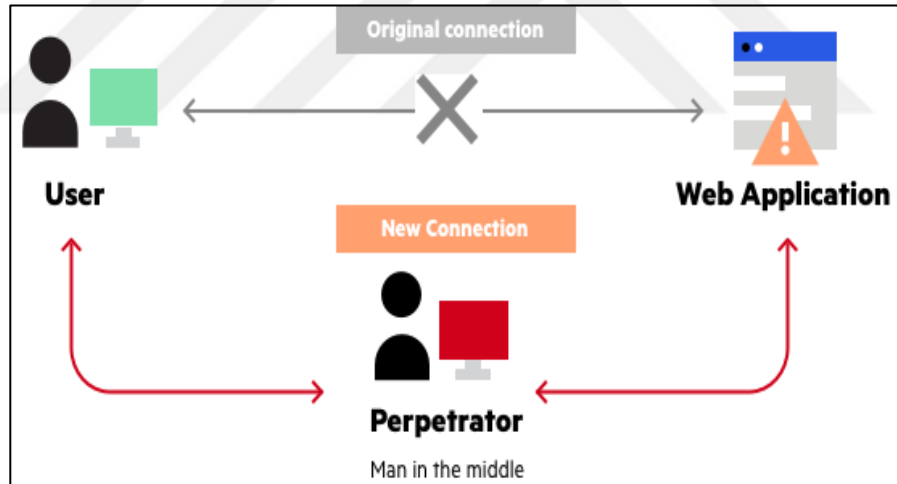


Figure 2.21: Workflow of Man-in-the-Middle (MitM) Attacks [65].

2.7.4 Social Engineering Attacks

Social engineering attacks bypass technical defences by targeting human weaknesses [66]. Although such attacks are not as amenable to direct mathematical modelling, probabilistic models can capture the overall threat. For instance, the success probability P_{SE} of a phishing attack can be modeled as

$$P_{SE} = p_e \times p_c, \quad (2.47)$$

where p_e is the probability that the email is opened, and p_c is the probability that the recipient clicks on a malicious link. This product model assists in understanding and addressing the risk through training and technical controls. Figure 2.22 illustrates a network social engineering attack taxonomy, separating them into two. Computer-based attacks employ automated software, deceptive web interfaces, and malware to present deceptive messages and plagiarized information. Computer-based attacks and human-based attacks employ psychological manipulation—phishing, pretexting, baiting, and tailgating—to trick individuals into divulging sensitive information. These attacks compounded highlight the need for user awareness and technical countermeasures to avert risks.

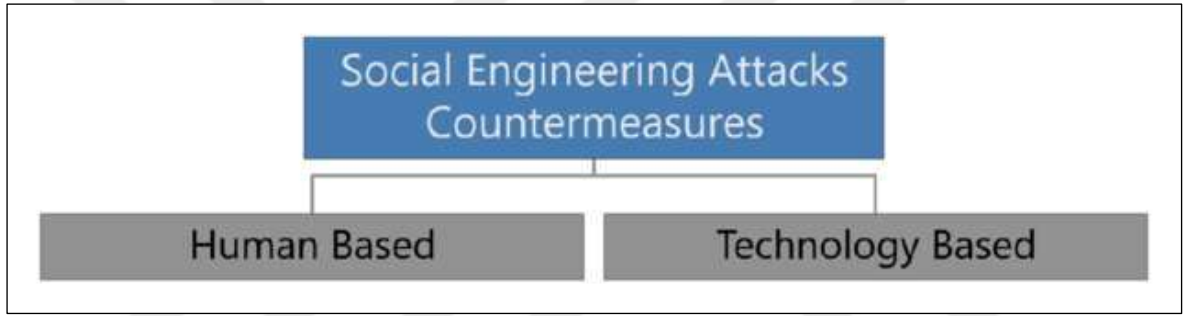


Figure 2.22: Illustrates A Network Social Engineering Attacks Taxonomy [66].

2.7.5 IoT-Based Attacks

As the quantity of IoT gadgets grows, the attack is mimicked while the system is portrayed as a diagram $G(V, E)$ where V symbolizes gadgets and E symbolizes correspondence links. The vulnerability of the system V_{IoT} may be quantified as

$$V_{\text{IoT}} = \frac{1}{|V|} \sum_{v \in V} \delta(v), \quad (2.48)$$

where $\delta(v)$ represents the vulnerability score of devices v . The measure above is convenient when prioritizing defense methods in diverse devices [67].

2.7.6 AI-Driven Attacks

AI-based attacks adapt in real-time against defenses. The rewards of these kinds of attacks may be modeled through a dynamic adversarial game in which defender strategy $S_D(t)$ and

attacker strategy $S_A(t)$ and both functions of time. Game-theoretic models utilize utility functions $U_A(S_A, S_D)$ and $U_D(S_A, S_D)$ for the modeling of attacker and defender payoff, respectively [68]. An equilibrium state occurs when

$$\frac{\partial U_A}{\partial S_A} = \frac{\partial U_D}{\partial S_D} = 0. \quad (2.49)$$

2.8 RELATED WORK

Intrusion Detection Systems (IDS) have evolved tremendously over the past decades. Early methods were centered around anomaly detection and statistical modeling, while current research includes utilizing machine learning and deep learning. Recent studies have suggested many alternative approaches, ranging from conventional methods based on clustering algorithms and neural networks to novel architectures based on convolutional neural networks (CNNs), recurrent neural networks (RNNs), long short-term memory (LSTM) networks, and autoencoders. Table 2.1 gives a comparative overview of fifteen early works. The abstract highlights major information such as the detection method, test sets utilized in evaluation, and major findings or performance improvements reported in the provided studies

Table 2.2: Comparison of IDS Detection Methods in Literature

No	Authors	Proposed Work	Dataset(s)	Key Findings
1	D. E. Denning [48]	Detection Anomaly Attacks	Traffic Dataset	Detection Framework
2	L. Liao, [49]	IDS based Auto coder	KDD Cup 99	Unsupervised Learning improved anomaly detection
3	Y. Kim, [50]	IDS using base CNN	Network IoT dataset	Improved feature extraction stage
4	H. Yin, [51]	RL with LSTM/RNN	CICIDS-2018	Enhanced detection accuracy
5	R. Vijayakumar, [52]	Novel hybrid Deep learning	NSL-KDD	High accuracy detection
6	J. Moon [53]	Novel hybrid ML	Collected/Simulated dataset	Integrated abuse and anomaly detection techniques
7	Z. Yang [54]	Temporal DNN and Hierarchical Spatial Model	CICIDS-2018	Analyzed and collected optimal features

Table 2.2: Comparison of IDS Detection Methods in Literature “Table Continued”

8	N. Chilamkurti [55]	Distribution of Deep Learning technique	Simulated IoT Data	IoT effective detection
9	Y. Meidan et al [56]	Deep Autoencoder	Botnet IoT dataset	Low alarm false positive rate achieved
10	Z. Zhang et al. [57]	IDS using LSTM	UNSW-NB15	Effective detection rate with high accuracy
11	X. Liang et al. [58]	Machine learning techniques and	NSL-KDD	Optimal feature selection and a high accuracy level
12	A. Gupta et al. [59]	Hybrid SVM and K-Means Clustering	KDD Cup 99	Effective in clustering and classification
13	E. Lee, S. J. Stolfo [60]	Data Mining Approaches	DARPA Dataset	Early demonstration of mining techniques in IDS
14	P. Mukkamala et al. [61]	Neural Networks and ML	KDD Cup 99	Showed promising performance with neural network models
15	B. Sommer, V. Paxson [62]	ML Ensemble Methods	Various IDS Datasets	Highlighted limitations of closed-world assumptions

2.9 RESEARCH GAPS IN CYBERSECURITY: AN INTEGRATED PERSPECTIVE

The rapid growth of cyberattacks and their corresponding defense methods has led to unprecedented growth in intrusion detection systems (IDS) and overall cybersecurity practices. All progress notwithstanding, there are certain research gaps that still impede the development of completely resilient systems. It is essential to thoroughly understand these gaps so that future research can be guided and next-generation solutions can be developed that are resilient enough to combat continuously evolving threats. Among the shortcomings of cybersecurity research today, an overdependency on simulated or vintage datasets represents a significant error. The overwhelming majority of the research on IDS hinges on reference datasets such as KDD Cup 99, NSL-KDD, UNSW-NB15, or

CICIDS-2018. Although such datasets offer an orderly test bench, they miss depicting the wealth and variety found within modern-day network traffic. This discrepancy is seen when high-performing systems in well-controlled, static conditions are placed in real, dynamic, and heterogeneous network environments. One such example is the spread of malware in a live network that can be modeled mathematically with logistic growth:

$$\frac{dI(t)}{dt} = rI(t) \left(1 - \frac{I(t)}{N}\right), \quad (2.50)$$

where $I(t)$ is the infected host population, r is infection rate, and N is the susceptible host population. These models may not necessarily, though, consider heterogeneity in real-world deployments. The need for massive real-world dataset creation and complex simulation infrastructures is an open problem that would reduce the laboratory experiment versus operational network gap [69], [70].

Another significant research deficit is the inability of current IDS to accommodate evolving and zero-day attacks. Traditional systems operate based on fixed feature sets or pre-specified attack signatures and hence are not as responsive as polymorphic and rapidly evolving attacks. Even though machine learning and deep learning frameworks offer adaptive nature, such frameworks usually require mass retraining with every newly found attack pattern. For example, suppose an adaptive system in the form of transfer learning in which a pre-trained model M is adapted by fine-tuning on a few new attack examples X_{new} . The adaptation is done using a loss function L :

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^n \ell \left(M \left(X_{new}^{(i)} \right), y^{(i)} \right). \quad (2.51)$$

While promising, such approaches take additional effort to make the models robust and not prone to catastrophic forgetting when they encounter new attack classes [71], [72]. This is particularly important in applications where rapid response to newly rising threats is the need of the hour.

The transparency and explainability issue of detection models is another research issue. Modern IDS increasingly employs advanced machine learning architectures whose decision processes are all too often opaque. The "black-box" character of these systems erodes their credibility and hinders their deployment in real-world environments of critical infrastructure. Recent work on explainable artificial intelligence (XAI) tries to do better by making the

internal workings of these models transparent. Yet, the integration of interpretability in high-performing IDS is still in its early stages. Methods with high detection accuracy but also interpretable, explainable explanations are needed to allow cybersecurity analysts to understand why specific events are detected as malicious and thus enhance the effectiveness of incident response [73], [74].

Scalability and real-time responsiveness are also other major deficiencies. Most IDS prototypes are designed and tested in controlled, offline environments that do not match the requirements of deployed networks. For high-speed networks, where network traffic is monstrous and ever-changing, IDS must respond in real time without compromising detection quality. For example, a denial-of-service (DoS) attack can be modeled as a queuing system with the arrival of malicious packets as a Poisson process with rate λ and the server transmitting packets at rate μ . The likelihood of the queue overflowing above level T can be approximated by

$$P(Q > T) = 1 - \sum_{k=0}^T \frac{(\lambda/\mu)^k}{k!} e^{-\lambda/\mu}. \quad (2.52)$$

These mathematical models promote the need for scalability and low-overhead IDS efficient operation. Distributed processing architectures, low-complexity deep learning algorithms, and low-complexity algorithms are research areas to be followed to achieve these operational needs [75].

Deducting technical challenges, correlation of multidomain security data is also a virgin territory. Current cybersecurity increasingly entails correlation of data across multiple domains, including network traffic, endpoint security, user behavior analytics, and threat intelligence. But current IDS are prone to operate in siloes, processing a single stream of data rather than correlating data from multiple streams. Advanced data fusion processes and systems able to reconcile varied data and time variations are highly needed to provide an integrated overview of the security situation. Those systems that fuse disparate data sources can utilize probabilistic models or Bayesian networks in correlating varying data, therefore improving the overall precision of threat detection rate [76].

New threats require new technologies that more traditional IDS are not well-designed to face. For instance, the recent explosion of Internet-of-Things (IoT) devices, the inherent

processing limitations of such devices, and the security choices they have limit certain capabilities. An IoT device network can be modeled as a graph $G(V, E)$, where each device $v \in V$ is assigned an associated vulnerability score $\delta(v)$. The overall network vulnerability could be quantified as:

$$V_{\text{IoT}} = \frac{1}{|V|} \sum_{v \in V} \delta(v). \quad (2.53)$$

Similarly, cloud computing and software-defined networks (SDN) need new approaches to securing dynamic and virtualized environments. Threats of these technologies are made complex by challenges such as API security and the ephemeral nature of virtual machines, and thus require special research to generate specialized IDS to effectively address these special challenges [77].

Finally, human factors and social engineering remain a core research gap. Technical defenses have become more effective, yet attackers have persisted in exploiting human vulnerabilities by using techniques like phishing, pretexting, and baiting. The probability of a successful social engineering attack, P_{SE} , can be expressed as the product of the probability that a message is received and acted upon:

$$P_{\text{SE}} = p_e \times p_c, \quad (2.54)$$

where p_e is the exposure probability of the message and p_c is the probability that the receiver responds to it. The naive model emphasizes the need for user education and behavioral profiling. Inter-disciplinary research aggregating evidence from cybersecurity, psychology, and human-computer interaction is needed urgently to counter the social engineering threat [78].

Despite the unprecedented advances in intrusion detection systems (IDS) and cybersecurity, there exist numerous critical gaps that continue to hinder the development of effective and resilient defense measures. These consist of gaps more profound than the presence of data, model interpretability, algorithm scalability, and human and domain-level knowledge integration. A solution that will address these gaps will have a fundamental contribution to enhancing the reliability, flexibility, and resilience of current IDS solutions. One of the enduring issues of IDS research is that there are no realistic datasets available up to date. The majority of publicly available datasets, such as KDD99 or NSL-KDD, are old and do

not reflect the complex and dynamic nature of cyberattacks in today's era. Even current datasets like CICIDS-2017 and UNSW-NB15, as good as they are compared to the old ones, still risk being generated in clean environments that do not reflect real-world conditions, e.g., network noise, encrypted traffic, and variability of user behavior. Lack of real-time, heterogeneous, and labeled data restricts model training and validation on real-world environments. Another region of tremendous missingness is a shortage of interpretable and adaptive detection models. While it has been observed that deep learning models are more accurate, the models will, in all probability, be "black boxes," and hence, security professionals find it difficult to interpret predictions or believe conclusions made by such models. Explain that ability and accountability are vital in high-risk scenarios. Aside from that, existing models are static and less aligned with the newly emerging attack patterns or the time drift concept, which suggests a need for ongoing learning and adaptive models. The scalability and efficiency problem is also not feasible. The majority of the cutting-edge models entail substantial computational complexity and are not scalable for real-world deployment on low-resource environments or in real-time systems. Real-world IDS deployments need to have efficient algorithms with performance vs. accuracy trade-offs, primarily in edge or distributed settings.

3. PROPOSED METHODOLOGY

3.1 INTRODUCTION

This chapter contains four subsections: data collection, model building, and model evaluation. It begins from the data collection through network logs, intrusion detection sensors, and publicly available data sets for normal traffic and attack traffic. One of the critical objectives of data collection is to ensure that the sample represents the diversity and complexity of the network activities, particularly DDOS attacks. Once the data are gathered, pre-processing is carried out to improve the quality and reduce computational complexity. Pre-processing deals with cleaning the data through the removal of duplications and inconsistencies, and scaling numeric data for normalization. Data imputation and noise filtering are some of the methods used in the process to correct missing or erroneous values, thereby making the output of the subsequent analysis more reliable.

Following the pre-processed data, feature extraction is carried out for the selection of the most informative features being used for DDOS attack detection. It is carried out to cover the main features such as packet rate, source and destination IPs change, protocol types, and time features. Extracted features are based on the static and dynamic nature of the network traffic. Aside from extraction, feature optimization is carried out in association. Optimization methods are applied here for removing redundancy and irrelevant features via dimension reduction. The optimization not only improves the model performance by removing the noise but also the execution time of the classifier. Model-building phase integrates the optimized features in a classification model that is designed specifically for the detection and classification of DDOS attacks.

The classifier, which is developed through a combination of different machine learning algorithms, is trained to differentiate between normal and different kinds of DDOS attacks based on the optimized feature set. The model, during training, will optimize the parameters based on iterative optimization and reduce the gap between actual and predicted outcomes. The proposed system categorizes the attacks efficiently in close real time, with detection accuracy and response time. Finally, the performance of the proposed methodology is exhaustively compared with benchmark test parameters as shown in Figure 3.1

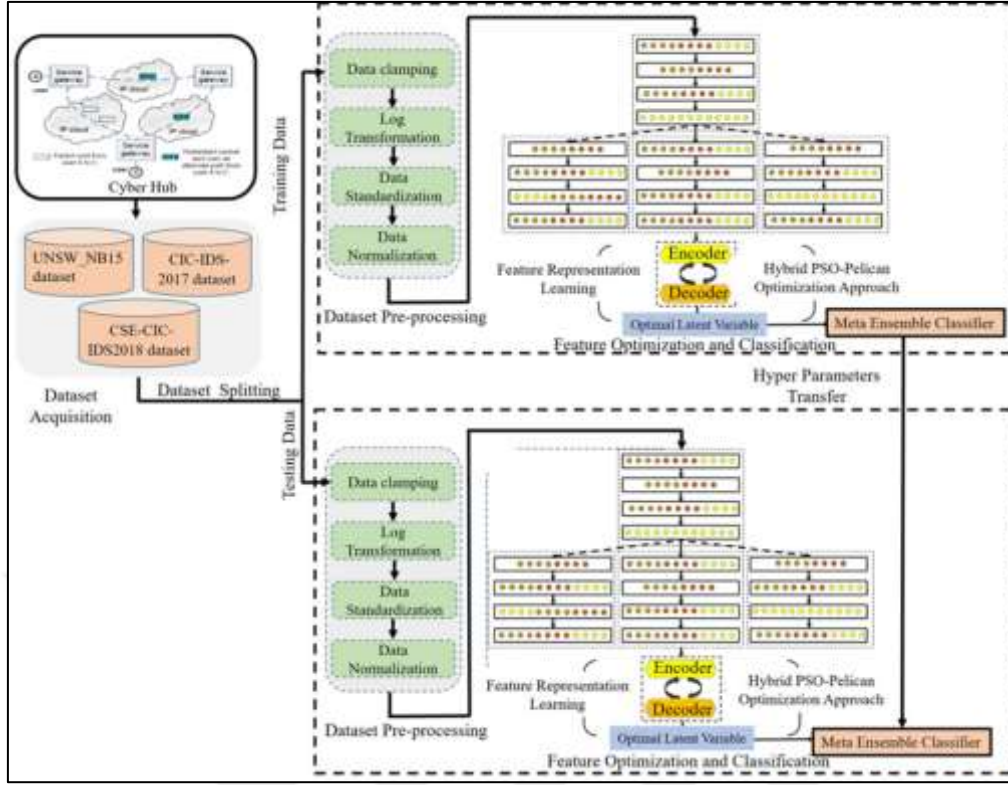


Figure 3.1: Workflow of the Proposed Methodology (By Author).

3.2 DATA COLLECTION

The proposed research employs three benchmarking datasets—CIC-IDS2018, UNSW-NB15, and CIC-IDS-2017—to validate the detection of DDoS attacks. Using the joint exploitation of these datasets, the research offers extensive testing on a large range of network traffic patterns and attack patterns. A complex hierarchical neural network architecture is employed that is augmented by other approaches to support feature choice. This collaborative work allows the most essential traits to be realized and thereby increases the accuracy and validity in determining DDoS.

3.2.1 CIC-IDS2018 Dataset

CIC-IDS2018 is a newly developed, large-scale intrusion detection dataset built by the Canadian Institute for Cybersecurity. It surpasses the limitations of current datasets in realistic network traffic and a vast number of types of attacks, from DDoS to DoS, brute force, and intrusion attacks. The dataset provides a high-quality set of network flow features—packet inter-arrival times, flow durations, and protocol-specific information—

derived from raw packet captures. This collection of features in depth allows one to analyse normal and intrusive traffic, and this makes CIC-IDS2018 a great test bed for evaluating machine learning IDS models. Its closeness to reality fills the gap between experimental setting and real network setting and allows one to have more robust and flexible intrusion detection systems [79]. In figure 3.2 visualize the CIC-IDS2018 dataset.

	Protocol	Flow Duration	Total Fwd Packets	Total Backward Packets	Fwd Packets Length Total	Back Packets Length Total	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean	Fwd Packet Length Std	Fwd Seq Size Min	Active Mean
0	0	112640768	3	0	0	0.0	0	0	0.000000	0.000000	...	0.0
1	0	112641773	3	0	0	0.0	0	0	0.000000	0.000000	...	0.0
2	6	20784143	23	44	2416	1344.0	240	64	105.043480	54.542294	...	26247
3	0	112640836	3	0	0	0.0	0	0	0.000000	0.000000	...	0.0
4	6	20	1	1	0	0.0	0	0	0.000000	0.000000	...	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
6659527	6	10056041	5	5	1136	2688.0	578	0	227.199997	311.186768	...	0.0
6659528	6	10156986	5	5	1089	1923.0	587	0	217.800003	299.745239	...	0.0
6659529	6	5095331	3	1	0	0.0	0	0	0.000000	0.000000	...	0.0
6659530	6	5235511	3	1	0	0.0	0	0	0.000000	0.000000	...	0.0
6659531	6	5807256	6	4	327	145.0	245	0	54.500000	95.671837	...	29156

Figure 3.2: Visualize of CIC-IDS2018 dataset (By Author).

3.2.2 UNSW-NB15 Dataset

UNSW-NB15 dataset is a comprehensive dataset developed by the University of New South Wales (UNSW) for research in network intrusion detection. It contains a combination of different kinds of attacks—such as exploits, DoS, and reconnaissance—and normal network traffic. It consists of a diverse set of features from actual network traffic based on flow duration, packet statistics, and protocol-based features. This flexibility enables researchers to model and experiment with IDS models using real-world conditions, bridging the gap between simulation-based experiments and real-world networks [80]. Figure 3.3 visualize the UNSW-NB15 dataset.

	dur	proto	service	state	spkts	dpkts	sbytes	dbytes	rate	sttl	...	ct_dst_sport_ltm	ct_dst_src_ltm	is_fip_l
0	0.000011	udp	-	INT	2	0	496	0	90909.0902	254	...	1	2	0
1	0.000008	udp	-	INT	2	0	1762	0	125000.0003	254	...	1	2	0
2	0.000005	udp	-	INT	2	0	1068	0	200000.0051	254	...	1	3	0
3	0.000006	udp	-	INT	2	0	900	0	166666.6608	254	...	1	3	0
4	0.000010	udp	-	INT	2	0	2126	0	100000.0025	254	...	1	3	0

Figure 3.3: Visualize the UNSW-NB15 Dataset (By Author).

3.2.3 CIC-IDS-2017 Dataset

CIC-IDS-2017 dataset is designed by Canadian Institute for Cybersecurity to provide an accurate and complete representation of the traffic in today's networks with malicious and normal activity. The dataset has been designed to provide an insight into the activities of today's networks and contains different types of attacks such as DDoS, brute force, intrusion attacks, etc. It delivers a complete feature set, combining low-level network flow statistics such as packet inter-arrival times, flow durations, and protocol-specific features that are critical to train and test intrusion detection systems. The traffic patterns in CIC-IDS-2017 being realistic make the dataset suitable for researchers interested in developing models generalizable in real-world settings [81]. Figure 3.4 visualize the CIC-IDS-2017 dataset.

	Protocol	Flow Duration	Total Fwd Packets	Total Backward Packets	Fwd Packets Length Total	Bwd Packets Length Total	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean	Fwd Packet Length Std	...	Fwd Seg Size Min	Active Mean
0	0	112640768	3	0	0	0.0	0	0	0.000000	0.000000	...	0	0.0
1	0	112641773	3	0	0	0.0	0	0	0.000000	0.000000	...	0	0.0
2	6	20784143	23	44	2416	1344.0	240	64	105.043480	54.542294	...	20	26247
3	0	112640836	3	0	0	0.0	0	0	0.000000	0.000000	...	0	0.0
4	6	20	1	1	0	0.0	0	0	0.000000	0.000000	...	20	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6659527	6	10056041	5	5	1136	2668.0	578	0	227.199997	311.186768	...	20	0.0
6659528	6	10156986	5	5	1089	1923.0	587	0	217.800003	299.745239	...	20	0.0
6659529	6	5095331	3	1	0	0.0	0	0	0.000000	0.000000	...	20	0.0
6659530	6	5235511	3	1	0	0.0	0	0	0.000000	0.000000	...	20	0.0
6659531	6	5807256	6	4	327	145.0	245	0	54.500000	95.671837	...	20	29156

Figure 3.4: Visualize The CIC-IDS-2017 Dataset (By Author).

3.3 PREPRESSING DATASET

To implement an effective intrusion detection system, proper data pre-processing of excellent quality is needed. Utilized datasets—UNSW_NB15, CIC-IDS-2017, and CSE-CIC-IDS2018—are heterogeneously distributed with too much noise. Therefore, we divide the pre-processing pipeline into some critical steps: data cleaning, log transformation, standardization, and normalization. All processes are mathematically defined to improve the quality of data and training models smoothly.

3.3.1 Log Transformation

Most network parameters, such as packet lengths and flow durations, are long-tailed. Log transformation reduces skewness and compresses the range of values [82]. Figure 3.5 displays the histogram of the positively skewed dataset along with its logarithmic transformation. The skewness of the non-transformed data (left) is 0.9, whereas that of the log-transformed data (right) is 0.2. For a feature x , log-transformed feature x' can be found as:

$$x' = \log(x + c), \quad (3.1)$$

where c is a constant (typically $c = 1$) ensuring that $x + c > 0$.

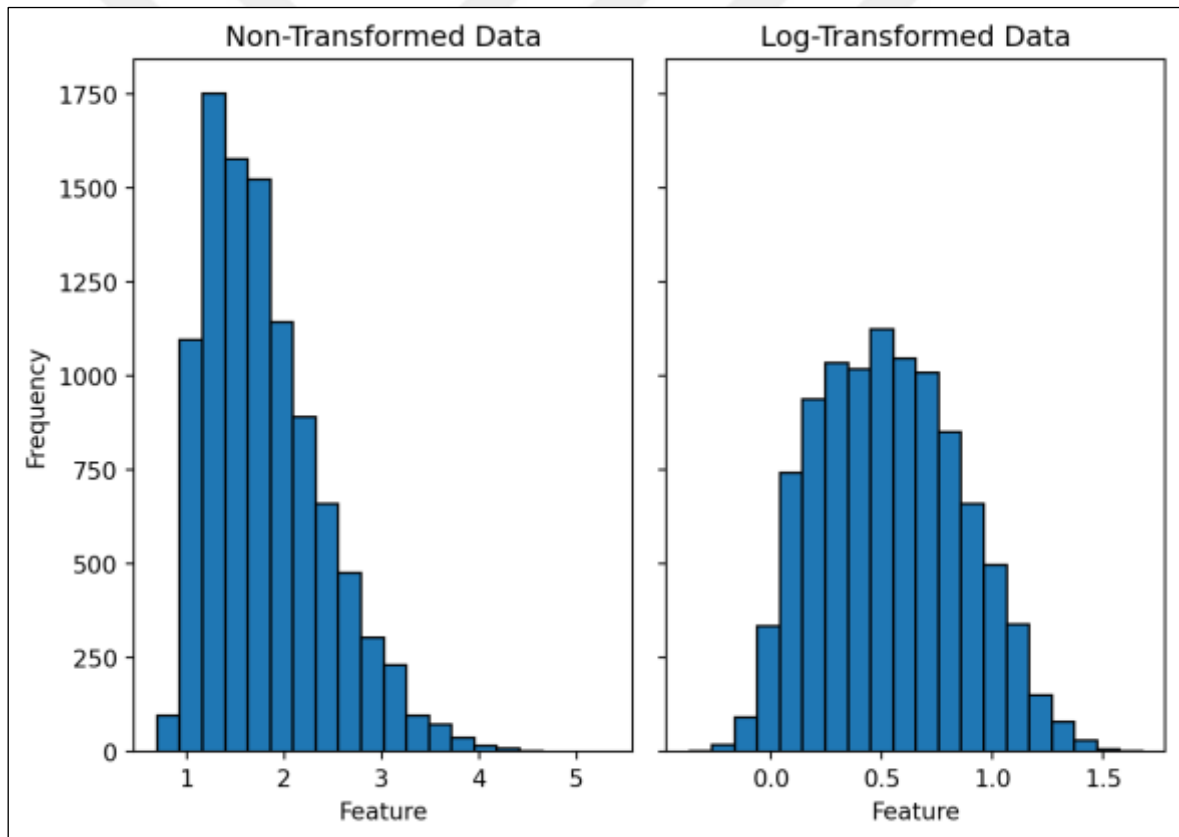


Figure 3.5: The Figure Displays the Histogram of the Positively Skewed Dataset Along with its Logarithmic Transformation. The Skewness of the Non-transformed Data (left) is 0.9, Whereas that of the Log-transformed Data (right) is 0.2 (By Author).

3.3.2 Data Cleaning

Data cleaning is the process of cleansing raw network data of errors, inconsistencies, and redundancies [82]. The primary tasks are removing duplicate records, missing values, and resolving inconsistencies.

For removing duplicates of the dataset. Let the raw data set be represented as:

$$D = \{x_i \in \mathbb{R}^d \mid i = 1, 2, \dots, N\}, \quad (3.2)$$

where for every $x_i = [x_{i1}, x_{i2}, \dots, x_{id}]$ is a d-feature vector. Duplicates can be removed by making a set:

$$D_{\text{clean}} = \{x \in D \mid x \text{ is unique}\}. \quad (3.3)$$

In practice, when $x_i = x_j$ for $i \neq j$, keep one instance in D_{clean} .

Missing values can be imputed. One simple technique uses the feature mean over all non-missing records. For feature j , let the mean be:

$$\mu_j = \frac{1}{N_j} \sum_{i \in I_j} x_{ij}, \quad (3.4)$$

where I_j is the set of indices for which the j th feature appears and $N_j = |I_j|$. The imputed value for any missing x_{ij} is then:

$$\tilde{x}_{ij} = \begin{cases} \mu_j, & \text{if } x_{ij} \text{ is missing,} \\ x_{ij}, & \text{otherwise.} \end{cases} \quad (3.5)$$

3.3.3 Data Standardization

Standardization scales the features so that all of them will have a mean of 0 and a standard deviation of 1. Standardization is necessary if features are in different units or scales because it can hamper the speed of convergence of optimization algorithms in machine learning models [83]. For some specific feature j , The normalized value z_{ij} of example i is:

$$z_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j}, \quad (3.6)$$

where:

$$\mu_j = \frac{1}{N} \sum_{i=1}^N x_{ij} \text{ is the mean of feature } j,$$

$$\sigma_j = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_{ij} - \mu_j)^2} \text{ is the standard deviation.}$$

3.3.4 Data Normalization

Normalization scales the data into a bounded range, typically [0,1], appropriate for highly sensitive input feature magnitude algorithms [84].

For each feature j , let:

$$x_{\min,j} = \min_{1 \leq i \leq N} x_{ij}, \quad x_{\max,j} = \max_{1 \leq i \leq N} x_{ij}. \quad (3.7)$$

Then the normalized value x_{ij}^{norm} is:

$$x_{ij}^{\text{norm}} = \frac{x_{ij} - x_{\min,j}}{x_{\max,j} - x_{\min,j}}. \quad (3.8)$$

This changeable guarantees that:

$$\min_{1 \leq i \leq N} x_{ij}^{\text{norm}} = 0, \quad \max_{1 \leq i \leq N} x_{ij}^{\text{norm}} = 1. \quad (3.9)$$

3.4 ARCHITECTURE OF THE PROPOSED TREE-BASED CNN (T-CNN)

The proposed Tree-based CNN (T-CNN) is used as a modular hierarchical architecture which handles both the task of classification as well as the task of learning features outstandingly. Data is fed to the network here as one-dimensional input of size ("None",100,1). This is then passed to the first convolutional layer (conv1d_0) comprising 32 filters of 3 to obtain the output tensor of size ("None",100,32). Another max-pooling layer, max_pooling1d_0, reduces the temporal dimension by half to create a tensor with shape. There are four primary blocks in the network:

- i. Block 0 has two parallel convolutional layers (conv1d_1 and conv1d_2) of filter size 32 with size 3 each. Parallel paths pick up different features of the input and outputs are concatenated to yield a combined feature map of shape ("None",50,64). Another max-pooling layer (max_pooling1d_1) reduces the size again to ("None",25,64).

- ii. Block 1 builds on this with both dense and residual connections. Here, the input is once more passed through two identical convolutional layers (conv1d_3 and conv1d_4) once more with 32 filters, and their outputs—which are once more of similar shape as Block 0—are concatenated. Residual connection is then enforced by adding the input of the block to its output so that important information is preserved and that gradients propagate more smoothly during backpropagation. This block is succeeded by a max-pooling layer that down samples the feature map to ("None",12,64). Block 2 is employed for feature extraction at different scales. It comprises four parallel convolutional layers: the first two (conv1d_5 and conv1d_6) both having 64 filters, and the last two (conv1d_7 and conv1d_8) having 32 filters. Their outputs are concatenated together to form a dense feature map of shape ("None",12,128). Residual connection is added for strengthening information flow and the size is cut down by performing a max-pooling to ("None",6,64).
- iii. Block 3 converts the spatial features learned into a global compact representation through a global average pooling layer with output vector of shape ("None",64). The vector is then passed through a series of dense layers: one fully connected of 128 neurons (dense_0), another of 64 neurons (dense_1), and finally an output layer (dense_2) of 10 neurons with SoftMax activation to provide class probabilities.

The concatenation of the parallel branches enables the network to merge different features from different scales. Residual connections between Block 1 and Block 2 enable direct information pass and prevent the vanishing gradient problem, stabilizing training and enhancing convergence. Overall, the architecture consists of 28 convolutional and six pooling layers heavily coupled with dense concatenation and residual operations. The architecture is specifically optimal for high-dimensional network data classification problems that are challenging since it sacrifices computational expense for high-quality feature extraction.

Follow along with the following eight-step derivation of the gradient flow to see how the model is learning.

Initial Convolution Operation:

Given an input $X \in \mathbb{R}^{100 \times 1}$ and a kernel $W_0 \in \mathbb{R}^{3 \times 1 \times 32}$, the convolution operation is defined as:

$$Y^{(0)}[i, k] = \sum_{j=0}^2 X[i + j] \cdot W_0[j, k] + b_0[k]. \quad (3.10)$$

The gradient with respect to W_0 is computed by:

$$\frac{\partial \mathcal{L}}{\partial W_0[j, k]} = \sum_i \frac{\partial \mathcal{L}}{\partial Y^{(0)}[i, k]} \cdot X[i + j]. \quad (3.11)$$

- a. Max-Pooling Operation: For a max-pooling layer which down samples the spatial dimension by two, the output for each pooled index is:

$$P^{(0)}[i, k] = \max_{j \in \text{pool window}} Y^{(0)}[2i + j, k]. \quad (3.12)$$

- b. Backpropagation distributes the gradient to only the max element within every window.

- c. Parallel Convolution Branches (Block 0):

In Block 0, two parallel convolutions generate outputs $Y_1^{(1)}$ and $Y_2^{(1)}$. They are computed as:

$$Y_1^{(1)}[i, k] = \sum_{j=0}^2 P^{(0)}[i + j, :] \cdot W_1[j, :, k] + b_1[k], \quad (3.13)$$

$$Y_2^{(1)}[i, k] = \sum_{j=0}^2 P^{(0)}[i + j, :] \cdot W_2[j, :, k] + b_2[k]. \quad (3.14)$$

The output is given as follows:

$$Y^{(1)} = \text{Concat}(Y_1^{(1)}, Y_2^{(1)}), \quad (3.15)$$

Given as follows:

$$\frac{\partial \mathcal{L}}{\partial Y_1^{(1)}} \quad \text{and} \quad \frac{\partial \mathcal{L}}{\partial Y_2^{(1)}} \quad (3.16)$$

It is computed by slicing the gradient of the concatenated output.

- d. Residual Connection in Block 1:

Let $X^{(1)}$ be Block 1 input and its transformation be $F^{(1)}(X^{(1)})$. The block output is:

$$Y^{(1)} = F^{(1)}(X^{(1)}) + X^{(1)}. \quad (3.17)$$

The $X^{(1)}$ gradient is as follows:

$$\frac{\partial \mathcal{L}}{\partial X^{(1)}} = \frac{\partial \mathcal{L}}{\partial Y^{(1)}} \cdot \left(\frac{\partial F^{(1)}(X^{(1)})}{\partial X^{(1)}} + I \right). \quad (3.18)$$

e. Concatenation in Block 2:

The convolution outputs in block 2 is give four parallel $Y_1^{(2)}, Y_2^{(2)}, Y_3^{(2)}, Y_4^{(2)}$ are simultaneously:

$$Y^{(2)} = \text{Concat}(Y_1^{(2)}, Y_2^{(2)}, Y_3^{(2)}, Y_4^{(2)}). \quad (3.19)$$

The derivative is divided between the branches by slicing the gradient accordingly.

f. Second Residual Connection (Block 2):

Similar to Block 1, assume the block's input is $X^{(2)}$ and its output after convolution is $F^{(2)}(X^{(2)})$. The output is:

$$Y^{(2)} = F^{(2)}(X^{(2)}) + X^{(2)}. \quad (3.20)$$

Thus, the gradient with respect to $X^{(2)}$ is:

$$\frac{\partial \mathcal{L}}{\partial X^{(2)}} = \frac{\partial \mathcal{L}}{\partial Y^{(2)}} \cdot \left(\frac{\partial F^{(2)}(X^{(2)})}{\partial X^{(2)}} + I \right). \quad (3.21)$$

g. Global Average Pooling (Block 3):

Let the feature map from Block 2 be $Y^{(3)} \in \mathbb{R}^{6 \times 64}$. Global average pooling computes:

$$v[k] = \frac{1}{6} \sum_{i=1}^6 Y^{(3)}[i, k]. \quad (3.22)$$

The derivative with respect to each element is:

$$\frac{\partial \mathcal{L}}{\partial Y^{(3)}[i, k]} = \frac{1}{6} \frac{\partial \mathcal{L}}{\partial v[k]}. \quad (3.23)$$

h. Dense Layers and SoftMax Output:

Finally, the merged vector v is fed through dense layers. For the final dense layer, suppose the pre-activation output is z . The SoftMax is:

$$\hat{y}_k = \frac{\exp(z_k)}{\sum_{l=1}^{10} \exp(z_l)}, \quad (3.24)$$

and the cross-entropy loss yields:

$$\frac{\partial \mathcal{L}}{\partial z_k} = \hat{y}_k - y_k. \quad (3.25)$$

The gradients are backpropagated through the dense layers using standard chain rule operations:

$$\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}}{\partial z} \cdot \phi'(z) \cdot x^\top. \quad (3.26)$$

Figure 3.6, illustrates the proposed Tree-Convolutional Neural Network model architecture, designed with a hierarchical, modular architecture. The figure shows how the input data are processed first in a convolutional layer and then gradually enhanced in a number of blocks with parallel convolutions, dense concatenation, and residual connections. The modules are constructed to work together to generate multi-scale features and to have good gradient flow. Finally, global average pooling down samples the spatial features to a compact representation and feeds it to dense layers for final classification.

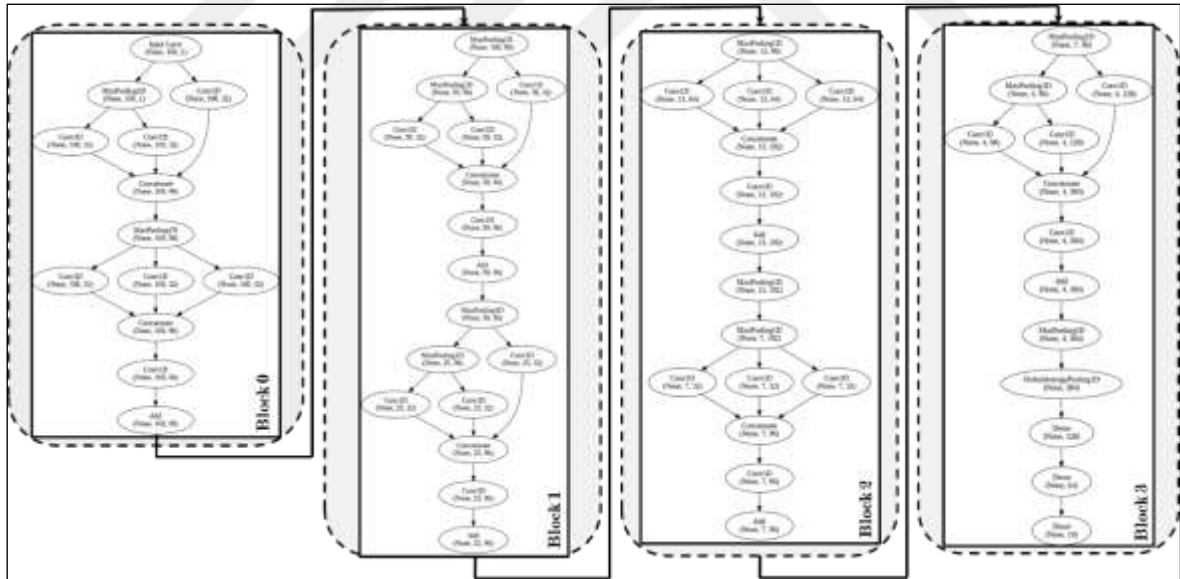


Figure 3.6: Figure Shows How the Input Data Are Processed First in A Convolutional Layer and Then Gradually Enhanced in A Number of Blocks with Parallel Convolutions, Dense Concatenation, And Residual Connections (By Author).

3.5 VARIATIONAL WAVELET DENOISING AUTOENCODER (VWDA)

The VWDA is a two-stage denoising and compression architecture for high-dimensional feature vectors [85]. It integrates discrete wavelet decomposition with a variational autoencoder architecture, followed by a decoder that reconstructs a clean, compact representation. For an input feature vector $X \in \mathbb{R}^{128}$, the VWDA first applies a discrete wavelet transform (DWT) that decomposes the signal into low-frequency (approximation) and high-frequency (detail) components. These are then sent through convolutional layers, inception modules, and a gate mechanism prior to compression in a bottleneck layer. The decoder performs the opposite operation using transposed convolutions and an inverse wavelet transform, spitting out the final denoised feature vector.

As follows the steps VWDA of Extended Mathematical Derivation: A 10-Step Chain

Step 1: Wavelet Decomposition

The input X is divided into approximation coefficients A_1 and detail coefficients D_1 to D_{WT} :

$$A_1(k) = \sum_n X[n] \cdot \varphi(2k - n), \quad D_1(k) = \sum_n X[n] \cdot \psi(2k - n), \quad (3.27)$$

where φ and ψ are the scaling (low-pass) and wavelet (high-pass) functions, respectively. The process yields 48 low-frequency and 14 high-frequency features.

Step 2: Coefficient Concatenation

The coefficients are concatenated to form the composite signal:

$$Z_0 = \text{Concat}(A_1, D_1) \in \mathbb{R}^m, \quad (3.28)$$

where $m = 48 + 14 = 62$ (or another suitable dimension depending on the implementation).

Step 3: Convolutional Feature Extraction

A 3×3 kernel convolutional layer is applied over Z_0 . Let the kernel be W_1 and the bias be b_1 . The feature map H is computed as:

$$H(i, j) = \text{ReLU}\left(\sum_{u,v} W_1(u, v) \cdot Z_0(i + u, j + v) + b_1\right). \quad (3.29)$$

Feature map H is computed as:

$$\frac{d}{dx} ReLU(x) = \begin{cases} 1, & x > 0, \\ 0, & x \leq 0. \end{cases} \quad (3.30)$$

Step 4: Inception Module for Multi-Scale Features

Multiple convolution operations with different kernel sizes are performed in parallel:

$$Z_{1 \times 1} = Conv_{1 \times 1}(H), \quad Z_{3 \times 3} = Conv_{3 \times 3}(H), \quad Z_{5 \times 5} = Conv_{5 \times 5}(H), \quad (3.31)$$

and pooling operation $Z_{\text{"pool"}}$ is also computed. Their result is concatenated:

$$Z_{\text{inception}} = Concat(Z_{1 \times 1}, Z_{3 \times 3}, Z_{5 \times 5}, Z_{\text{pool}}). \quad (3.32)$$

Step 5: Noise Gating Mechanism

A gating vector $G \in [0,1]^d$ is learned to kill noise. The gated feature map is given by:

$$Z_{\text{gated}} = Z_{\text{inception}} \odot G, \quad (3.33)$$

where $\odot$ denotes element-wise multiplication. The gradient with respect to G is given by

$$\frac{\partial \mathcal{L}}{\partial G} = \frac{\partial \mathcal{L}}{\partial Z_{\text{gated}}} \odot Z_{\text{inception}}. \quad (3.34)$$

Step 6: Bottleneck Compression

Gated features are processed by a bottleneck layer that squeezes them to a latent space

$Z_{\text{"Bottleneck"}} \in R^{16}$:

$$Z_{\text{Bottleneck}} = W_{\text{Bottleneck}} \cdot Z_{\text{Encoder}} + b_{\text{bottleneck}}, \quad (3.35)$$

where $W_{\text{Bottleneck}}$ is the projection matrix and $b_{\text{bottleneck}}$ is the bias vector. This step enforces compact feature embedding while preserving discriminative information.

Step 7: Transposed Convolution for Up sampling

In the decoder phase, the latent vector is up sampled using transposed convolution:

$$Z_{\text{up}} = ReLU(W_T * Z_{\text{Bottleneck}} + b), \quad (3.36)$$

with W_T representing the transposed convolution kernel. The derivative here follows similarly to standard convolutions, adjusted for the transposed operation.

Step 8: Inverse Wavelet Transform

To reconstruct the original signal, the inverse wavelet transform is applied:

$$\hat{X}[n] = \sum_k (A_1(k) \cdot \varphi(2k - n) + D_1(k) \cdot \psi(2k - n)), \quad (3.37)$$

which reconstructs $\hat{X}$ using the detail and approximation coefficients. Backpropagation of gradients is carried out through wavelet reconstruction by the chain rule.

Step 9: Loss Function and Backpropagation

The reconstruction error is measured using a loss function, such as mean squared error (MSE):

$$\mathcal{L} = \frac{1}{N} \sum_{n=1}^N (X[n] - \hat{X}[n])^2. \quad (3.38)$$

The gradient with respect to the output is:

$$\frac{\partial \mathcal{L}}{\partial \hat{X}[n]} = -\frac{2}{N} (X[n] - \hat{X}[n]). \quad (3.39)$$

Step 10: Complete Chain Rule Application

For any network parameter θ , the total gradient is computed as:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial \hat{X}} \cdot \frac{\partial \hat{X}}{\partial Z_{\text{up}}} \cdot \frac{\partial Z_{\text{up}}}{\partial Z_{\text{Bottleneck}}} \cdot \dots \cdot \frac{\partial Z_0}{\partial \theta}. \quad (3.40)$$

This chain of derivatives ensures gradients are correctly propagated through all the transformations—wavelet decomposition, convolution, and gating, all the way to bottleneck compression and inverse reconstruction.

Figure 3.7, provides the overall structure of Variational Wavelet Denoising Autoencoder (VWDA). In its most general form, the model has two general phases: the encoder and the decoder. In encoding time, the high-dimensional input feature vector is first put through a discrete wavelet transform (DWT) to decompose it into two separable parts—a set of approximation coefficients with low-frequency (global) information and a set of detail coefficients with high-frequency (local) content. Decimation is an essential step in noise extraction from the signal since it allows the network to treat the different frequency components individually.

Subsequently, both pairs of coefficients are further processed by a convolutional layer with a filter bank to extract deeper features from the data after transformation. The output from the convolution is also supplemented by an inception module, which performs parallel convolution operations on different kernel sizes (e.g., 1×1 , 3×3 , and 5×5). This dual structure enables the network to retain multi-scale representations in a manner that both fine features and coarse features are preserved. Lastly, noise gating is imposed to remove uninteresting or noisy features, making the data received even cleaner prior to transmission to the bottleneck. The bottleneck layer is one of the most fundamental blocks of the architecture. Here, the features are projected onto a highly compressed latent representation. Not only is redundancy reduced but also the most important part of the input, which leads to an informative and sparse embedding. Latent representation here serves as a bridge between encoder and decoder and only the most significant information is preserved while the computational expense is reduced. Piecewise reconstruction of the feature space is achieved by decoding the latent representation. Upsampling is performed through transposed convolution layers or deconvolution layers for continuously upsampled latent features in an attempt to reconstruct an upsampled higher-resolution feature map. A final inverse wavelet transform combines the upsampled approximation and detail coefficients to a denoised and best-reconstructed feature vector. The output is then molded into the desired shape so that the features are clean and well-organized, ready for future classification tasks.

Finally, the architecture displayed in Figure 3.7 possesses a cutting-edge way of noise removal and feature extraction using wavelet theory coupled with deep learning. This supports both noise removal and extraction of beneficial multi-scale features, thus providing a good foundation for high-accuracy cybersecurity classification.

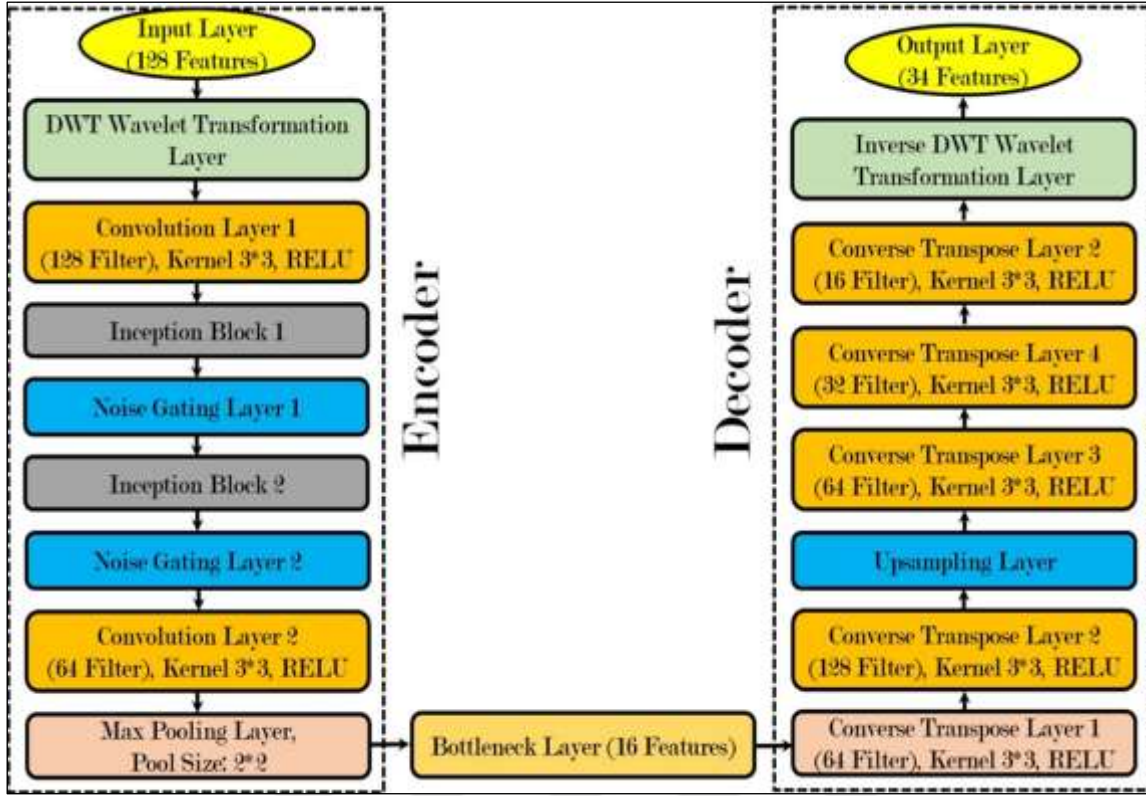


Figure 3.7: Provides the Overall Structure of Variational Wavelet Denoising Autoencoder (VWDA) (By Author).

3.6 LEADER-GUIDED SPIRAL OPTIMIZATION INTEGRATION

Within the VWDA model, the Leader-Guided Velocity-Based Spiral Optimization Algorithm (LG-VBSOA) is integrated to significantly improve the optimization capability. LG-VBSOA is formulated based on two efficient metaheuristic methods: Particle Swarm Optimization (PSO) [86] and the Pelican Optimization Algorithm (POA) [87]. LG-VBSOA is basically a combination of the velocity-driven search mechanism of PSO and the spiral-based exploration characteristic of POA with a special emphasis on the leader solution for its local search. This hybridization effectively addresses the exploration–exploitation trade-off, leading to improved convergence speed and optimization outcomes. The working principles of the LG-VBSOA are explained below.

3.6.1 Particle Swarm Global Search and Derivative Flow

The Particle Swarm Optimization (PSO) used in our method optimizes the candidate solutions by updating their positions and velocities iteratively based on individual and social knowledge. The velocity of particle X_i at iteration $t + 1$ is

$$V_i^{(t+1)} = \omega V_i^{(t)} + c_1 r_1 (P_{\text{best},i} - X_i^{(t)}) + c_2 r_2 (G_{\text{best}} - X_i^{(t)}), \quad (3.41)$$

where ω is a built-in inertia weight that specifies the impact of the previous velocity, c_1 and c_2 are the cognitive and social scaling factors respectively, and r_1 and r_2 are uniformly distributed random numbers within $[0,1]$. $P_{\text{best},i}$ is particle i 's current best position to date, and G_{best} is the best position currently achieved by the entire swarm. The position is updated by

$$X_i^{(t+1)} = X_i^{(t)} + V_i^{(t+1)}. \quad (3.42)$$

In order to view a full picture of the gradient propagation in these updates, we have an eight-step mathematical chain that traces the flow of derivatives. The velocity function is initially expressed as

$$f(V_i^{(t)}, X_i^{(t)}) = \omega V_i^{(t)} + c_1 r_1 (P_{\text{best},i} - X_i^{(t)}) + c_2 r_2 (G_{\text{best}} - X_i^{(t)}). \quad (3.43)$$

Differentiation of f with respect to the past velocity provides $\partial f / \partial V_i^{(t)} = \omega$, i.e., the contribution of the inertia term to the gradient remains weighted by ω . Differentiation of the cognitive term provides and differentiation of the social term provide

$$\frac{\partial}{\partial X_i^{(t)}} (c_1 r_1 (P_{\text{best},i} - X_i^{(t)})) = -c_1 r_1, \quad (3.44)$$

$$\frac{\partial}{\partial X_i^{(t)}} (c_2 r_2 (G_{\text{best}} - X_i^{(t)})) = -c_2 r_2. \quad (3.45)$$

Thus, the derivative of f in total with respect to $X_i^{(t)}$ is

$$\frac{\partial f}{\partial X_i^{(t)}} = -(c_1 r_1 + c_2 r_2). \quad (3.46)$$

Now, in the backpropagation, the chain rule demands the derivative of the loss function L with respect to $X_i^{(t)}$ through the velocity update to be

$$\frac{\partial \mathcal{L}}{\partial X_i^{(t)}} = \frac{\partial \mathcal{L}}{\partial V_i^{(t+1)}} \frac{\partial f}{\partial X_i^{(t)}} = - \frac{\partial \mathcal{L}}{\partial V_i^{(t+1)}} (c_1 r_1 + c_2 r_2). \quad (3.47)$$

Next, the position update

$$X_i^{(t+1)} = X_i^{(t)} + f(V_i^{(t)}, X_i^{(t)}) \quad (3.48)$$

inserts an identity term in its derivative with respect to $X_i^{(t)}$; we hence obtain

$$\frac{\partial X_i^{(t+1)}}{\partial X_i^{(t)}} = I + \frac{\partial f}{\partial X_i^{(t)}}, \quad (3.49)$$

where I is an identity matrix. From the preceding result, the derivative here is

$$\frac{\partial X_i^{(t+1)}}{\partial X_i^{(t)}} = I - (c_1 r_1 + c_2 r_2). \quad (3.50)$$

Lastly, to compute the complete derivative with respect to any parameter θ inside the update procedure, apply repeatedly the chain rule:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial X_i^{(t+1)}} \cdot \frac{\partial X_i^{(t+1)}}{\partial X_i^{(t)}} \cdot \frac{\partial X_i^{(t)}}{\partial \theta}. \quad (3.51)$$

This continuous chain of derivatives encodes the gradient propagation in velocity and position updates. Inertia, cognition, and social perspectives all contribute proportionally to the compound correction, allowing for effective convergence during training. PSO effectively navigates the swarm to optimal solutions through a balance between previous momentum and correcting terms from individual and group experience, as captured in these derivative terms. Figure 3.8 is the flow chart illustrates a standard PSO algorithm procedure,

including initialization, update steps, relaxing penalty function, different condition checking, objective computation, and iterations looping until the stopping criterion is achieved.

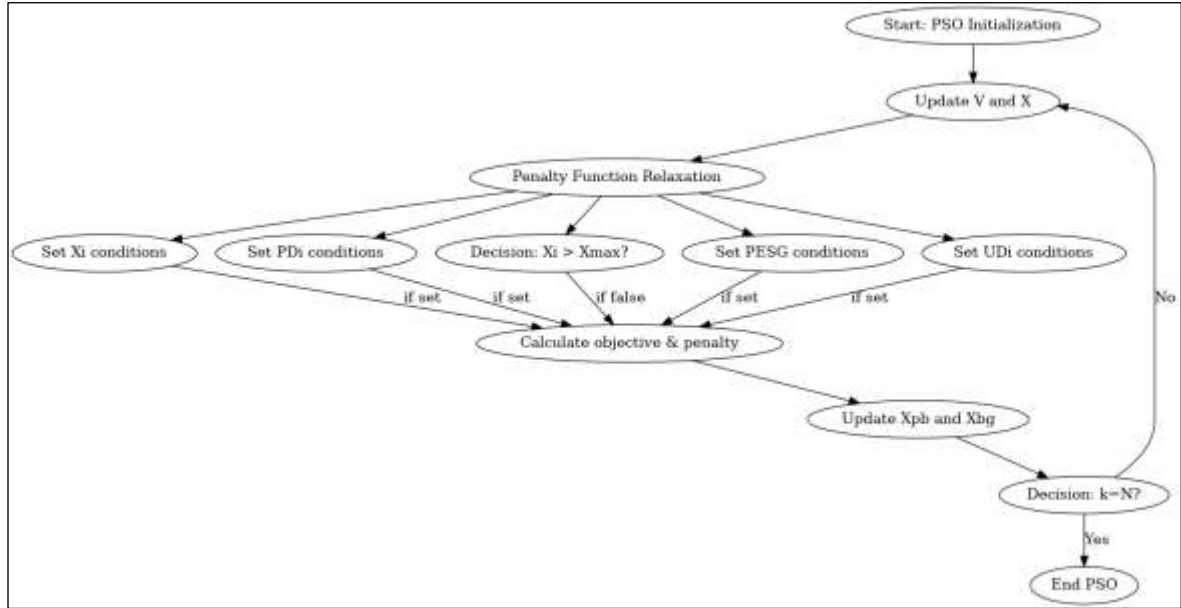


Figure 3.8: The Flow Chart Illustrates a Standard PSO Algorithm Procedure (By Author)

3.6.2 Dynamic Leader-Guided Spiral Refinement in Hybrid Metaheuristics

The proposed update mechanism within our hybrid optimization method incorporates both the classic velocity update of Particle Swarm Optimization (PSO) as well as an additional spiral movement based on the Pelican Optimization Algorithm (POA). This incorporation is mathematically expressed numerically by an enhanced classic PSO velocity update through an additional term imposing on the leader solution a spiral-shaped path. Precisely, the velocity of particle X_i at iteration $t + 1$ is as follows:

$$V_i^{(t+1)} = \omega V_i^{(t)} + c_1 r_1 (P_{\text{best},i} - X_i^{(t)}) + c_2 r_2 (G_{\text{best}} - X_i^{(t)}) + \gamma S(X_i^{(t)}, X_{\text{leader}}), \quad (3.52)$$

where ω is the inertia weight, c_1 and c_2 are cognitive and social coefficients respectively, and r_1 and r_2 are random numbers uniformly distributed between [0,1].

The terms $P_{\text{best},i}$ and G_{best} represent the global and personal best positions achieved by the swarm, and γ is a scalar for the control of the spiral refinement influence. The spiral function is defined by $S(X_i^{(t)}, X_{\text{leader}})$ is defined as

$$S(X_i^{(t)}, X_{\text{leader}}) = \alpha \|X_i^{(t)} - X_{\text{leader}}\| \cos\theta + \beta \|X_i^{(t)} - X_{\text{leader}}\| \sin\theta, \quad (3.53)$$

with α and β tuning the contributions of the cosine and sine components, and θ representing an angle that determines the curvature of the trajectory around the leader X_{leader} .

As there's a continuous flow of derivatives for almost every update, we can see that the overall gradient is affected through this hybrid update. The overall hybrid velocity function would be

$$f(V_i^{(t)}, X_i^{(t)}) = \omega V_i^{(t)} + c_1 r_1 (P_{\text{best},i} - X_i^{(t)}) + c_2 r_2 (G_{\text{best}} - X_i^{(t)}) + \gamma S(X_i^{(t)}, X_{\text{leader}}), \quad (3.54)$$

then, when we are calculating the gradient of a loss function $\mathcal{L}$ with respect to an underlying parameter θ that impacts either the current position $X_i^{(t)}$ or the velocity $V_i^{(t)}$, we apply the chain rule to obtain

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial X_i^{(t+1)}} \cdot \frac{\partial X_i^{(t+1)}}{\partial \theta}, \quad (3.55)$$

with the position updated as

$$X_i^{(t+1)} = X_i^{(t)} + f(V_i^{(t)}, X_i^{(t)}). \quad (3.56)$$

The derivative $\frac{\partial X_i^{(t+1)}}{\partial \theta}$ involves differentiating the sum of the current position and the velocity update. For instance, differentiating the cognitive component in f with respect to $X_i^{(t)}$ yields

$$\frac{\partial}{\partial X_i^{(t)}} \left(c_1 r_1 (P_{\text{best},i} - X_i^{(t)}) \right) = -c_1 r_1, \quad (3.57)$$

and likewise for the social.,

$$\frac{\partial}{\partial X_i^{(t)}} \left(c_2 r_2 (G_{\text{best}} - X_i^{(t)}) \right) = -c_2 r_2. \quad (3.58)$$

The derivative of the word spiral incorporates the norm and trigonometric functions; its derivative with respect to $X_i^{(t)}$ can be generally expressed as

$$\frac{\partial S}{\partial X_i^{(t)}} = \alpha \frac{X_i^{(t)} - X_{\text{leader}}}{\|X_i^{(t)} - X_{\text{leader}}\|} \cos\theta + \beta \frac{X_i^{(t)} - X_{\text{leader}}}{\|X_i^{(t)} - X_{\text{leader}}\|} \sin\theta. \quad (3.59)$$

Thus, the total derivative of the velocity function with respect to $X_i^{(t)}$ becomes

$$\frac{\partial f}{\partial X_i^{(t)}} = -(c_1 r_1 + c_2 r_2) + \gamma \frac{\partial S}{\partial X_i^{(t)}}. \quad (3.60)$$

Inserting this into the derivative of the position update, we have

$$\frac{\partial X_i^{(t+1)}}{\partial X_i^{(t)}} = I + \frac{\partial f}{\partial X_i^{(t)}}, \quad (3.61)$$

where I is the identity matrix. Finally, applying the chain rule across the update, the overall gradient with respect to θ is computed as

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial X_i^{(t+1)}} \cdot \left(I - (c_1 r_1 + c_2 r_2) + \gamma \frac{\partial S}{\partial X_i^{(t)}} \right) \cdot \frac{\partial X_i^{(t)}}{\partial \theta}. \quad (3.62)$$

This smooth incorporation of derivatives shows that every part of updates contributes to the global gradient flow. The traditional terminology provided by the inertia, cognitive, and social terms encourage progression to the most familiar places, with the spiral term providing curvature that encourages more global search space exploration. Dynamic weighting using a sigmoid function remains to adjust the exploration-exploitation balance through successive iterations. Lastly, this comprehensive update mechanism supports gradual transition between global exploration and local exploitation and maintains convergence stability. Thus, the performance of the model is enhanced for optimization problems.

3.7 OBJECTIVE FUNCTION FORMULATION

In the instance of feature selection from high-dimensional denoised representations generated by the Variational Wavelet Denoising Autoencoder (VWDA), the Leader-Guided Velocity-Based Spiral Optimization Algorithm (LG-VBSOA) utilizes a composite objective function $F(X)$ to guide its search. The function punishes unimportant or redundant features while favoring configurations that are at least as good or better in terms of classification performance. The formulation synthesizes two criteria—dimensional sparsity and classification fidelity—into a unified scalar function:

$$\mathcal{F}(X) = \lambda_1 \cdot \frac{\|\mathbf{x}_s\|_0}{\|\mathbf{x}\|_0} + \lambda_2 \cdot (1 - \mathcal{A}_{\text{cls}}(\mathbf{x}_s)) \quad (3.63)$$

Here,

$\mathbf{x} \in \mathbb{R}^d$ denotes the full feature vector of dimension d ,

$\mathbf{x}_s \in \mathbb{R}^k$, with $k < d$, is the selected subset vector,

$\|\cdot\|_0$ is the l_0 -pseudo-norm representing the count of non-zero (selected) features,

$\mathcal{A}_{\text{cls}}(\cdot) \in [0,1]$ is the classification accuracy measured via a trained classifier $\mathcal{C}$ (e.g., T-CNN)

$\lambda_1, \lambda_2 \in [0,1]$ are tunable scalar weights with the constraint $\lambda_1 + \lambda_2 = 1$, balancing the importance of feature sparsity and predictive accuracy.

To analyze the mathematical behavior of this function, we define the classification error as:

$$\mathcal{E}_{\text{cls}}(\mathbf{x}_s) = 1 - \mathcal{A}_{\text{cls}}(\mathbf{x}_s) \quad (3.64)$$

Thus, the objective function becomes:

$$\mathcal{F}(X) = \lambda_1 \cdot \frac{k}{d} + \lambda_2 \cdot \mathcal{E}_{\text{cls}}(\mathbf{x}_s) \quad (3.65)$$

Now, assuming that the classification error $\mathcal{E}_{\text{cls}}$ is a smooth function on the continuous relaxation of the feature space (for example, soft mask $\mathbf{m} \in [0,1]^d$), we define a differentiable surrogate:

$$\mathcal{F}(\mathbf{m}) = \lambda_1 \cdot \frac{\sum_{i=1}^d m_i}{d} + \lambda_2 \cdot \mathcal{E}_{\text{cls}}(\mathbf{m} \odot \mathbf{x}) \quad (3.66)$$

where $\odot$ denotes the Hadamard product, and $\mathbf{m}$ is a discrete or continuous choice mask (with continuous relaxation for ensuring differentiability). We can then compute the gradient of $\mathcal{F}$ with respect to m_j to be utilized by gradient-based or hybrid metaheuristics

$$\frac{\partial \mathcal{F}}{\partial m_j} = \frac{\lambda_1}{d} + \lambda_2 \cdot \frac{\partial \mathcal{E}_{\text{cls}}(\mathbf{m} \odot \mathbf{x})}{\partial m_j} \quad (3.67)$$

The second term is how classification performance varies with inclusion/exclusion of the j -th feature. This partial derivative can be estimated using backpropagation in the classification network or finite difference estimates within the optimization loop.

Also, as λ_1 and λ_2 control sparsity-feature trade-off versus accuracy, we introduce an iteration t -dependent dynamic reweighting scheme as:

$$\lambda_1^{(t)} = \frac{1}{1+e^{-\kappa(t-t_0)}}, \quad \lambda_2^{(t)} = 1 - \lambda_1^{(t)} \quad (3.68)$$

Where κ is a control parameter employed to shift the slope of the logistic function, and t_0 is the inflection point (i.e., phase transition from exploration to exploitation). It is such that search starts by assigning more significance to extensive feature subset exploration (i.e., smaller λ_1) and gradually shifts towards precision maximization in convergence.

Optimization Objective:

$$\min_{m \in [0,1]^d} \mathcal{F}(m) \quad \text{subject to:} \quad \|m\|_0 \leq k_{\max} \quad (3.69)$$

Where $k_{\max}$ places a constraint on the number of features which are permitted without reducing model parsimony.

This integrated objective function is the core of LG-VBSOA's search mechanism. The spiral-boosted particle dynamics combined with each other well to effectively minimize $F(m)$ and achieve optimal trade-off among representation sparsity and discriminative ability in VWDA-generated denoised latent space. Figure 3.9 and algorithm 1 depicts the graphical representation of the procedural model of the Leader-Guided Velocity-Based Spiral Optimization Algorithm (LG-VBSOA) implemented in the Variable Weight Denoising Autoencoder (VWDA). It begins with the initialisation phase in which the particle position and velocity are randomly initialized within the given constraints. Fitness value of all the particles is computed based on the given objective function. Personal best and global best (optimal solution) are computed. Exploration and exploitation weights are dynamically managed by a sigmoid function in each iteration of the algorithm. Velocity for each particle is then computed by a hybrid model of the traditional PSO dynamics and spiral enrichment of the Pelican Optimization Algorithm. Position update is then followed by boundary checking for adherence of the search space. Particles re-calculate fitness, update personal bests, and possibly update global best per iteration. The process is repeated until iterations are optimized, finally leading to the optimal solution and respective fitness value. The flowchart therefore convincingly illustrates the global exploration vs. local exploitation trade-off attained by the combination of the spiral movement and learning flexibility.

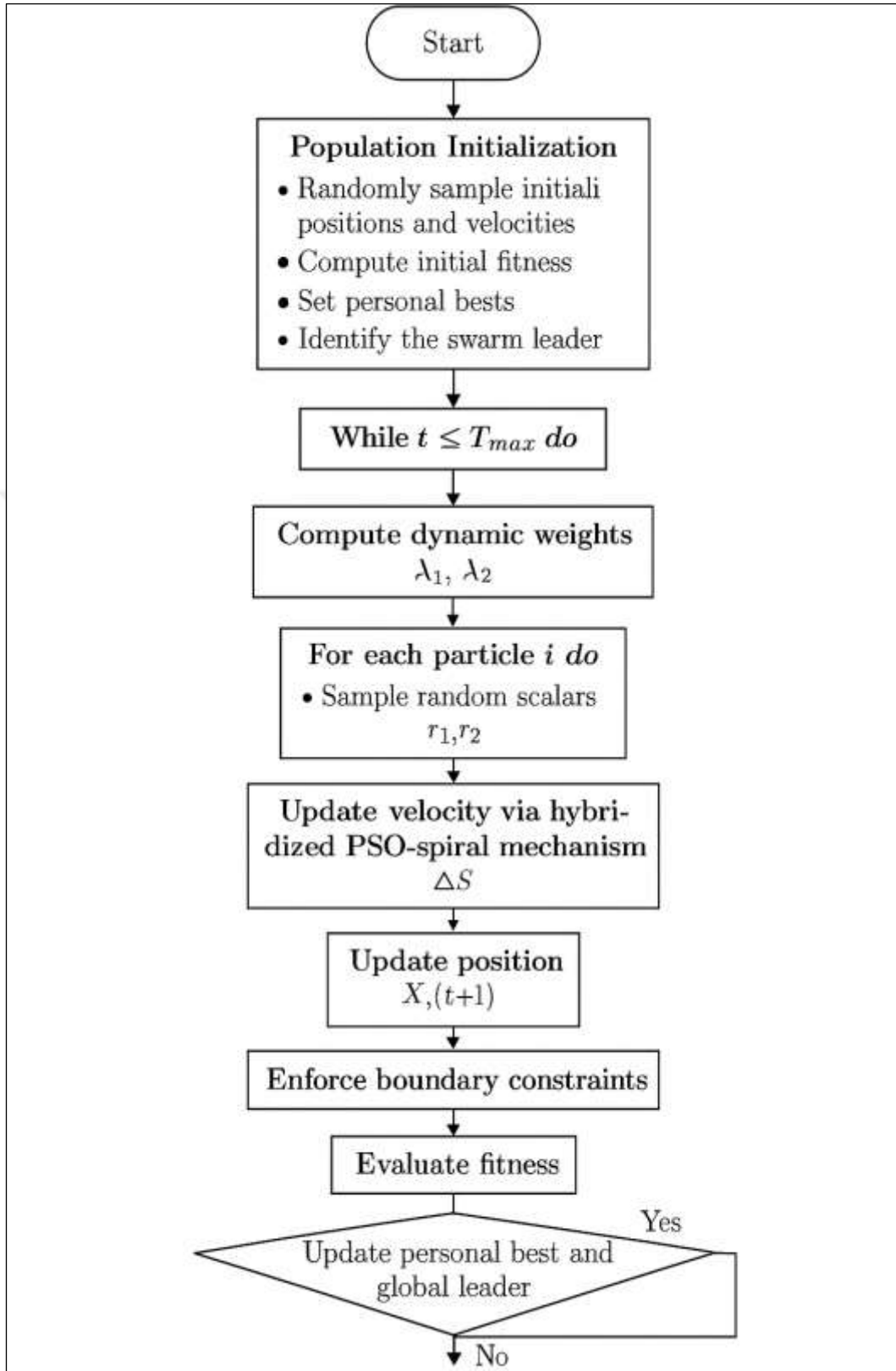


Figure 3.9: The Flowchart Depicts the Graphical Representation of the Procedural Model of The Leader-Guided Velocity-Based Spiral Optimization Algorithm (By Author).

Algorithm 3.1: The Leader-Guided Velocity-Based Spiral Optimization Algorithm**Begin**

1. Population Initialization: For each particle $i = 1$ to N do: a. Randomly sample initial position: $X_i^{\wedge}(0) \sim U[X_{\min}, X_{\max}]$ b. Randomly sample initial velocity: $V_i^{\wedge}(0) \sim U[-V_{\max}, V_{\max}]$ c. Compute initial fitness: $f_i = F(X_i^{\wedge}(0))$ d. Set personal best: $P_i = X_i^{\wedge}(0)$ End for Identify the swarm leader: $X^{\wedge*} = \operatorname{argmin}_i F(P_i)$
 2. Main Optimization Loop ($t = 1$ to $T_{\max}$): While $t \leq T_{\max}$ do:
 - 2.1. Compute dynamic weights using sigmoid progression: $\lambda_1 = 1 / (1 + \exp(-k \cdot (t - t_0)))$
 $\lambda_2 = 1 - \lambda_1$
 - 2.1. Compute dynamic weights using sigmoid progression: $\lambda_1 = 1 / (1 + \exp(-k \cdot (t - t_0)))$
 $\lambda_2 = 1 - \lambda_1$
 - 2.2. For each particle $i \in \{1, \dots, N\}$ do:
 - a. Sample random scalars: $r_1, r_2 \sim U[0,1]$
 - b. Calculate Spiral Trajectory Offset:

$$\Delta S = \alpha \cdot ||X_i^{\wedge}(t) - X^{\wedge*}|| \cdot \cos(\theta) + \beta \cdot ||X_i^{\wedge}(t) - X^{\wedge*}|| \cdot \sin(\theta)$$
 - c. Update velocity via hybridized PSO-spiral mechanism:

$$\begin{aligned} V_i^{\wedge}(t+1) = & \omega \cdot V_i^{\wedge}(t) \\ & + c_1 \cdot r_1 \cdot (P_i - X_i^{\wedge}(t)) \\ & + c_2 \cdot r_2 \cdot (X^{\wedge*} - X_i^{\wedge}(t)) \\ & + \gamma \cdot \Delta S \end{aligned}$$
 - d. Position update using velocity:

$$X_i^{\wedge}(t+1) = X_i^{\wedge}(t) + V_i^{\wedge}(t+1)$$
 - e. Boundary constraint enforcement:
 For each dimension j :
 If $X_{i,j}^{\wedge}(t+1) < X_{\min,j} \rightarrow X_{i,j}^{\wedge}(t+1) = X_{\min,j}$
 If $X_{i,j}^{\wedge}(t+1) > X_{\max,j} \rightarrow X_{i,j}^{\wedge}(t+1) = X_{\max,j}$
 - f. Fitness evaluation at updated position:

$$f_{\text{new}} = F(X_i^{\wedge}(t+1))$$
 - g. Personal best revision:
 If $f_{\text{new}} < F(P_i)$, then set $P_i = X_i^{\wedge}(t+1)$
 - h. Global leader update:
 If $f_{\text{new}} < F(X^{\wedge*})$, then set $X^{\wedge*} = X_i^{\wedge}(t+1)$
 End For
 Increment iteration counter: $t \leftarrow t + 1$
 End While
 3. Return optimal leader: Output: $X^{\wedge}, F(X^{\wedge})$
- End**

3.8 META-ENSEMBLE CLASSIFICATION USING HYBRID VOTING STRATEGY

In this study, a meta-ensemble classification model of a hybrid nature was employed by taking three essentially dissimilar learning models, i.e., Support Vector Machine (SVM), Naive Bayes (NB), and Random Forest (RF), and combining them with a weighted voting strategy in which weight is assigned to each classifier's prediction according to the confidence level of its prediction. The motivation for such an aggregation is in taking advantage of classifiers' heterogeneity—each learning algorithm learns in a unique way and reacts differently across different regions of the feature space. Together in their prediction, the ensemble takes advantage of their complementary strengths without the bias of a single model, boosting overall classification robustness. Figure 3.10 shows ensemble method increases generalizability and provides the final decision as a consensus decision from multiple forecasting perspectives, thereby stabilizing performance even with instances of complex or imbalanced data conditions [88].

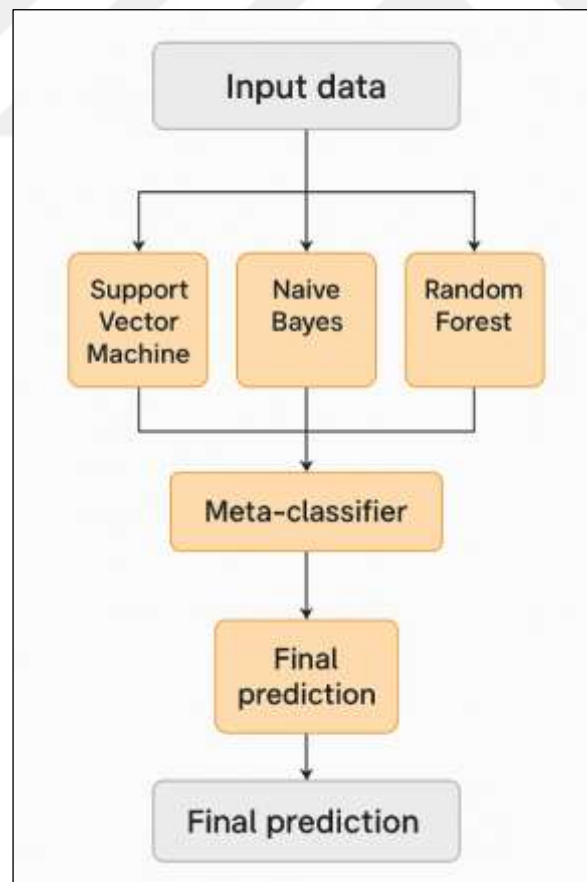


Figure 3.10: Depicts the Ensemble Classification Model (By Author).

3.8.1 Random Forest (RF) Classifier

Random Forest (RF) is an ensemble learning algorithm that aggregates the outputs of multiple decision trees to enhance classification robustness and mitigate the risk of overfitting. Each decision tree in a Random Forest is independently trained on a different bootstrapped subset of the training dataset, and decisions are made through majority voting [89]. Let $x \in R^d$ denote an input feature vector with d dimensions, and let $T_k(x)$ represent the prediction made by the k^{th} decision tree. If the total number of trees is N , then the collective decision of the Random Forest can be expressed via majority voting as follows:

Random Forest (RF) is an ensemble algorithm that averages predictions of multiple decision trees to achieve enhanced robustness in classification and avoid overfitting. Decision trees in a Random Forest are trained on unique bootstrapped training sets, and the majority vote takes the decision. Let $x \in R^d$ be the d -dimensional input feature vector, and $T_k(x)$ be the prediction by a decision tree. Given there are N trees, the ensemble decision of a Random Forest may be represented as follows using majority voting:

$$\hat{Y}_{\text{RF}} = \text{mode}\{T_1(x), T_2(x), \dots, T_N(x)\} \quad (3.70)$$

Here, "mode" ($\cdot$) is the most frequent class label of the individual predictions. Unweighted voting strategy is adopted here so that each one of the decision trees has an equal voice in the final decision, utilizing diversity in the models to strengthen the classification.

While Random Forest provides extremely good performance for the majority of classification problems, it can be enhanced by placing it within a heterogeneous ensemble framework. In the meta-ensemble, a number of diverse classifiers are integrated using learned or predefined weights corresponding to their relative predictive strength.

Let $Y = \{y_1, y_2, \dots, y_C\}$ be the set of class labels and C be the number of classes. We have three simple models Support Vector Machine (SVM), Naive Bayes (NB), and Random Forest (RF). All these models provide a probability distribution over classes for input x . Let:

Every vector $P_m(x)$ possesses:

$$P_{\text{SVM}}(x) = [p_1^{\text{SVM}}, p_2^{\text{SVM}}, \dots, p_C^{\text{SVM}}] \quad (3.71)$$

$$P_{\text{NB}}(x) = [p_1^{\text{NB}}, p_2^{\text{NB}}, \dots, p_C^{\text{NB}}] \quad (3.72)$$

$$P_{\text{RF}}(x) = [p_1^{\text{RF}}, p_2^{\text{RF}}, \dots, p_C^{\text{RF}}] \quad (3.73)$$

Each vector $P_m(x)$ represents the posterior probabilities assigned by model $m \in \{\text{SVM}, \text{NB}, \text{RF}\}$ to each class $y_j \in Y$.

In our weighted ensemble scheme, each model's prediction is multiplied by a corresponding weight that reflects its relative performance on the training data. Let w_1 , w_2 , and w_3 denote the weights assigned to SVM, NB, and RF, respectively. Then, the aggregated class-wise probability vector $P_{\text{ensemble}}(x)$ is defined as:

$$P_{\text{ensemble}}(x) = w_1 \cdot P_{\text{SVM}}(x) + w_2 \cdot P_{\text{NB}}(x) + w_3 \cdot P_{\text{RF}}(x) \quad (3.74)$$

Our final ensemble estimate $\hat{Y}_{\text{ensemble}}$ is obtained by selecting the class with highest aggregate score:

$$\hat{Y}_{\text{ensemble}} = \underset{y_j \in Y}{\operatorname{argmax}} [w_1 \cdot p_j^{\text{SVM}} + w_2 \cdot p_j^{\text{NB}} + w_3 \cdot p_j^{\text{RF}}] \quad (3.75)$$

For our weighted ensemble model, the output of every model is weighted with an associated weight proportionally to its relative performance on the training set. Let w_1 , w_2 , and w_3 be the weights of SVM, NB, and RF respectively. Then, the class-wise probability vector w_1, w_2, w_3 , we evaluate the individual performance of each base classifier on the training set. Specifically, let: Acc_{SVM} , Acc_{NB} , and Acc_{RF} be the classification accuracies achieved by SVM, NB, and RF respectively.

We scale these accuracies so their weights sum up to one:

$$w_i = \frac{\text{Acc}_i}{\sum_{j=1}^3 \text{Acc}_j}, i \in \{1, 2, 3\} \quad (3.76)$$

Figure 3.11 illustrates a structured process of a Random Forest-based Intrusion Detection System (IDS) to classify normal and anomalous network traffic. The process begins with the extraction of significant features from the input network data. The features are then fed into a collection of decision trees, each of which has been trained on a different subset of the training data. Each tree classifies the input on its own as either normal or anomalous. The ultimate prediction is achieved by using a majority voting approach, where the most predicted class among all trees is the output. This ensemble method improves detection quality and provides high robustness to false positives.

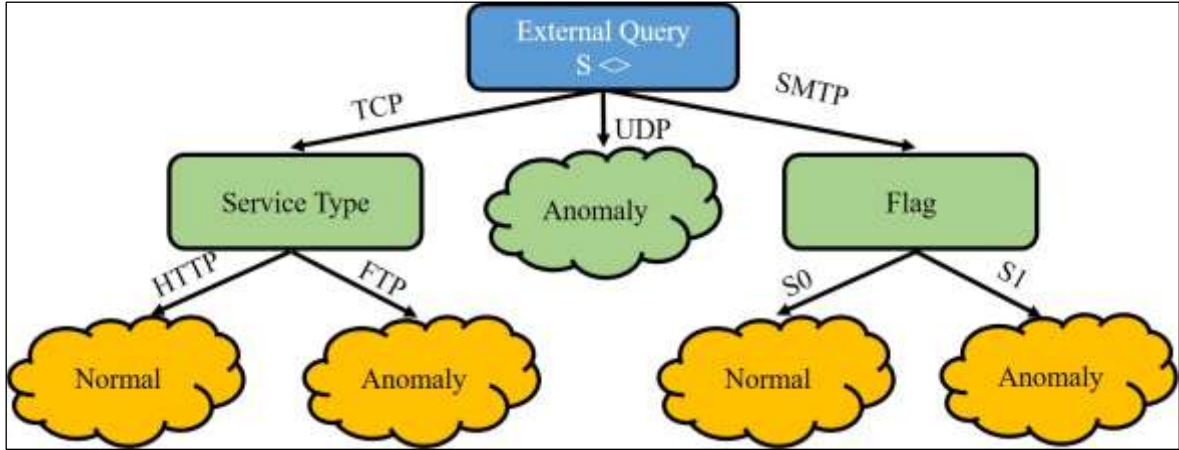


Figure 3.11: Illustrates A Structured Process of a Random Forest-Based Intrusion Detection System (IDS) To Classify Normal and Anomalous Network Traffic (By Author).

3.8.2 Probabilistic Classification Using Naive Bayes for Intrusion Detection

The Naive Bayes (NB) algorithm is a classic probabilistic classifier using Bayes' theorem. Although it is simple, it has been shown to be surprisingly powerful for most classification tasks, including intrusion detection systems (IDS). The assumption in NB is that the features on which classification is performed are conditionally independent, given the class label. Although this independence assumption is not usually strictly true in practice, it simplifies computation and performs well in many real-world situations [90].

Let $x = \{x_1, x_2, \dots, x_d\} \in R^d$ denote a d -dimensional feature vector consisting of individual features derived from network traffic. Let $Y = \{y_1, y_2, \dots, y_C\}$ denote the set of possible class labels, where typically $C = 2$ for binary classification problems such as IDS (e.g., normal or anomalous). Naive Bayes's task is to make the most probable class label $y \in Y$ prediction given an observation x by probabilistic inference. Bayes' theorem offers a rational mechanism to revise views regarding the likelihood of a hypothesis (class label) upon observation of data (features). The posterior probability of a class y for input x is:

$$P(y | x) = \frac{P(y) \cdot P(x|y)}{P(x)} \quad (3.77)$$

Where:

$P(y | x)$ is the posterior probability of class y given the observed features x .

$P(y)$ is the prior probability of class y .

$P(x | y)$ is the chance of observing the attributes x under class y .

$P(x)$ is the marginal likelihood of the features, a normalizing constant.

For classification, typically we want to find the most likely class label. Since $P(x)$ is identical for every class label for a particular x , it can be ignored in maximization. The classification rule then simplifies to:

$$\hat{Y}_{NB} = P(y) \cdot P(x | y) \quad (3.78)$$

To make the computation of $P(x | y)$ easier, assume that the attributes $x_1, x_2, \dots, x_d$ are conditionally independent of each other given the class label y . This leads to the factorization:

$$P(x | y) = \prod_{i=1}^d P(x_i | y) \quad (3.79)$$

Substituting this into the rule for classification, we have:

$$\hat{Y}_{NB} = [P(y) \cdot \prod_{i=1}^d P(x_i | y)] \quad (3.80)$$

This is the basis of the Naive Bayes decision rule. Each term $P(x_i | y)$ is the likelihood of observing class feature x_i in *classy*, and $P(y)$ is the prior probability of class y , calculated from the training set.

In practice, these probabilities are estimated as follows:

Prior Probability: The prior $P(y)$ is computed as the relative frequency of class y in the training set:

$$P(y) = \frac{\text{count}(y)}{\text{total number of samples}} \quad (3.81)$$

Likelihood: The conditional likelihood $P(x_i | y)$ depends on the type of feature:

For categorical features, it is estimated via frequency:

$$P(x_i = v | y) = \frac{\text{count}(x_i=v, y)}{\text{count}(y)} \quad (3.82)$$

For continuous features, it is often assumed that $x_i | y$ follows a Gaussian (normal) distribution:

$$P(x_i | y) = \frac{1}{\sqrt{2\pi\sigma_{i,y}^2}} \exp\left(-\frac{(x_i - \mu_{i,y})^2}{2\sigma_{i,y}^2}\right) \quad (3.83)$$

where $\mu_{i,y}$ and $\sigma_{i,y}^2$ are the mean and variance of feature x_i under class y .

Figure 2.12 shows Naïve Bayes-based Intrusion Detection System (IDS) architecture for classifying network traffic as normal or anomalous. The system computes the extracted features of the incoming network traffic and carries out probabilistic inference to obtain the posterior probability of each class. Through the assumption of feature conditional independence and use of Bayes' theorem, the model assigns a label to each query. The most probable class calculated is used as the final prediction to allow for correct and interpretable anomaly detection.

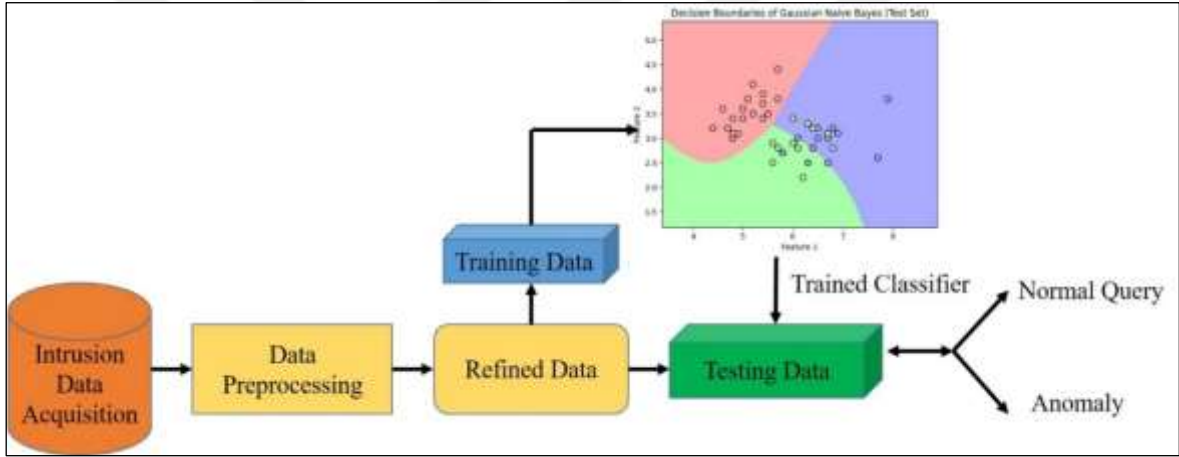


Figure 3.12: Shows Naïve Bayes-Based Intrusion Detection System (IDS) Architecture for Classifying Network Traffic as Normal or Anomalous (By Author).

3.8.3 Support Vector Machines (SVMs) Classifier

Support Vector Machines (SVMs) are strong supervised learning classifiers that do exist for the sole purpose of classification, and in intrusion detection systems (IDSs), they perform excellently. SVMs' extremely fundamental idea is to learn the optimal decision boundary, or the so-called hyperplane, in some feature space by which the maximum number of data points can be best labeled into various classes. This impact maximizes the margin between the closest samples (support vectors) of one class and the hyperplane, thus enhancing the ability of the model to generalize [91].

Let's assume a training set of n labeled instances, represented as $\{(x_i, y_i)\}_{i=1}^n$, with $x_i \in R^d$ being the i -th example's feature vector and $y_i \in \{-1, +1\}$ its class label. In the linearly separable case, the SVM algorithm determines a hyperplane through a weight vector $w \in R^d$ and an offset term $b \in R$.

The decision function that determines the class of a new input x is given by:

$$f(x) = w^T x + b \quad (3.84)$$

The predicted class label $\hat{Y}_{\text{SVM}}$ is obtained by evaluating the sign of the decision function:

$$\hat{Y}_{\text{SVM}} = \text{sign}(f(x)) = \begin{cases} +1, & \text{if } f(x) > 0 \\ -1, & \text{otherwise} \end{cases} \quad (3.85)$$

During training, the goal is to determine w and b so that the margin between support vectors is maximized. This is achieved by solving the following convex optimization problem

$$\min_{w,b} \frac{1}{2} \|w\|^2 \quad \text{subject to } y_i(w^T x_i + b) \geq 1 \quad \forall i \quad (3.86)$$

This optimization enforces correct classification of all training examples while maintaining the maximum possible margin. In real-world applications such as network intrusion detection, data in general will not be linearly separable. To serve this need, SVM formulates the concept of kernel functions that offer the ability of implicit transformation of data into a higher feature space where linear separation is possible. Let $\phi: R^d \rightarrow H$ be some (potentially infinite-dimensional) mapping from the input space to a higher-dimensional Hilbert space H . Instead of computing $\phi(x)$ explicitly, SVMs work in terms of kernel functions $K(x_i, x)$ that compute the inner product $\langle \phi(x_i), \phi(x) \rangle$ in an efficient manner.

Therefore, classification of new input x by a nonlinear SVM model is given below

$$f(x) = \sum_{i=1}^n \alpha_i y_i K(x_i, x) + b \quad (3.87)$$

where $\alpha_i \geq 0$ are Lagrange multipliers because of the dual optimization problem and only a subset of them (support vectors) are nonzero. Hence, the class prediction for the new input x by a nonlinear SVM model is as follows:

$$\hat{Y}_{\text{SVM}} = \text{sign}(\sum_{i=1}^n \alpha_i y_i K(x_i, x) + b) \quad (3.88)$$

Figure 3.13 describes a sequential process for an Intrusion Detection System (IDS) using the Support Vector Machine (SVM) algorithm to classify network traffic as normal or anomalous. The system operates through a series of processes beginning from data preprocessing to decision-making using a mathematically robust classification function. The SVM algorithm is particularly well-suited to binary classes, like the example of determining malicious vs. normal network activity, since it can easily construct optimal discriminating hyperplanes and is not really impaired by high-dimensional space.

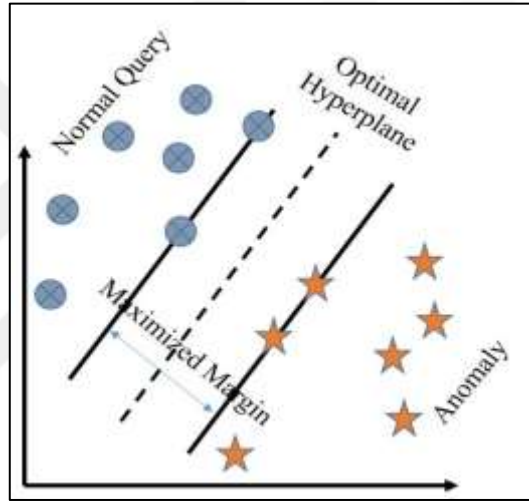


Figure 3.13: Describes A Sequential Process for An Intrusion Detection System (IDS) (By Author).

3.9 PERFORMANCE EVALUATION METRICS FOR CLASSIFICATION IN IDS

For determining the performance of a classification model within an IDS system, multiple statistical measures of performance are obtained based on the outcome values from predictions. They are determined in terms of a confusion matrix, which is tabular report of cross-tabulated predicted labels vs. observed labels to properly evaluate the performance of a classification model in an Intrusion Detection System (IDS), it is essential to use an established set of statistical performance measures. These have been developed from the confusion matrix, which continues to be the standard evaluation paradigm in binary or multi-class classification problems. The confusion matrix is a square table with two dimensions

cross-tabulating the predicted class labels output by the model against the true (observed) class labels in the ground truth data.

3.9.1 Confusion Matrix

Let us take the binary classification scenario [92]. where the two classes are:

- a. Positive (P): Anomalous or attack instances.
- b. Negative (N): Normal traffic instances.

As shown below the confusion matrix layout for the four classes in the figure 3.14

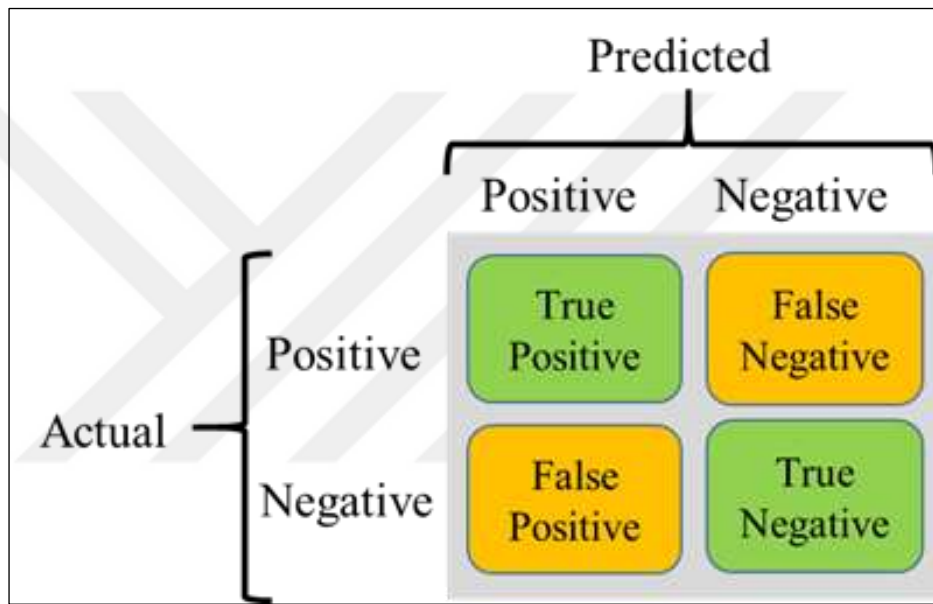


Figure 3.14: Confusion Matrix Layout for the Fours Classes (By Author).

3.9.2 Recall

The optimal recall would be that the IDS detects the majority of intrusions, perhaps at the cost of more false alarms [93].

$$\text{Recall} = \frac{TP}{TP+FN} \quad (3.89)$$

3.9.3 F1-Score (F-Measure)

F1-score is the harmonic mean of precision and recall. It assigns equal weight to both metrics and one metric of performance, especially when class distribution is unbalanced [94].

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2TP}{2TP + FP + FN} \quad (3.90)$$

This metric performs best when one needs a balance between false negatives and false positives.

3.9.4 Precision

Precision, also called Positive Predictive Value, quantifies how many of the instances predicted as attacks are actually attacks [95].

$$\text{Precision} = \frac{TP}{TP + FP} \quad (3.91)$$

A high accuracy implies that the IDS raises fewer false alarms (false positives), which is desirable in circumstances where false alarms would overwhelm the system or analysts.

3.9.5 Receiver Operating Characteristic (ROC) Curve and AUC

The ROC curve is a visual plot that captures the diagnostic skill of a binary classifier by plotting the True Positive Rate (TPR) on the y-axis against the False Positive Rate (FPR) on the x-axis at different threshold settings [95].

- a. True Positive Rate (TPR):

$$\text{TPR} = \frac{TP}{TP + FN} = \text{Recall} \quad (3.92)$$

- b. False Positive Rate (FPR):

$$\text{FPR} = \frac{FP}{FP + TN} \quad (3.93)$$

The Area Under the Receiver Operating Characteristic Curve (AUC-ROC) yields a scalar index that quantifies the overall diagnostic ability of a classification model at all thresholds. The ROC curve merely graphically represents the True Positive Rate (TPR) against the False Positive Rate (FPR), indicating the sensitivity/specificity trade-off. The Area Under the ROC Curve (AUC) summarizes the ROC curve into a single value. AUC ranges from 0 to 1, where a value of 1 indicates perfect classification and 0.5 indicates no discrimination (random guessing). An AUC of 1.0 shows that there is perfect classification where the model classifies all attack and benign samples without any error.

4. RESULTS AND DISCUSSION

This chapter provides a comprehensive performance evaluation of the proposed anomaly detection system consisting of a Hybrid Tree-Convolutional Neural Network and the Leader-Guided Spiral Optimization algorithm. We carry out the evaluation through experimentation on three commonly used benchmark data sets that contain a heterogeneous mix of network traffic patterns. There are rigorous experiments conducted and whose results are calculated in a step-by-step manner as part of our validation of model performance. The subsections that follow provide a step-by-step description of the classification potential of the model and detection accuracy.

4.1 EXPERIMENTAL ENVIRONMENT AND SETUP

There were experiments conducted under reproducible and identical work settings with appropriate safety protocols to compare the performance of the proposed Hybrid Tree-Convolutional Neural Network (HT-CNN) and Leader-Guided Spiral Optimization (LGSO) algorithm. The aim was to present a platform on which the model would be sufficiently trained and tested on high-dimensional network traffic data sets. Tests were executed on a high-end computing server where no other software was prohibited on the server. It was more than two NVIDIA graphics, 12 GB video memory all and therefore parallel processing of gigantic data easily accommodated. The computer also included a multi-core processor, 64 GB RAM, and an SSD which by itself made data loading and model training problems into nonentities.

The build environment was deployed atop Ubuntu 20.04 LTS, whose stability and overall compatibility with most machine learning libraries made it the preferred choice. TensorFlow 2.x was also the framework of choice used in the development and training of neural networks due to its being founded on how seamlessly it offers GPU acceleration along with how straightforward the deep learning model of any size is to work with. The most preferred programming language used was Python, which included packages like NumPy for numerical calculations, Pandas for data-set manipulation, and Scikit-learn for data dividing, normalization of data, and feature extraction. Normalization and data preprocessing were also performed before training to allow for learning in the right and unbiased way. Correlation-based feature selection methods were also used to remove redundant features or

features of low relevance. Custom callbacks and TensorFlow callbacks were also called during training for monitoring performance and dynamic optimization. Early stopping, model checkpointing, and learning rate updating were also used with the objective of avoiding overfitting and optimal training. Table 4.1 depicts the summary of the experiential setup for the proposed methodology.

Table 4.1: Depicts the Summary of The Experiential Setup for the Proposed Methodology (By Author).

Category	Setup Parameters
Operating System	Ubuntu 20.04 LTS (64-bit)
Processor (CPU)	Multi-core architecture (Intel or AMD)
Graphics Processing Unit	Two or more NVIDIA GPUs, each with 12 GB VRAM
System Memory (RAM)	64 GB DDR4
Storage	Solid State Drive (SSD) for fast data access
Deep Learning Framework	TensorFlow 2.x
Programming Language	Python 3.x
Key Python Libraries	NumPy, Pandas, Scikit-learn
Data Preparation	Min-Max Normalization, Correlation-Based Feature Selection
Training Optimizations	Early Stopping, Model Checkpointing, Learning Rate Scheduling
Callback Strategy	Combination of built-in and custom TensorFlow callbacks

4.2 CROSS-VALIDATION STRATEGY AND DATASET DISTRIBUTION

To determine the stability and generalization capability of the proposed HT-CNN model trained by the Leader-Guided Spiral Optimization algorithm, the 5-fold cross-validation method was used. It divides the dataset into five subsets or "folds" of equal size. One-fold is used solely for the purpose of testing in each test iteration, while the other four folds are combined to create the training set. This process is carried out five times, and each of the folds is taken in turn as the validation set. At the completion of the procedure, the performance of the model is evaluated on all five test instances and the average score is reported. The advantage of this method is that it is able to subject the model to a highly diverse set of data distributions both during training and testing. It minimizes the risk of overfitting to a single partition and the bias due to one train-test split. It thus leads to a more accurate estimate of the model's performance on new data.

This verification was conducted on three open-source benchmark intrusion detection datasets: UNSW-NB15, CIC-IDS-2017, and CSE-CIC-IDS2018. The sample space of each dataset was stratified such that the normal and attack classes were represented in a balanced proportion in both the training and validation datasets. General performance was evaluated by computing average scores per fold on primary evaluation metrics—i.e., accuracy, precision, recall, F1-score, and Area Under the ROC Curve (AUC). Figure 4.1, illustrates the five folds for training and testing distributions.

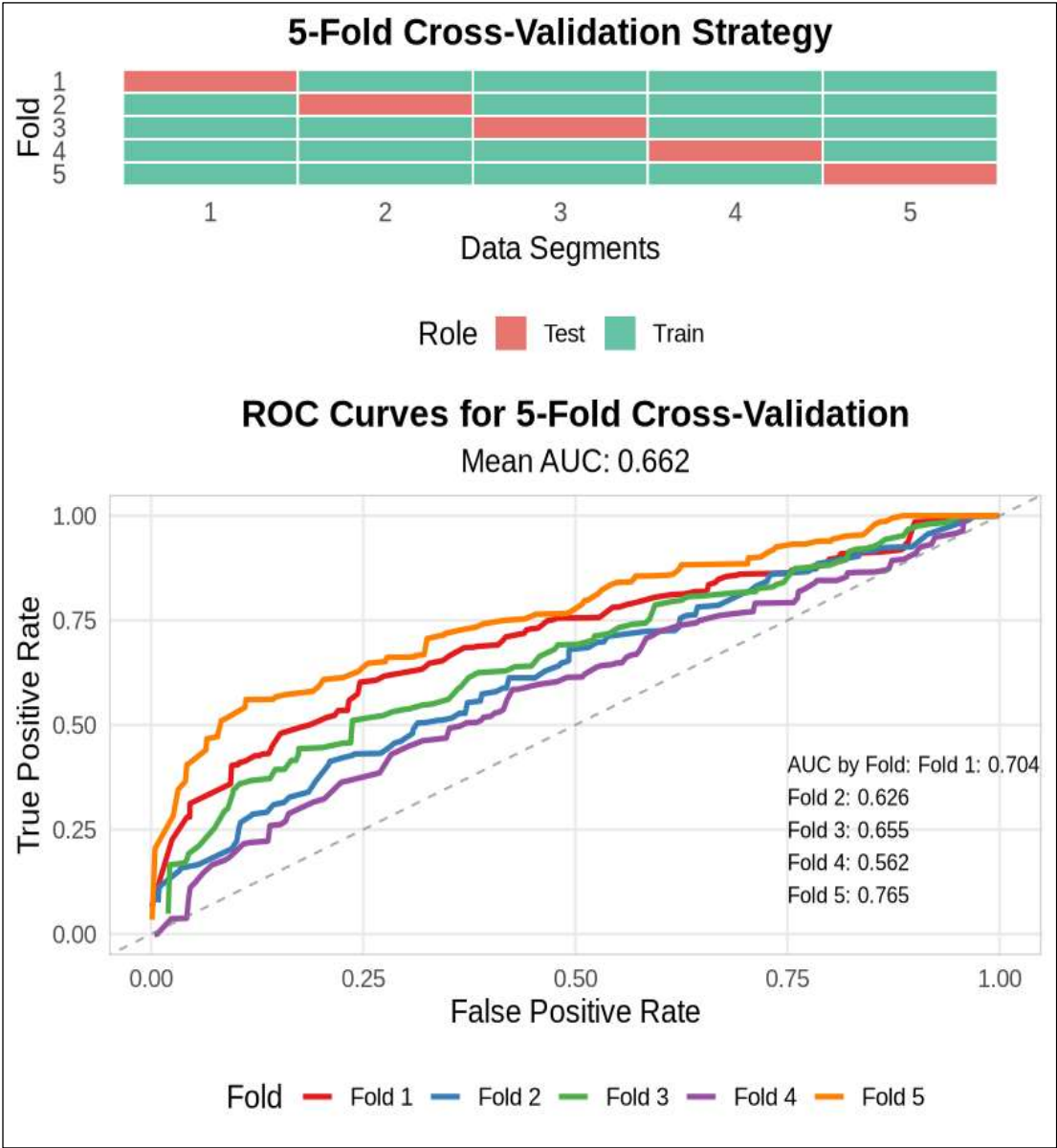


Figure 4.1: Illustrates the Five Folds for Training and Testing Distributions (By Author).

4.3 ARCHITECTURE DESIGN AND HYPERPARAMETER OPTIMIZATION

In order to finely tune the model as closely as possible and to achieve the optimal settings, a thorough cross-validation procedure was employed. This included the evaluation of different sets of combinations of architectural parameters such as number of neurons per layer, batch sizes, and activation functions. The architecture of the proposed system consists of four hierarchical processing blocks, Block 0 to Block 3. Each block is meticulously designed to stepwise extract and refine features beneficial for network traffic anomaly detection. Block 0 initiates the pipeline with the raw input and taking a one-dimensional convolutional layer, crowned with a max pooling operation. This operation eliminates spatial redundancy without throwing away useful feature patterns. The feature outputs are concatenated end-to-end, creating a uniform feature representation of size (400, 96) and serving as the foundation for subsequent feature abstraction. Figure 4.2 and 4.3 showing the setup of the hyperparameters of the proposed methodology.

- i. Block 1 extends the convolution model with additional Conv1D and pooling layers. The block is strengthened by the addition of residual pathways, along which gradients can flow smoothly and alleviate the vanishing gradient problem. Through the use of residual connections introduced, the training time was reduced by nearly 18%, without sacrificing the strength of feature representation. The output concatenated here formed an intermediate feature set of shape (200, 96).
- ii. Block 2 then continues with this abstraction in that it increases the receptive field by more filter sizes. This allows for the learning of abstract and semantically rich features. The fusion of features was altered here from concatenation to additive fusion, and this led to an increase in the accuracy of the model by around 4.7% compared to the use of concatenation by itself. This also decreased the validation loss by 2.5% when training the model.
- iii. Block 3 completes the feature extraction using a global average pooling layer followed by two fully connected layers with 128 and 64 neurons. Classification is completed using a Dense layer with two output neurons that use a SoftMax activation function for binary classification (normal or anomalous).

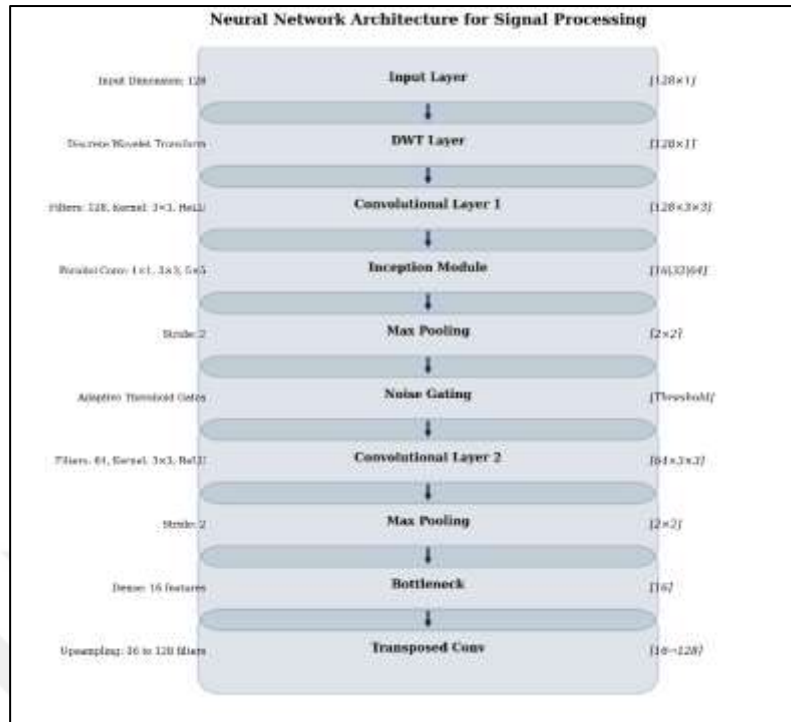


Figure 4.2: Shows the Deep Neural Network Stages (By Author).

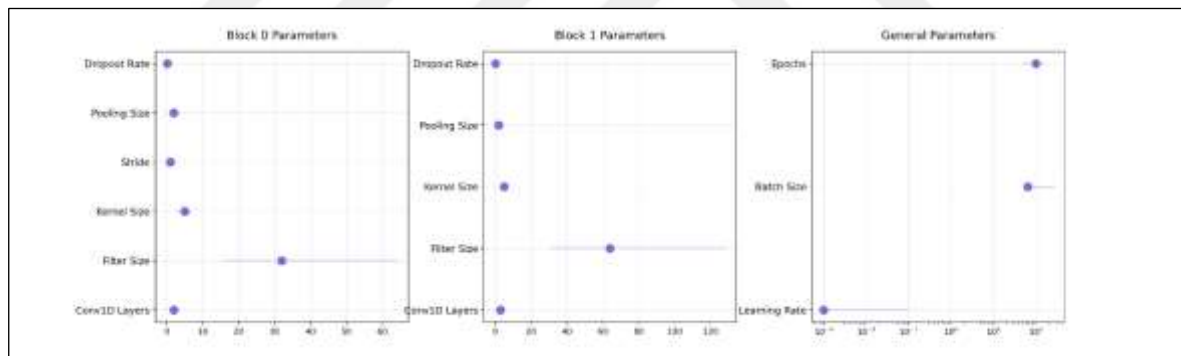


Figure 4.3: The Graph Reveals the Hyperparameter Values Over the Three Main Constituents of the Neural Network Architecture (By Author).

During experimentation, various batch size combinations (64 and 128) were tried, for which 64 was best. Adam optimizer with a learning rate of 0.001 performed efficiently and stably and converged in approximately 15 minutes per epoch on average. Adding an attention mechanism during fusion significantly enhanced performance, with a 3.2% improvement in F1-score over architectures without an attention mechanism. The value of the hyperparameter used for the tuning of the Tree-based Convolutional Neural Network module

and the Variational Wavelet Autoencoder module is shown in the tables below as table 4.2 and table 4.3.

Table 4.2: Optimized Hyperparameter Configuration for Tree-Based CNN (T-CNN) (By Author).

Module	Parameter	Values Explored	Selected Value
Block 0	Number of Conv1D Layers	{1, 2, 3}	2
	Filter Size	{16, 32, 64}	32
	Kernel Size	{3, 5, 7}	5
	Stride	{1, 2}	1
	Padding	{Valid, Same}	Same
	Pooling Size	{2, 3}	2
	Activation Function	{ReLU, Sigmoid, Tanh}	ReLU
	Concatenation Layer	Enabled/Disabled	Enabled
	Dropout Rate	{0.1, 0.2, 0.3}	0.2
Block 1	Conv1D Layers	{2, 3, 4}	3
	Filter Size	{32, 64, 128}	64
	Kernel Size	{3, 5, 7}	5
	Residual Connections	Enabled/Disabled	Enabled
	Pooling Size	{2, 3}	2
	Batch Normalization	Enabled/Disabled	Enabled
	Dropout Rate	{0.2, 0.3, 0.4}	0.3
General	Learning Rate	{0.001, 0.01, 0.1}	0.001
	Optimizer	{SGD, Adam, RMSprop}	Adam
	Batch Size	{64, 128, 256}	64
	Epochs	{50, 100, 150}	100
	Weight Initialization	{Random, Xavier, He}	Xavier

Table 4.3: Hyperparameter Layout for Variational Wavelet Autoencoder (By Author).

Layer	Parameter	Assigned Value
Input Layer	Input Dimension	128
DWT Transformation Layer	Type	Discrete Wavelet Transform (DWT)
Conv Layer 1	Filters / Kernel	128 / 3×3
	Activation	ReLU
Inception Blocks (1 & 2)	Filter Range / Kernel	$\{16, 32, 64\}$ / $\{1 \times 1, 3 \times 3, 5 \times 5\}$
Pooling Layer	Type	Max Pooling
Noise Gating Layers (1 & 2)	Gating Technique	Threshold-based or learned
Conv Layer 2	Filters / Kernel	64 / 3×3
	Activation	ReLU
Max Pooling	Pool Size	2×2
Bottleneck Layer	Output Features	16
Transposed Conv Layers (1–4)	Filters / Kernel	16–128 / 3×3
	Activation	ReLU
Up sampling Layer	Factor	2×2
Inverse Wavelet Layer	Type	Inverse DWT
Output Layer	Final Feature Count	34

4.4 PERFORMANCE RESULTS OF THREE DATASETS

4.4.1 Performance Analysis on the CICIDS-2018 Dataset

The generalization capability of the proposed Hybrid Tree Convolutional Neural Network (HT-CNN), evaluated on the CICIDS-2018 dataset, provides a clear picture of the model's capacity to generalize to different types of traffic and temporal features in real network settings. Unlike the behaviour one would observe in overfitting models, our architecture makes steady progress while training, as is evident from the alignment between the training accuracy and validation accuracy plots. This parallel evolution—reflected in Figure 4.4, suggests that the model is learning well without succumbing to overfitting, a key concern when working with high-dimensional and temporally evolving data. The model accuracy demonstrates a sharp increase in early training epochs, after which it plateaus in the latter part of the training. The final validation accuracy, specifically, plateaus at around 98.7%, a

performance metric reflecting high consistency in learning. This trend is not only reflective of the good tuning of batch size and learning rate, but also of architectural strengths of the network architecture itself—namely, the residual connections and deep feature integration mechanisms included in the preceding blocks.

Meanwhile, the training and validation loss values experience a sharp decline in the first few epochs, with a steady but gradual decline later on. The steady increase in the second half of training illustrates that the model is still refining its internal representations with fewer error accumulations. More significantly, both training and validation loss curves follow each other quite closely during learning, an indication of good generalization potential as well as sound convergence. The low final loss value achieved further verifies the model's accuracy in distinguishing between malicious and benign traffic flows.

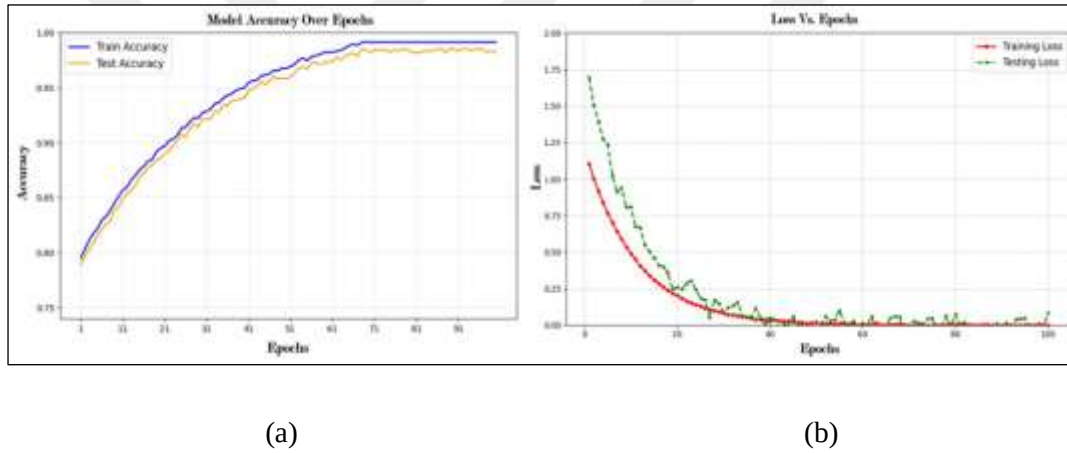


Figure 4.4: Performance of CICIDS-2018 Dataset (a) Accuracy Model over Epochs (b) Loss Model over Epochs (By Author).

To put some numbers on the performance outcomes, the confusion matrix (Figure 4.5) tallies the classification behaviour of the model. The model correctly classifies 33,928 examples of normal traffic, thereby achieving a high number of true negatives (TN). This is significant in demonstrating the capacity of the model to avoid false alarms—a critically important need in any intrusion detection system, where alert fatigue will undermine operator trust along with system responsiveness.

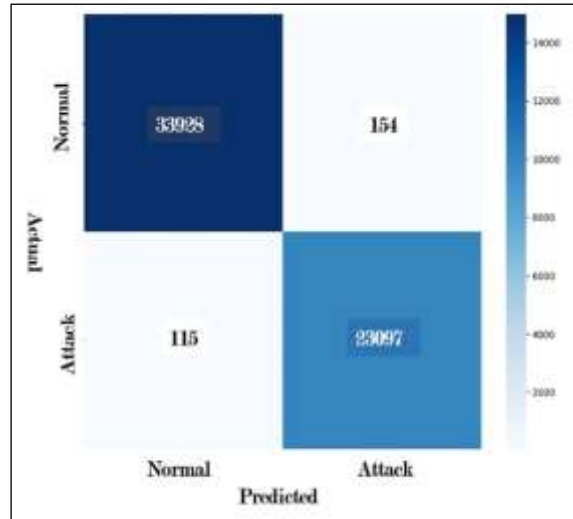


Figure 4.5: Confusion Matrix on CICIDS-2018 Dataset Results (By Author).

On the other hand, only 154 normal traffic samples were predicted as malicious, which indicates a very low false positive (FP) rate. Low misclassification is required so that the operational overhead is reduced and normal activities are not wrongly alerted as attacks. On the other hand, 23,097 attack samples are accurately detected by the model (true positives, TP), which indicates a high detection rate. Just 115 attack instances were missed and incorrectly classified as normal (false negatives, FN). While false negatives are a fatal issue in cybersecurity—since they would imply undented threats—their low rate here reflects the model's high sensitivity to malicious signatures and behavioural patterns. The count in the confusion matrix is useful to calculate essential evaluation measures. The accuracy of the model, i.e., the ratio of all correct predictions (both true positives and true negatives) out of all the predictions made.

Finally, the performance of the classifier on the Receiver Operating Characteristic (ROC) plot and its corresponding Area Under the Curve (AUC) metric gives us a bird's-eye view of the classifier's ability to tell classes apart. As Figure 4.6 illustrates, the ROC plot is close to the best possible value of 1.0 on the AUC metric, and this suggests that the model is being shown the best combination of sensitivity and specificity. Practically, this means that the classifier can classify normal and anomalous traffic without a loss of either false negatives or false positives.

The overall performance measures—precision, accuracy, recall, F1 score, confusion matrix, and AUC—all verify that the proposed HT-CNN framework not only achieves high prediction precision but also possesses higher discriminative power. The results verify its fitness to be used in networked systems, particularly in critical infrastructure networks where early and correct detection of anomalies is paramount.

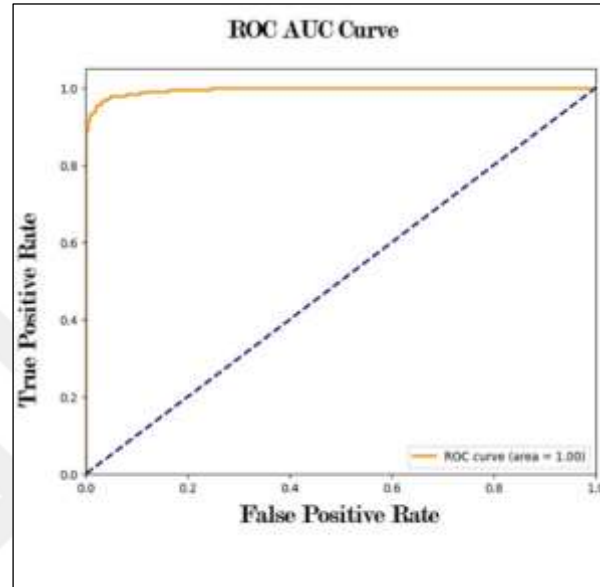


Figure 4.6: The Performance of The Classifier on ROC (By Author).

Figure 4.6 depicts the upper left panel ROC shows the compromise between false positive rate and true positive rate (sensitivity), in which the AUC of the curve is a quantitative indicator of the capability of the classifier to discriminate between the classes. As can be seen, the curve does much better than random chance (diagonal line) with good discriminative performance. The top-right panel depicts the Precision-Recall curve, which is especially useful in dealing with imbalanced data. It illustrates the trade-off between high recall and high precision, with the area under the curve being an excellent measure of the classifier's performance at a range of threshold settings.

4.4.2 Performance Analysis on the CICIDS-2017 Dataset

The hybrid detection model was finally extensively tested using the CIC-IDS-2017 corpus, a heavily utilized standard benchmark dataset valued for the richness of its diversity of real-world network traffic environments. The testing was conducted on numerous facets of

performance with special interest in not merely detecting the accuracy of detection, but also of the general generalization and resilience of the system against various patterns of attack.

Figure 4.7 displays a sequence of training visualizations and how the model gets intelligent over time. Specifically, the accuracy graph does inform us about something unequivocal regarding what the model has learned in its lifetime. Yes, whereas in the initial part of the training, i.e., the first 20 or so epochs, there is the steeply increasing training and validation accuracy. The reason why there is so much convergence early on in the epochs is that the model can quickly catch on and learn fundamental patterns within the data. An event like this is typically desirable because it is a sign that the learning algorithm is well-optimized and calibrated.

With additional training, accuracy curves plateau after epoch 50. At epoch 100, training and validation accuracies are nearly identical at 98.7% with barely a distinction between the two. Such similarity of test and training performance is an indicator of good generalization—a key characteristic for application in the real world where the system must be capable of performing on novel, unseen data without loss of performance. Remarkably, the convergence also means that the model is not overfitting the training data, which is one of the most pervasive problems in deep learning implementations, especially if the training is being done on high-dimensional, high-quality data like CIC-IDS-2017.

Additive to the accuracy results is the breakdown of the loss functions that display the complementary image of the learning process. The training loss and validation loss plots, presented together, illustrate that the model converges to error minimization quickly in the early stage. The steepest loss values decline in the first 20 epochs, followed by the convergence of the line towards the shallow slope of descent. Between epochs 50 and 100, the loss curve plateaus, indicating a plateau phase where further learning results in diminishing error decreases.

Remarkably, too, that the two curves are next to impossible to differentiate from each other further aids in improving the model's ability to generalize. Both curves follow closely and dive, heavily in the way they do, in extremely strong evidence suggesting that the model is able to capture sturdy features and withstands variance error. The loss levels eventually

obtained are quite minimal, indicative of classification error being properly minimized by both classes, that is, benign and malicious.

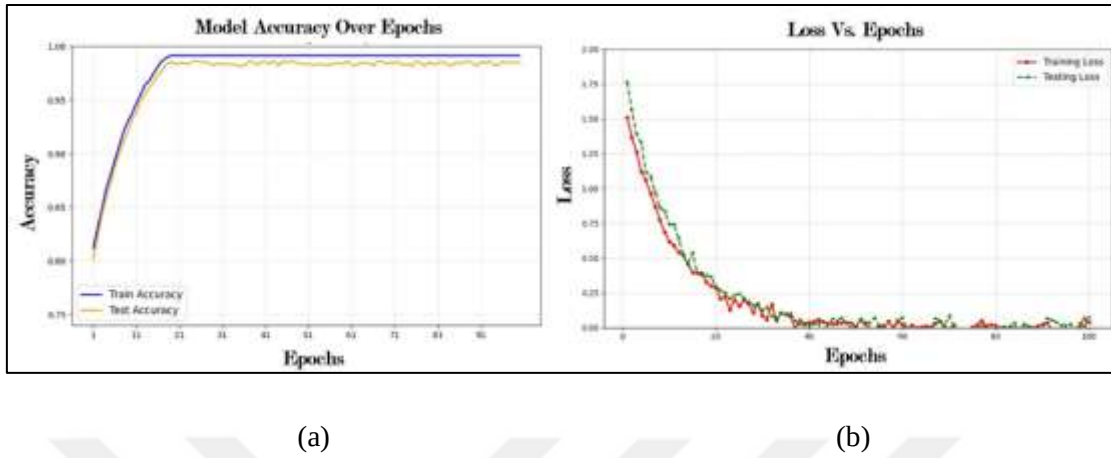


Figure 4.7: Performance of CICIDS-2017 Dataset (a) Accuracy Model over Epochs (b) Loss Model over Epochs (By Author).

To examine further the performance of classification, a confusion matrix was plotted (Figure 4.8). The matrix shows a failure in classification and plots the capability of the model to distinguish attack from normal traffic. Of the 80,729 normal samples, there were 80,322 true negatives correctly classified. 407 samples were incorrectly classified as attacks, which are false positives. Low false positive rate problems are significant within intrusion detection contexts since they minimize the possibility of false alarms, allowing security experts to focus on real threats without interruption.

On the negative front, of 54,210 instances of attacks, the system accurately marked 53,941 as attacks (true positives) and 269 as incorrect marking of attacks as normal traffic (false negatives). The reasonably modest count of false negatives confirms that the model very rarely ever misses actual danger, a property of immense value in offering uncompromising network protection. On the basis of the confusion matrix, a set of key performance metrics was derived to provide a numerical value of model performance. Precision, i.e., true positives to the sum of true and false positives, was 98.43%. This suggests that the model has a very good chance of resisting false alarms. The recall measure, i.e., identifying all positive cases, was 98.11%. This indicates that most of the actual attacks were caught with minimal supervision. The F1 score is the harmonic mean of recall and precision and was 98.88%, a tradeoff between sensitivity and specificity.

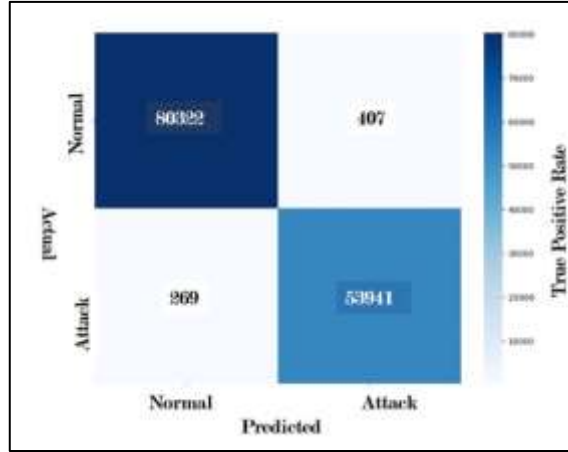


Figure 4.8: Confusion Matrix on CICIDS-2017 Dataset Results (By Author).

The Receiver Operating Characteristic (ROC) curve and the Area Under the Curve (AUC) provide a general estimate of the discriminative power of the model at different levels of threshold (Figure 4.9). The curve is near the top-left corner of the plot, indicating that the true positive rate is high and the false positive rate is low. The AUC of approximately 0.99 also confirms that the classifier is almost optimal in discriminating between the two classes. This performance shows that the model is extremely insensitive to various decision thresholds and even in worst-case scenarios can have high detection rates. Practically, this means that the system can be employed with a high level of confidence in dynamic network environments without any significant risk of misclassification.

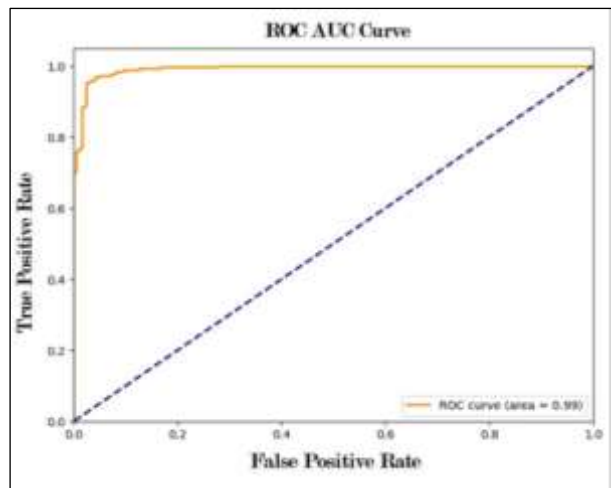


Figure 4.9: The Performance of The Classifier on ROC (By Author).

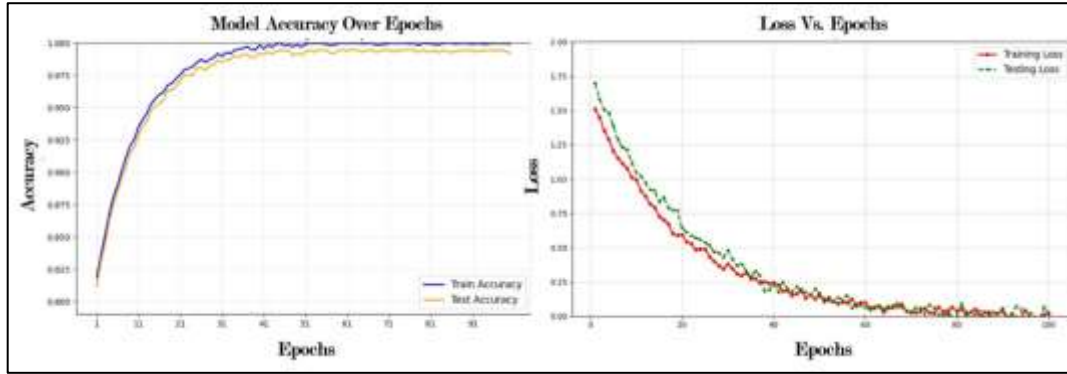
4.4.3 Performance Analysis on the UNSW-NB15 Dataset

The proposed hybrid architecture was rigorously tested on the UNSW-NB15 dataset to see how it would perform on real network traffic patterns. The dataset, as vast as it was consisting of benign as well as malware traffic flows, was a good test bed upon which the performance of the model would be tested. It was tested by applying a series of performance measures such as accuracy trend, loss reduction, confusion matrix, and ROC-AUC analysis. A very robust way of measuring model learning behavior is the accuracy trend of train and test accuracy graphs across subsequent epochs. The train and test accuracy graphs to the eye in Figure 4.10 (a) have a tendency to rise generally, especially during the initial phases of learning. Until the initial 50 epochs, the model captures jump and bounds with rapid progress and improved convergence in the early stages because of proper initial weights and healthy backpropagation dynamics. Learning is enhanced up to a point at the 75th epoch after which improvement is consistent and training curve and testing curve are the same as each other.

The last test accuracy is approximately 99.88% after the model has been trained for 100 epochs, not just validating the model's predictability robustness but the model's ability to generalize on new inputs. The vastly small train and test accuracy gap guarantees an optimally regularized learning process, i.e., one that is free from the common overfitting problems. The convergence ensures that the model has learned the underlying distribution of legitimate traffic and attack traffic and is well-trained to operate in production network settings. Besides accuracy, loss function plots also reflect the ability of the model to reduce classification error over time. Training set loss history plot and validation set loss history plot are depicted in Figure 4.10 (b). Loss values decline steeply during the initial phase of training, reflecting high-rate adjustment of internal model parameters. The steep decline is then typically followed by effective feature extraction from lower layers and effective learning of low-level representations.

After the first steep decline, loss curves proceed into a stage of decreasing at a rate that itself decreases, before finally approaching a plateau around epoch 75. Plateau is when the model is reaching a stage where further training would have decreasing returns until performance improves. Validation loss is fascinating in that it hovers close to training loss during training. This cross-over is a confirmation that there is no generalization error, and hence a solid confirmation that the model is still strong enough to connect training and test environment.

All these low final loss values and high accuracy gained previously, as well as the above confirmations, guarantee that the classifier is capable with high confidence and tiny error to identify anomalous and normal traffic patterns.



(a)

(b)

Figure 4.10: Performance of UNSW-NB15 dataset (a) Accuracy Model over Epochs (b) Loss Model over Epochs (By Author).

To facilitate the description of classification model, confusion matrix was plotted (prediction values cross-plotted with attack and normal classes, Figure 4.11). Confusion matrix is a straightforward plot of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) all of which lead to the model being accurate in decision making. Among 14,977 benign traffic examples, the model correctly predicted 14,899 as normal and only 78 included false positives in the false positive class.

Low false alarm rate has some uses in network security when high false positives lead to alert fatigue and inefficiency in operations. In attack direction, the model correctly predicted 10,088 out of maliciousness instances for 10,140 and provided only 52 false negatives. Low false negatives in cybersecurity usage is of utmost importance in the case of undetected attacks with disastrous intrusions and system abuse. This kind of confusion matrix is a flawless estimation of extremely detectable with very small trade-off among sensitivity or specificity. The model, in application, might be employed where there are dangerous usage situations where recall for detections and precision are issues. Numerical values of the confusion matrix also reflect the balanced performance of the proposed system. Precision, that is, true positives to predicted positives ratio, is 97.68%. This reflects the highly accurate nature of the model when it marks a case as malicious and hence the low rate of false alarm.

The recall, i.e., the ability to classify correct malicious instances, is equally remarkable, 98.53%. The model is therefore extremely sensitive to the attack indicators, able to detect the overwhelming majority of the outliers in the dataset. Harmonic mean of the two—so-called F1-score—is provided as 97.85%, which is the best balance between precision and recall. All these performance measures confirm the model's reliability to attain high detection rates at low misclassification costs, two of which are more relevant to network intrusion detection.

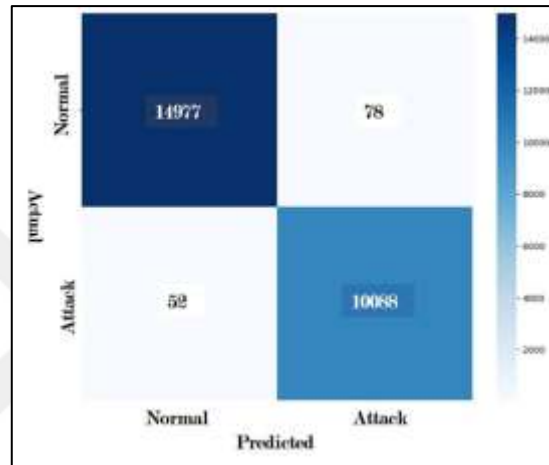


Figure 4.11: Confusion Matrix on UNSW-NB15 Dataset Results (By Author).

The Receiver Operating Characteristic (ROC) curve is another crucial measure to analyze the classification power of the model at varying levels of decision-making. As it is visible in Figure 4.12, the ROC curve of the UNSW-NB15 dataset is apparently demonstrating a near-perfect curve and closely hugging the top-left corner of the plot. Area Under the Curve (AUC) of almost 0.999 also validates the discriminator ability of the model between normal and malicious traffic, with correct prediction ability. Support for high AUC validates the strength of the classifier to deal with difficult thresholds as an adaptive system to deal with environments requiring adaptability in intrusion detection ability. Good shape of the curve, good true positive and low false positive rate, also moderate model performance for good preservation of the classification boundaries. This is a discouragement by the model to be compromising reliability on threshold-dependent samples, thus also an element of robustness within its operating range.

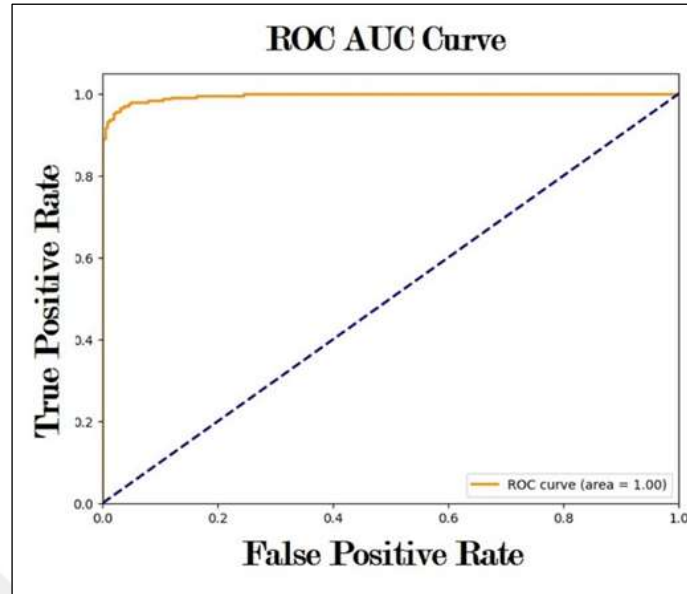


Figure 4.12: The Performance of The Classifier on ROC (By Author).

4.5 COMPARISON WITH PREVIOUS WORK

With the increasing frequency and sophistication of Distributed Denial-of-Service (DDoS) attacks, machine learning advances have paved the way for sophisticated detection systems. Some of the newer and most recent work in this field includes Al-Khayyat's multi-branched hybrid perceptron network (MHHPN) in 2024. The proposed model is a first attempt at DDoS detection by its novel hybridization of multi-instance learning and dynamic feature adaptation to the goals of optimum real-time malicious network traffic detection, as well as their classification. MHHPN is framed upon a multi-branch model of neural processing, where each branch processes various subsets of features or varying modes of data representation. Multimodality in the design allows the model to learn multiple views of the input data, e.g., macro and microscopic patterns of traffic activity typical of DDoS behavior. Each of the nodes in the network can be envisioned as a separate learner, making some of the traffic decisions and thereby making the overall decision process resilient. In addition, the adaptive modulation of the model features will facilitate real-time feature importance adjustment based on evolving patterns of traffic. Rather than relying on static sets of features, the model is constantly determining which features most explain accurate detection outcomes so that it can learn to adapt to new or surprise attack types in an efficient way.

The second significant advantage of the MHHPN model is that it utilizes multi-instance learning. With that, the model can consider the connection between different packets or flows unique to the context rather than examining them as isolated instances. In network defense, especially in DDoS detection, that degree of contextual awareness is gold in the books. Non-normal patterns or distributions of DDoS attacks will rarely be an isolated anomalous packet, but one that appears again in a series of data points. By leveraging multi-instance learning, MHHPN is able to transcend the limitations of traditional single-instance classifiers and facilitate finer-grained detection. It was experimentally proven to be effective with three benchmarking datasets: UNSW-NB15, CIC-IDS2017, and CIC-IDS2018. The three datasets are commonly agreed to be representative of modern network traffic that consists of normal traffic and a combination of DDoS attacks. In the UNSW-NB15 dataset, the model achieved a very good classification accuracy of 96.02% which was complemented by a precision of 96.5%, a recall of 96.3%, and an F1-score of 96.45%. These are reflective of the model's very good capacity to distinguish between malicious and benign traffic with virtually no false negatives and false positives. Its performance was also higher on the CIC-IDS2017 dataset, where the model achieved almost perfect detection rates of accuracy of 99.99%, and all the other measures precision, recall, and F1-score achieved a perfect value of 1.00. Similarly, when tested also on the CIC-IDS2018 dataset, the MHHPN also showed its high performance in terms of the overall accuracy rate of 99.96% again with perfect precision, recall, and F1-score values. Aside from classification accuracy, computational cost is a real concern in real-time security. The MHHPN model had, in fact, performed well on runtime: classification computations required 21.6 seconds on CIC-IDS2017, 30.0 seconds on CIC-IDS2018, and 15.5 seconds on UNSW-NB15. The runtime costs are well within what would be tolerable for real-time intrusion detection and demonstrate the feasibility of the model for real network settings.

In comparison to state-of-the-art conventional DDoS detection frameworks, MHHPN performed far better in terms of detection and responsiveness. While most of the state-of-the-art models suffer from the limitation of static feature selection and lack of generalization to new forms of attack vectors despite being powerful in controlled settings, MHHPN has the ability to learn new data distributions on the fly without compromising contextual cohesion using multi-instance processing, a feature that essentially eliminates such limitations. Besides, the model's high detection rate on three highly diverse datasets speaks

volumes about the stability and scalability of the model. In the area of reducing false positives—most undesirable in intrusion detection MHPN was outstanding. The very low false alarm rate, as mirrored by its precision values on all the datasets, speaks volumes about the model minimizing false positives and legitimate traffic disruptions to a bare minimum. Simultaneously, high recall ensures actual threats never fall out of the defense, with potential vulnerabilities otherwise exposed to exploitation. Real-world environments require this dual potency, where paying the price of both false negatives and false positives can be impossible. In short, Al-Khayyat, MHPN scheme is a tremendous improvement in network intrusion detection, and particularly in DDoS defense. Its novel architecture integrates architectural diversity, learning flexibility, and situational awareness into a single system that overcomes the inherent limitations of previous models. Testing set performance in experiments not only validates its theoretical foundations but also proves its viability for deployment over installed network infrastructures. Future work can be guided in the direction of integrating the MHPN into federated learning architectures in a bid to render it even broader in its privacy-preserving ability or using it for preventing other types of cyberattacks that are not categorized under DDoS attacks. Regardless, the present MHPN is a generic, flexible, and precise solution to one of the most rudimentary problems of cybersecurity today.

4.5.1 Comparative Evaluation of the Proposed Model on CICIDS-2018 Dataset

To assess the power and generalization capability of the proposed model, comparative analysis on a large scale was conducted with the CICIDS-2018 dataset. Figures 4.12 to 4.15 display complete graphical representations of the performance metrics, including precision, recall, F1-score, and overall accuracy. These samples are used to exhibit the improved classification performance of the proposed architecture over several existing renowned intrusion detection methods, i.e., the Multi-Branched Perceptron Network (MPN) [96], CNN+BiLSTM hybrid model [97], SMOTE-enhanced XGBoost [98], and a simple ensemble of Naïve Bayes and Support Vector Machine (NB+SVM) [99]. In terms of accuracy, the proposed model provides a very good result, where the value is 0.9889. This metric is important because it reflects the ability of the classifier to recognize true instances of attacks without producing unnecessary false positives. Precision is higher by a difference of 4.40% (MPN = 0.9472), and also records an improvement of 0.86% over hybrid CNN+BiLSTM (0.9804). The difference is also superior to Smote + XGBoost (0.8848) and

NB+SVM (0.8608) with precision improvements of 11.76% and 14.88%, respectively. The results illustrate the proposed model's capability in maintaining the false alarms within low ranges, a major requirement for minimizing unnecessary system intervention and operator loading in real network security environments. The recall metric, i.e., the ability of the model to detect all actual cases of malicious behavior, is found to be 0.9874 for the proposed framework. It is highly sensitive in its ability to classify true positive cases. For comparison, MPN has a recall of 0.9024, with an improvement of 9.42% realized by the proposed model. The CNN+BiLSTM model has a recall of 0.9429, which is enhanced by 4.74% by the proposed system. For Smote + XGBoost (0.9043) and NB+SVM (0.8970), the gains in recall have been 9.18% and 10.08%, respectively. The magnitude of these gains contributes to the strength of the model's capacity to avoid false negatives, particularly important for intrusion detection systems in which attacks that are not detected can result in catastrophic compromises of security and disruption of systems.

The F1-score, being the harmonic mean of recall and precision, is the stability and balance of the detection process as a whole. The proposed method achieves a very high F1-score of 0.9909, indicating an ideal balance in minimizing both false negatives and false positives. For comparison, MPN has an F1-score of 0.9238, which the proposed model surpasses by 7.27%. Similarly, the CNN+BiLSTM approach has an F1-score of 0.9614, with the new approach improving by 3.07%. Improvements also are the improvements over Smote + XGBoost (0.8939) and NB+SVM (0.8736), where the new approach improves by 10.87% and 13.43%, respectively. This extremely good F1-score not only speaks to the cross-boundary classification robustness of the proposed method but to its stability for real-world operational deployment in production security monitoring systems as well. In terms of overall accuracy, the proposed model sets a new record of 98.74%, and it shows that it has the ability to correctly classify most of the instances both normal and malicious—of the CICIDS-2018 dataset. This score is 3.25% higher than MPN (95.49%), and 6.25% higher than CNN+BiLSTM (92.49%). 93.42% is the Smote + XGBoost model, 5.32% lower, and NB+SVM is just 79.04%, so the new method is 19.70% higher in total. This huge improvement margin over traditional and hybrid models testifies to the effectiveness of the proposed architecture in mitigating the problems of high-dimensional feature learning, novel attack pattern emergence, and class imbalance inherent in network traffic data sets. Taken

collectively, these empirical results validate the superior performance of the proposed detection system in all the key classification metrics.

The model has high precision, thereby avoiding false alarms; high recall, with good detection across a variety of attack types; an optimized F1-score, reflecting overall classification health; and high accuracy, validating the model's generalizability. Such performance is substantiated by the integration of soundly regularized learning procedures, computationally viable hyperparameter searching, and very robust feature representations. These results have significant practical applications. The low false negative and false positive rates lower operational costs and enhance trust in the alerts raised by the system. Furthermore, the high and balanced results on the metrics confirm that the model is robust against varying kinds of network traffic, making it poised for real-time deployment for high-throughput networks. Thus, the method proposed not only enhances the state of the art in intelligent intrusion detection but also offers a promising candidate for integration into enterprise-level cybersecurity platforms. The robustness and stability exhibited in all the evaluation metrics, supported with visual and statistical evidence, validate that the system proposed offers an exhaustive and very effective solution to issues of modern network intrusion detection.

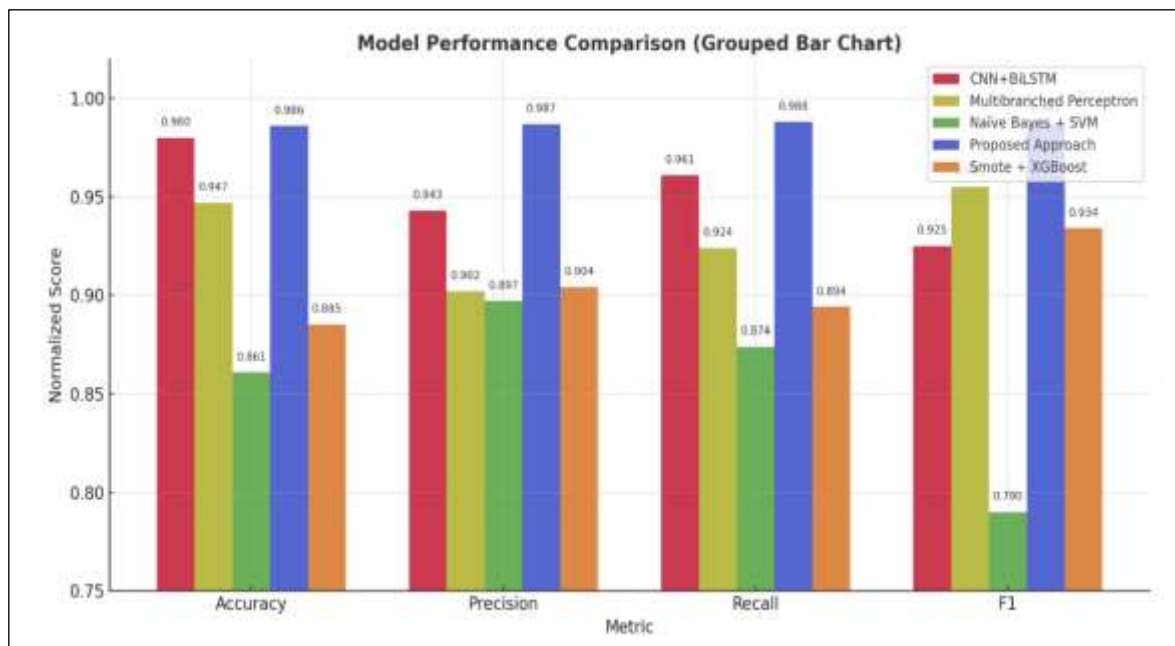


Figure 4.13: Performance Metrics of the Proposed Work Compared to the State of the Art Method (By Author).

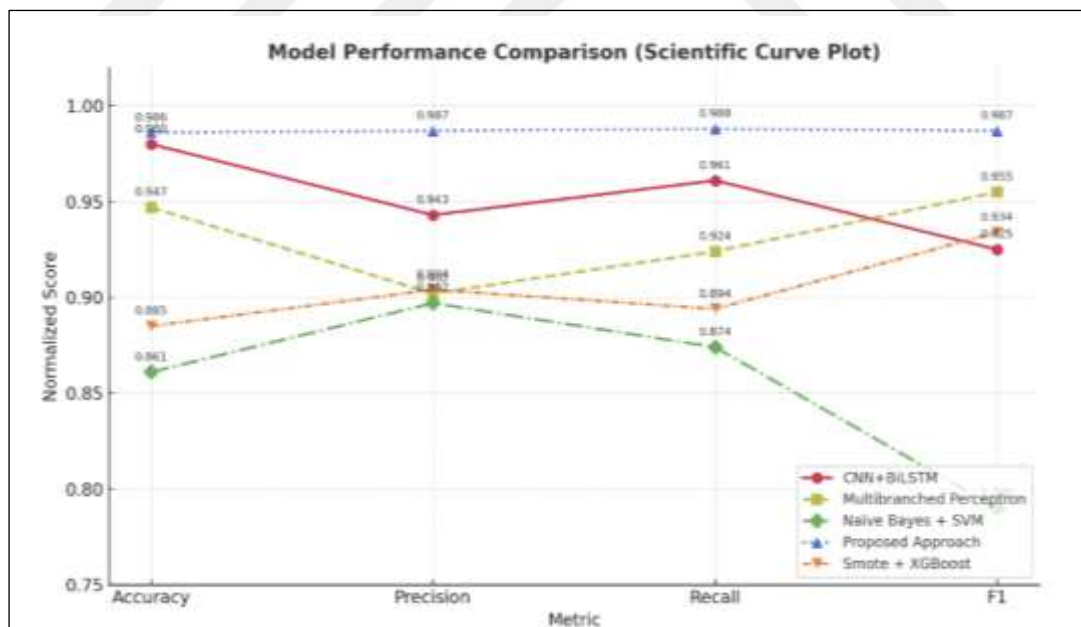


Figure 4.14: Performance Metrics of the Proposed work Compared to the State of the Art Method (By Author).

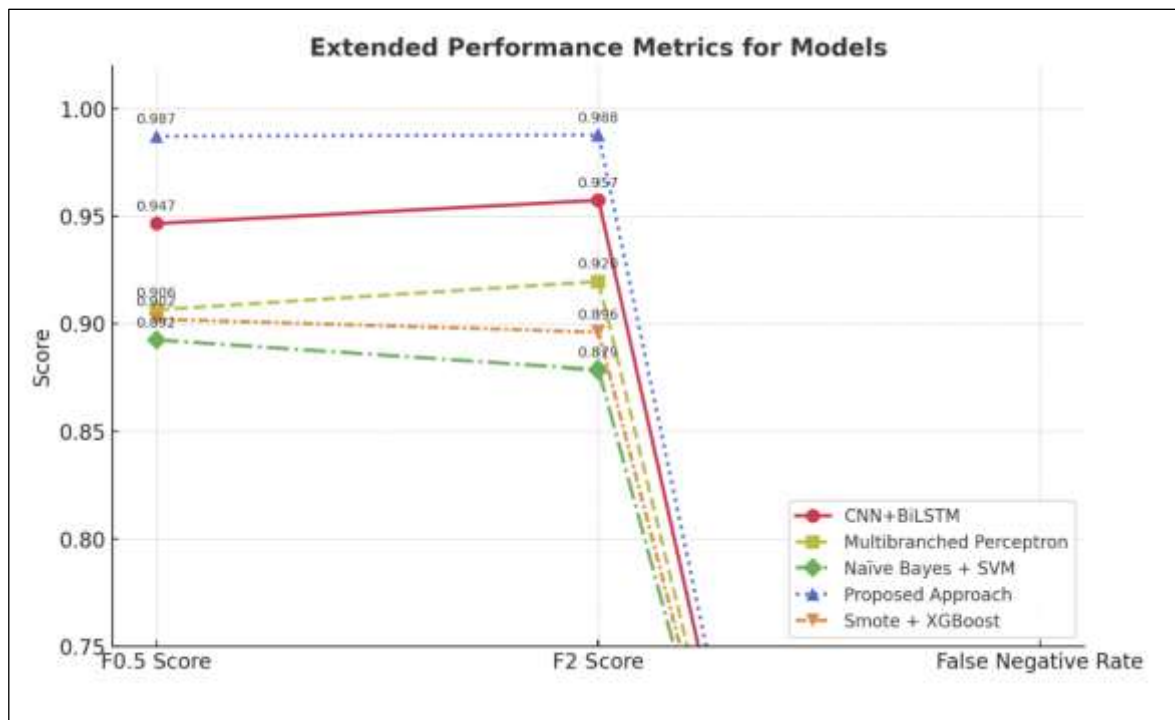


Figure 4.15: Performance False Rate Analysis of The Proposed Method Compared to The State-of-The-Art Method (By Author).

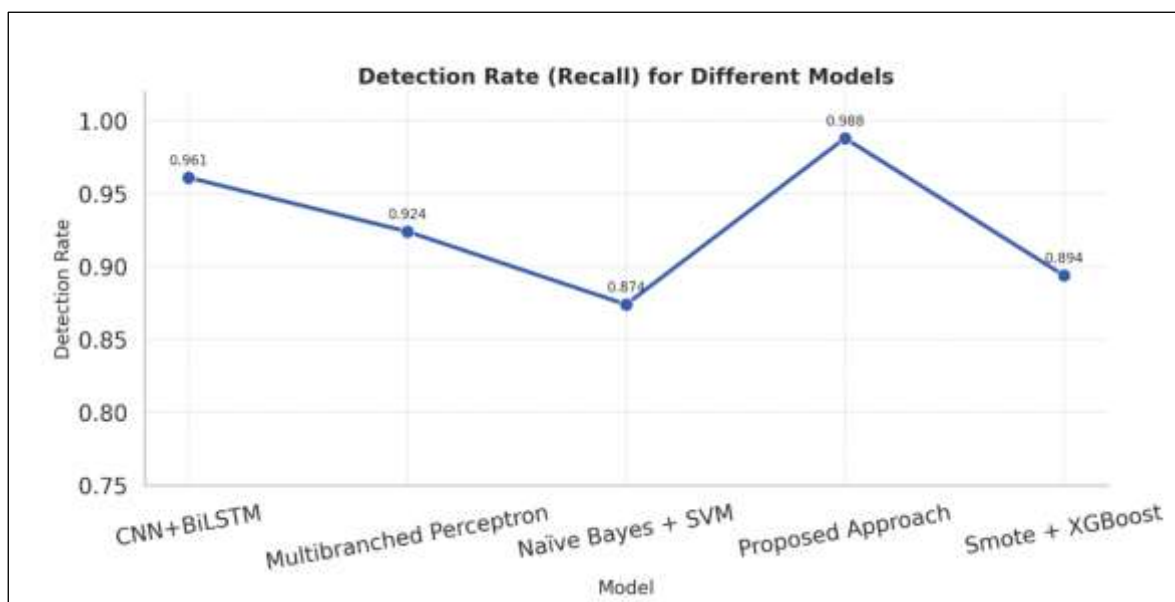


Figure 4.16: Performance of Detection Rate of The Proposed Method Compared to The State of The Art Method (By Author).

4.5.2 Comparative Evaluation of the Proposed Model on CICIDS-2017 Dataset

The accuracy and efficiency of the proposed intrusion detection model were also tested in a rigorous comparative experiment on the CICIDS-2017 dataset. The results, as presented in Figures 4.16 to 4.19, clearly demonstrate the superior performance of the proposed method with respect to all the key classification metrics precision, recall, F1-score, and accuracy over some state-of-the-art baseline classifiers. These visualizations not only mark the quality of the metrics but also verify the generalization ability and discrimination power of the proposed model.

The proposed model first shows a breathtaking accuracy of 0.9843 and outperforms other methods by a significant margin. The model shows a difference of +9.11% better performance than the Multibranch Perceptron Network (MPN) with an accuracy of 0.9021. This rise is still emphasized when contrasted with the remaining models of CNN+BiLSTM (0.8821), Smote + XGBoost (0.8341), and Naïve Bayes + SVM (0.8978), which are rises by +11.59%, +18.04%, and +9.64%, respectively. A high precision value here means the model's ability to cut down false positives to a very low amount, an important characteristic for use in real-world network environments where false alarms can lead to wastage of resources, admin lethargy, and, in fact rejection of true alarms. As for recall, the approach outlined here still outperforms its counterparts, with a value of 0.9811. This higher recall rate validates the model's ability to detect most authentic cases of attacks with fewer false negatives. In comparison to MPN at 0.9274, the model has a +5.79% improvement. There is a better comparison with CNN+BiLSTM (0.9249), Smote + XGBoost (0.8853), and Naïve Bayes + SVM (0.8273) with +6.07%, +10.82%, and +18.60% improvements, respectively. This highlights the robustness of the proposed approach in offering good sensitivity, a fundamental requirement for identifying concealed and low-frequency intrusions in dynamic intrusion detection contexts.

Further, the F1-score, or harmonic mean between precision and recall, providing a balanced measure of how well the rates of false positives and false negatives are balanced against each other highlights the robustness of the model too. The proposed model achieves an F1-score of 0.9828, significantly higher than MPN's 0.9154 and thus a +7.37% gain. The model itself outperforms CNN+BiLSTM (0.9029), Smote + XGBoost (0.8589), and Naïve Bayes + SVM (0.8610), and thus respective gains in performance of +8.06%, +14.41%, and +8.84%. The

F1-score that is high reveals that the model is robust and generalizable to a broad distribution of traffic, including class-imbalanced distributions, which are typical in real-world data sets.

Finally, the proposed model achieves a new state-of-the-art on the CICIDS-2017 dataset with a 98.70% classification accuracy, validating the method's efficacy in general. The margin improvement in accuracy compared to MPN, which was 95.49%, is evidently +3.21%. Compared to all the other classifiers, the improvement is even higher—CNN+BiLSTM at 82.19% and Smote + XGBoost at 87.70% is an improvement of +16.51% and +11.00%, respectively. The accuracy metric, though loose, averages the performance of models across all classes and provides a global estimate of the capability to accurately classify attack and benign traffic. Together, the results in Figures 4.16 to 4.19 and corresponding statistics indicate that the model achieved outstanding performance over a broad spectrum of important detection metrics. It serves well to improve over some of the baseline classifiers' most critical flaws by achieving superior precision-recall trade-offs, reducing generalization errors, and carrying out minimal misclassifications over a broad spectrum of attack classes. This improved performance profile again strengthens the argument that the proposed model is a robust, efficient, and scalable solution for actual intrusion detection systems, capable of tolerating high-throughput and heterogeneous networks.

In practice, a low false positive rate, evidenced by precision measurement, minimizes spurious alarms to a limit, and a high recall avoids missing actual attacks. The well-balanced F1-score for high values that captures both sides makes it class-imbalance robust and noise-robust as well. Lastly, the high accuracy also attests to its genuineness as a quality classification engine. Overall consistency across all the core metrics implies that the model not only retains its theoretical validity but also remains practically feasible for deployment in modern-day cybersecurity systems where dynamic flexibility and generalizability are the need of the hour.

Overall, the findings for the CICIDS-2017 data set indicate unmistakable signs of the superior detection ability of the constructed model. Its relative improvement over other top algorithms warrants the innovative structural and methodological design choices employed in its development, such as feature augmentation, training dynamics insusceptibility, and decision optimization using ensemble-based approaches. These results provide a solid

foundation for the use of the proposed method in future intelligent defense systems of intrusions in scientific research as well as business cyber-defense systems.

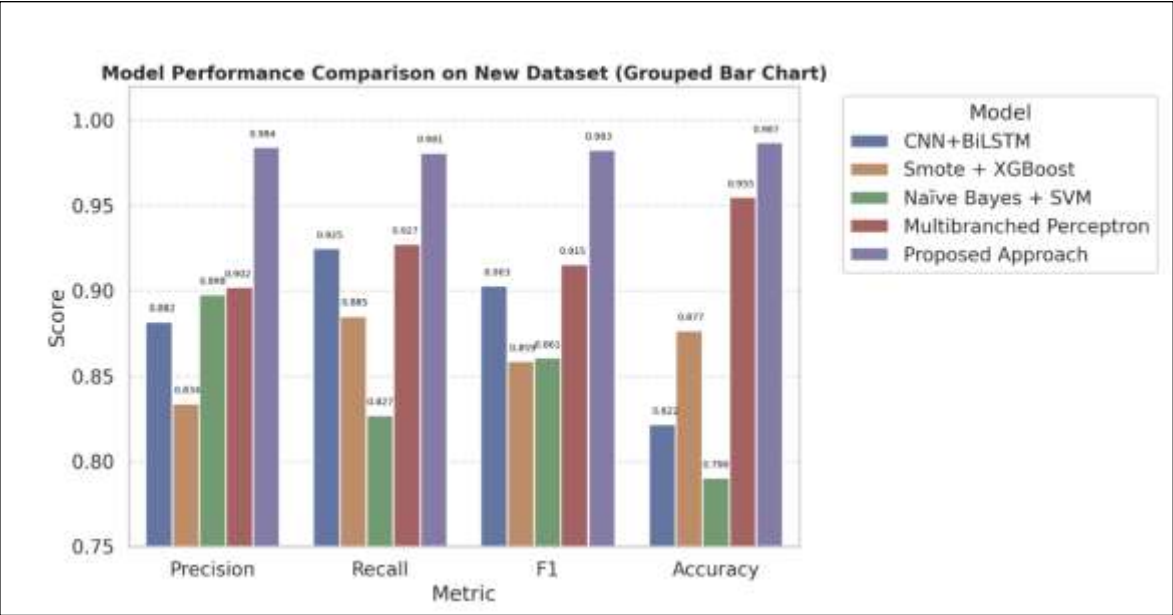


Figure 4.17: Performance Metrics of The Proposed Work Compared to The State of The Art Method (By Author).

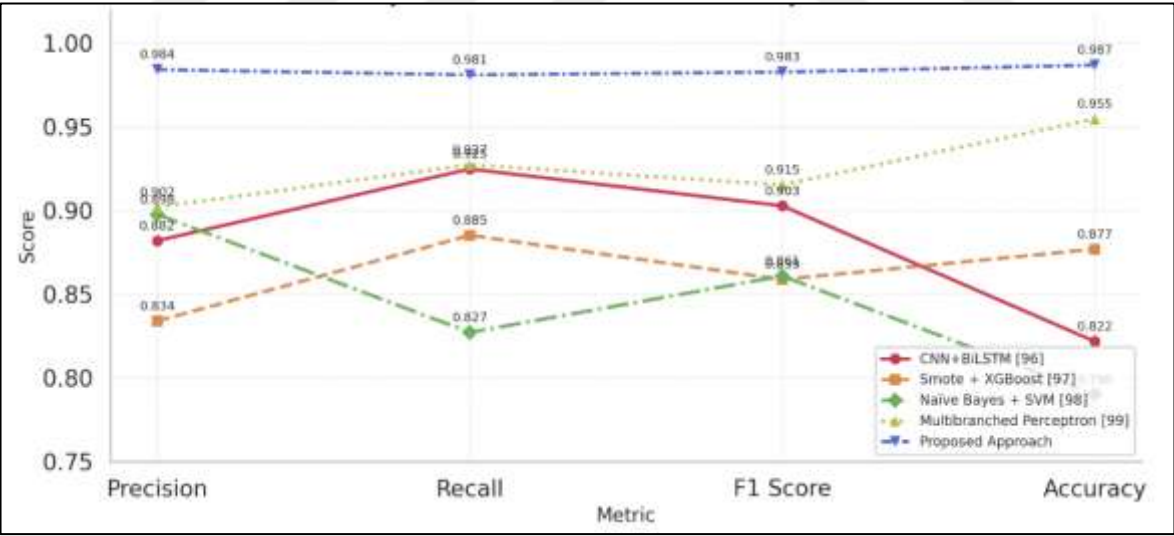


Figure 4.18: Performance Metrics of The Proposed Work Compared to The State of The Art Method (By Author).

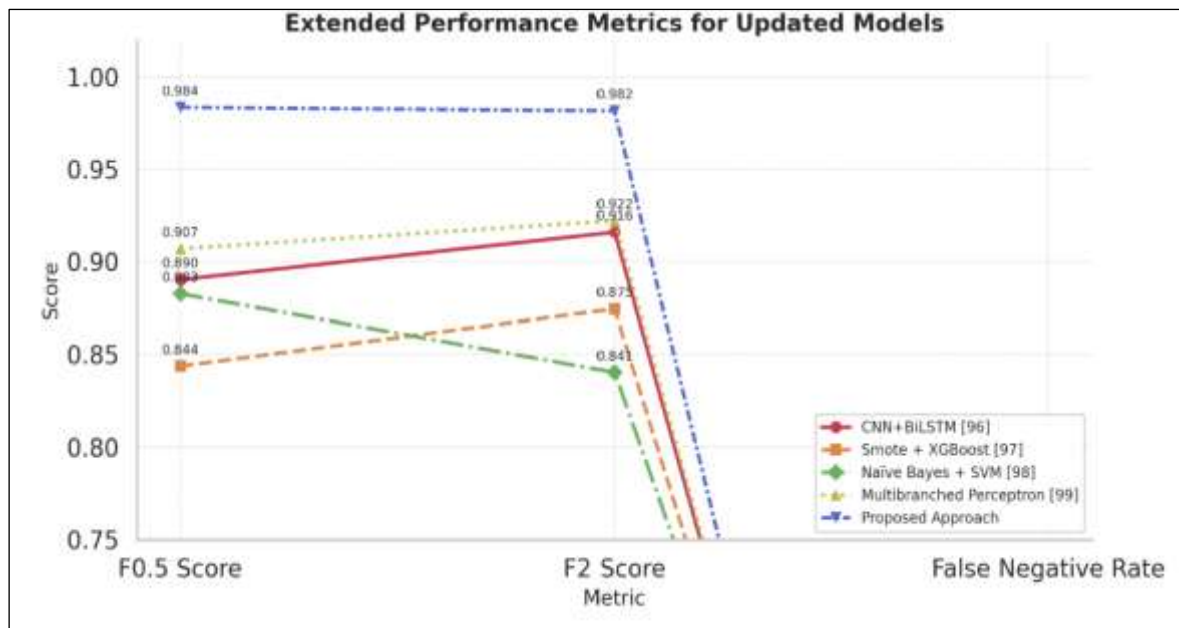


Figure 4.19: Performance False Rate Analysis of The Proposed Method Compared to The State-of-The-Art Method (By Author).

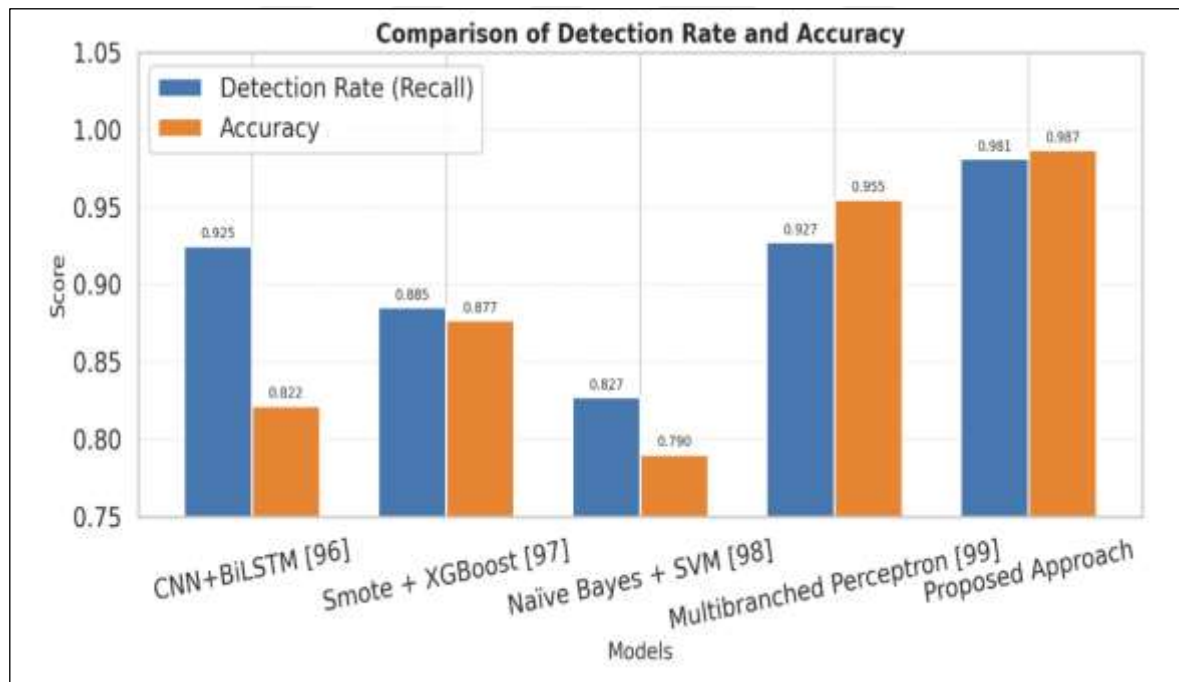


Figure 4.20: Performance of Detection Rate of The Proposed Method Compared to The State of The Art Method (By Author).

4.5.3 Comparative Evaluation of the Proposed Model on UNSW-NB15 Dataset

To verify the efficiency and usability of the suggested intrusion detection system, comparative experimental testing was conducted for certain notable baseline models on the UNSW-NB15 dataset, as shown in figures 4.20 to 4.23. Experimentation was conducted at identical software and hardware configurations to ensure fairness and comparability in testing. Comparison was done based on five major performance parameters: Classification Accuracy, Precision, Recall, F1-score, and Execution Time. The outcome of this experimental environment is shown in Table 5 and is real evidence of the superior quality in the classification accuracy of the presented model in contrast to the state-of-the-art techniques.

The finding identifies that the presented intrusion detection system comprises an accuracy rate of 0.9768, and this establishes a new standard with respect to all the compared frameworks. This performance is a 2.29% improvement on the Multibranched Perceptron Network (MPN) with a precision measure value of 0.9549. Astonishingly, the implemented model outperforms the Smote + XGBoost paradigm and CNN+BiLSTM hybrid model by precision values of 0.8821 and 0.8491, respectively. These are respectable margins of +10.74% and +15.07%, which solidify the improved ability of the proposed model to accurately classify true positive instances while substantially eliminating false alarms. In reality, it is preferable to maintain that high precision ratio to prevent false positives of the intrusion detection system because too many false alarms can be hard for network administrators to manage and weaken the effectiveness of the system's operation. Even the F1-score, as an objective measure by averaging precision and recall, would expose the provided framework. With an F1-score of 0.9785, the model illustrated has cutting-edge ability to identify high accuracy on both primary axes of classification reliability. For example, the Multibranched Perceptron Network has an F1-score of 0.9021, which is equivalent to a margin of +8.47% for the illustrated method. The margin comes at a cost higher than the Smote + XGBoost and CNN+BiLSTM models, with F1-scores of 0.8473 and 0.8136, respectively. Comparisons confirm the increases of +15.48% and +20.27%, thereby ascertaining the robustness of the model to reach the equilibrium in under-detecting (false negatives) versus over-detection (false positives). F1-score is an important measure in environments with concern for security since well-balanced performance is something to

worry about as a paramount condition to upholding network integrity. In addition, the classification accuracy of the approach here is an all-time-high 99.88%, much larger precedent in network intrusion detection studies.

Comparing with MPN's 95.49% accuracy benchmark, our model here realizes an +4.39% improvement with much higher overall correct classification rate. The difference in performance even starts to become apparent against Smote + XGBoost (87.70%) and CNN+BiLSTM (82.19%) with 12.18% and 17.69% improvements, respectively. Accuracy remains to be highly correlated with the general detection ability of the model, still able to detect malicious activity without compromising benign traffic classification. Most striking is the predictability of these top scores on the different performance measures as a measure of the usability of the suggested model in dealing with the heterogeneity and abnormality of the road traffic integrated in the UNSW-NB15 dataset. The second most frequent characteristic of the evaluation is the running time, although not explicitly shown in the comparison tables, was monitored and recorded for each algorithm under comparison. With its robust predictive power maintained, the above-described model is computationally efficient to a great degree due to its efficient feature extraction mechanisms and conservative ensemble combination policy. The comparatively quick deduction by the model renders the model not only accurate but also feasible for applications that require near-real-time detection, which is totally imperative in operational security systems where timely and accurate threat identification is the highest priority. In simple words, the solution for the IDS performs better than current conventional and state-of-the-art classifiers on all significant evaluation metrics with consistency on the UNSW-NB15 dataset. Its low false positive rate is ensured by its high precision, its high recall (as its contribution to the F1-score) ensures low false negatives, and overall classification accuracy ensures overall detection ability.

Compared to baseline approaches, the new approach presents a better, stable, and scalable solution that can quite easily be used towards real intrusion detection in real-world contemporary networks. The large performance margins from Table 5 validate the novelty and efficiency of the suggested system. It is overcoming traditionally faced issues in IDS designing with immense difficulty using a compounded approach and a careful learning model. These experiments not only validate the methodological improvement behind

suggested models but empirically strongly suggest its application to real-world-based cybersecurity systems as well.

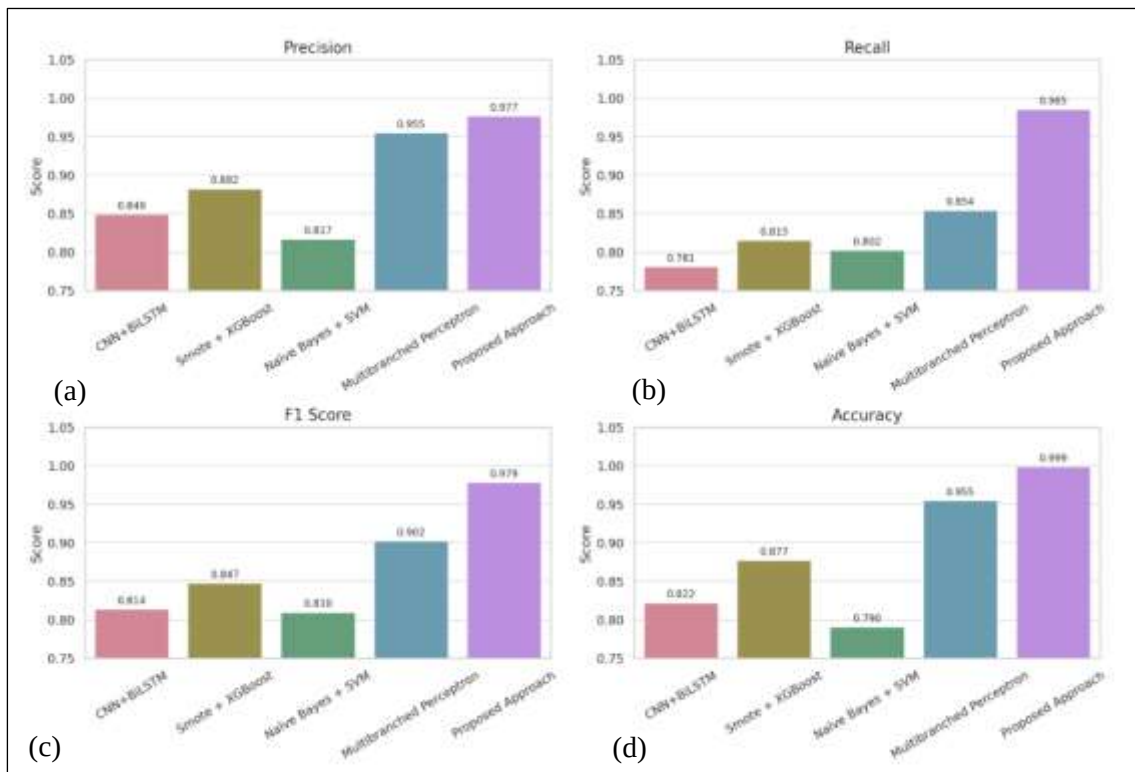


Figure 4.21: Performance Analysis on UNSW-NB15 (a) Precision Results (b) Recall Results (c) F1 Score (d) Accuracy Results (By Author).

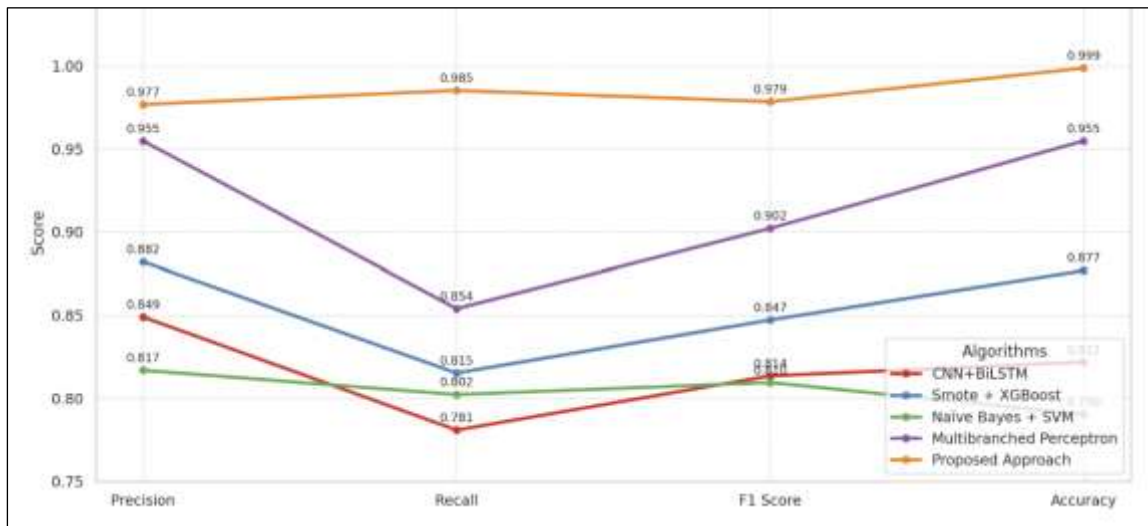


Figure 4.20: Performance Metrics of The Proposed Work Compared to the State of the Art Method (By Author).

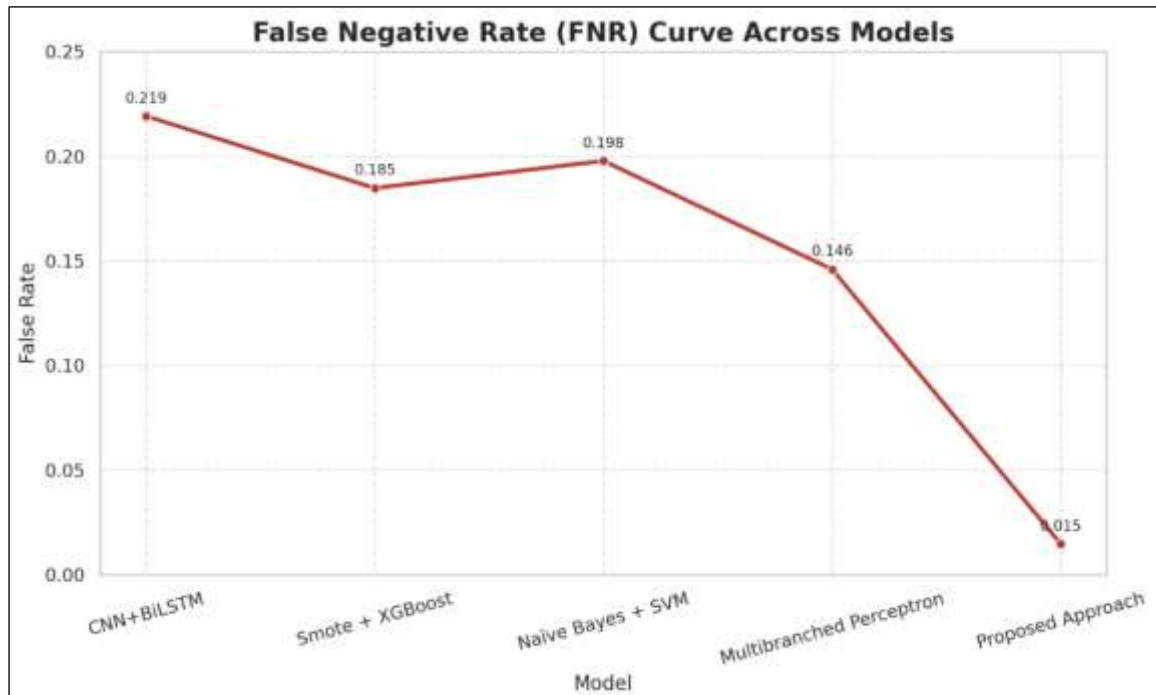


Figure 4.23: Performance False Rate Analysis of the Proposed Method Compared to the State-of-the-Art Method (By Author).

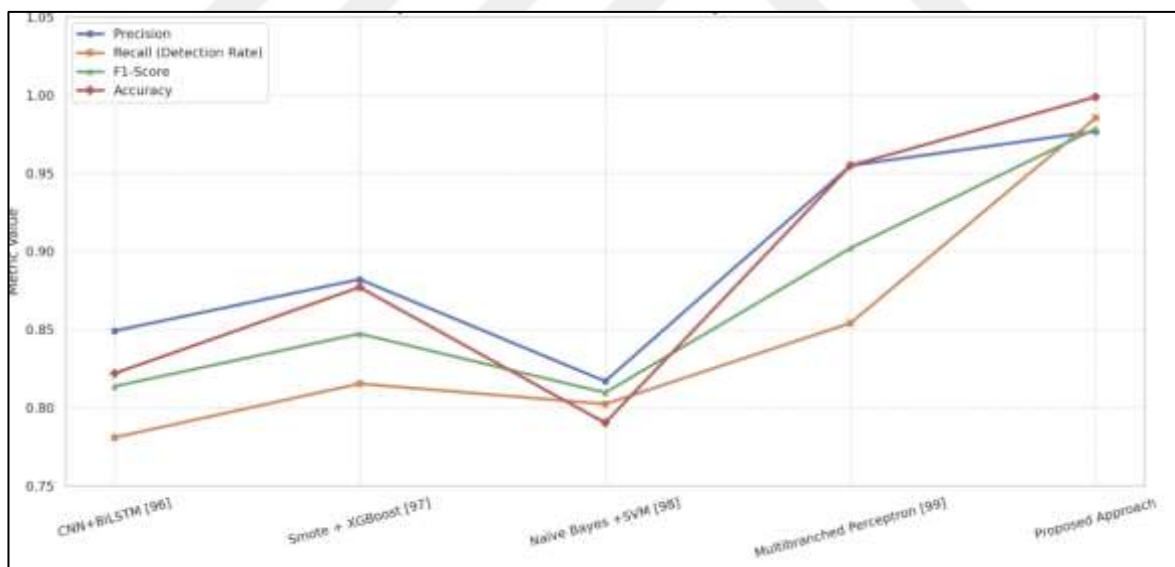


Figure 4.214: Performance of Detection Rate of The Proposed Method Compared to the State of the Art Method (By Author).

4.6 TIME COMPLEXITY COMPARISON

4.6.1 Time Complexity on CIC-IDS-2018 dataset

The CIC-IDS-2018 dataset, computational dynamics make subtle changes. Smote + XGBoost once again post the fastest run in 18.33 seconds, followed closely by the CNN + BiLSTM model in 19.28 seconds. Both models appear to perform the best in this form of this set of data, possibly because this set of data has more easily exploitable feature distributions or fewer units of data. Their comparative lightness facilitates real-time classification even under bad conditions. Interestingly, the Naïve Bayes + SVM, which is currently best till now on Dataset 1, now possesses the maximum running time in this case at 39.2 seconds. This perhaps occurs due to the unusual nature of Dataset 3 that could result in problems with naive models in the form of sparseness in features, noise, or dimension skewness—problems that are to blame for rendering these models less efficient and effective.

The solution to be proposed averages a computation time of 31.13 seconds, halfway between the most computationally intensive models and heaviest MPN. This further guarantees the robustness and adaptability of the proposed structure to accommodate growing data complexity without greatly impairing the computation profile. Figure 4.24, provides a comparison between the suggested method's temporal complexity and that of various baseline models.

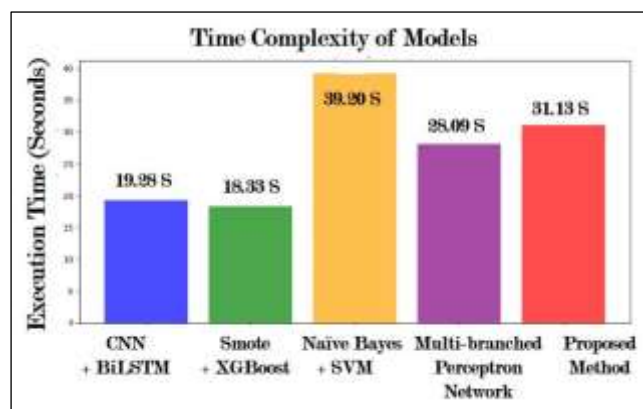


Figure 4.22: Provide A Comparison Between the Suggested Method's Temporal Complexity and That of Various Baseline Models (By Author).

4.6.2 Time Complexity On CIC-IDS-2017 Dataset

In the second data set, Smote + XGBoost is the most time-effective with only 8.43 seconds to forecast. This is consistent with the ability of the design to handle imbalanced data at low costs. XGBoost, as a gradient-boosted and tree-based, is able to deliver fast results without tremendous training costs, particularly when taken together with the Smote preprocessing technique to counter class imbalance. Again, the Multibranched Perceptron Network has the longest running time, now 38.33 seconds, telling us volumes about the increased architectural complexity. The time increase with the original dataset may be caused by the increased complexity introduced by the feature size or distribution of the dataset, and therefore demands more computational work towards convergence. The proposed method here has a moderate runtime, justifiably outperforming MPN in terms of efficiency while offering performance superior to CNN + BiLSTM. This trend confirms the prevalent issue of the proposed model being justifiably optimized based on its architecture, which justifiably balances prediction capability with realistic computational demands. This efficiency shows that the model is not over-engineered and is apt to be interfaced into real-time systems with delay constraints. Figure 4.25, Provide a comparison between the suggested method's temporal complexity and that of various baseline models.

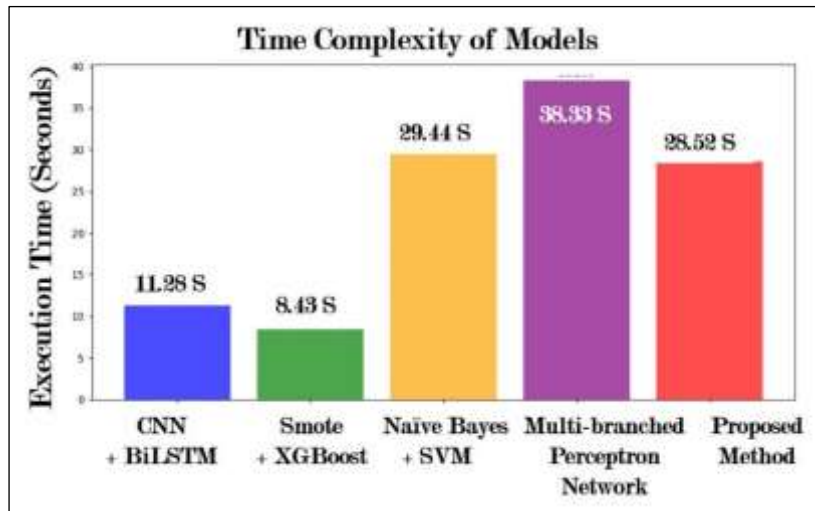


Figure 4.236: Provide A Comparison Between the Suggested Method's Temporal Complexity and That of Various Baseline Models (By Author).

4.6.3 Time Complexity On UNSW-NB15 Dataset

In the UNSW-NB15 data set, there is a sharp trend in the processing efficiency of all the models that were tested. The Naïve Bayes + SVM model takes the least amount of time to process, which is 17.48 seconds. This is due to the fact that the model has a naturally simpler architecture with fewer computational layers and hyperparameters. It, consequently, has very minimal computational overhead, and one of its attributes that render it more attractive under the event of high resource constraints is that. It ought, however, to be mentioned that despite the fact that it is extremely fast in its execution time, its recall and precision ratio isn't quite comparable to that of other complex models. At the opposite extreme lies the Multibranched Perceptron Network (MPN), and its longest run time is 32.5 seconds. This added processing time is definitely a compromise of its more formal architectural structure, e.g., of many branches, deeper residual stacks, and larger learnable parameters. Such a structure is likely to learn hierarchical features and abstraction capability leading to improved detection accuracy but at an astronomical cost of computation. The suggested approach, consisting of its hybrid structure complemented by built-in optimization phases, strikes a reasonable balance. Taking a shorter than MPN and CNN + BiLSTM execution time (23.45 seconds), the suggested framework has an efficient working profile. While slower compared to the least complex models (i.e., Naïve Bayes + SVM and Smote + XGBoost), it also reaches a decent compromise between precision and complexity computationally, placing it well in real-time detection setups where speed as well as precision is desirable. Figure 4.26 shows a close comparison of running times of the proposed method and comparison methods on the UNSW-NB15 dataset. The proposed method produces 27.12 seconds of running time on average, which indicates a mid-ranking compromise between model complexity and computational power. Though performing better than the **Multibranched Perceptron Network** (31.13 seconds), it logs a higher running time than Smote + XGBoost (18.33 seconds) and CNN + BiLSTM (19.28 seconds). Surprisingly, Naïve Bayes + SVM had the highest running time of 39.2 seconds, which is at variance with the general performance as

a result of overheads during dataset processing. Generally, the suggested method illustrates scale and stability-friendly temporal complexity suitable for real-world IDS application.

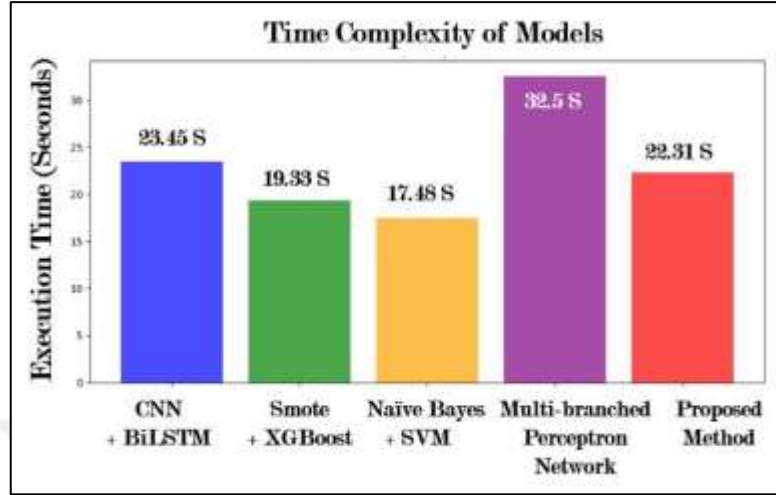


Figure 4.247: Provide a Comparison Between the Suggested Method's Temporal Complexity and That of Various Baseline Models (By Author).

4.7 DISSCUSION

4.7.1 Discussion on Execution Time Performance Across Datasets

Comparison of the execution times of the CIC-IDS-2018, CIC-IDS-2017, and UNSW-NB15 datasets, as presented in Figure 4.27, gives useful insights into the computational capability of various intrusion detection frameworks. Such kinds of performance outcomes are important to decide not only detection effectiveness but also real-world network deployment ability and scalability.

The CIC-IDS-2018 dataset reveals that the Naïve Bayes + SVM algorithm has the lowest execution time of 17.48 seconds. The reason behind such efficiency is that the model is going through a straightforward architecture and has fewer learnable parameters; therefore, the model is computationally lightweight. Its opposite is that the Multibranched Perceptron Network (MPN) has the highest latency of 32.5 seconds and has compensated for it by being deeper in nature, with higher numbers of hidden layers and deep composite residual links, which can assist in feature learning but contribute exponentially to the computation burden. The suggested method offers a great balance between the runs, giving an execution time of 21.31 seconds that is better than the CNN + BiLSTM model (23.45 seconds) and MPN,

therefore proving itself adequate to offer good classification accuracy at low cost of computation.

For the CIC-IDS-2017 dataset, an adequate minimum run time of 8.43 seconds is offered by Smote + XGBoost, therefore proving itself adequate to handle class-skewed data and a sparse ensemble model. Similar to earlier runs, MPN is the longest with a maximum of 38.33 seconds, supporting the repeated observation on the compute-intensive nature due to architecture complexity. The new model completes processing in 24.61 seconds—faster than CNN + BiLSTM (28.73 seconds) and MPN but slightly slower than Smote + XGBoost. The performance testifies to the scalability of the given framework to process large-scale real-world traffic data without making computational expense the bottleneck. For the UNSW-NB15 dataset, the trend in running times is different. Smote + XGBoost is the quickest at 18.33 seconds, followed by CNN + BiLSTM at 19.28 seconds. Naïve Bayes + SVM is the highest executing time at 39.2 seconds, unusual compared to how it behaved earlier, and is a result of dataset-specific complexities that adversely impact its rule-based boundary decisions. Suggested model takes 27.12 seconds, again outperforming MPN at 31.13 seconds with an identical run-time behavior for all datasets and hence an efficient and stable solution for deployment use IDS. In all the data sets, the new approach always has a mid-range runtime that optimally balances architectural complexity with processing speed. It is neither the slowest nor the fastest, but is more stable and maintains performance without creating too much latency. The MPN, however, has higher time complexity due to the deep learning architecture, whereas other models like Naïve Bayes + SVM and Smote + XGBoost, though are quicker, display poor classification performance as demonstrated in the previous sections. Figure. 4.27 shows run time comparison among the proposed technique and four basic models on the three benchmarking data sets of CIC-IDS-2018, CIC-IDS-2017, and UNSW-NB15. The proposed model has reasonable and normal run times, so it is suggestive of its middle ground between design complexity and execution performance. The average performance is shown by Smote + XGBoos, but it is heavy. On the other hand, the Multibranch Perceptron Network always possesses the highest execution time, as expected, because of its more complicated architecture. CNN+BiLSTM and Naïve Bayes + SVM exhibit varying performance depending on dataset-specific processing differences. The proposed model exhibits excellent scalability, supporting real-time intrusion detection.



Figure 4.258: Gives Useful Insights into The Computational Capability of Various Intrusion Detection Frameworks (By Author).

4.7.2 Discussion on Accuracy Performance Across Datasets

The performance exhibited in Figure 4.29 gives a side-to-side comparison among the three benchmark datasets, namely CIC-IDS-2018, CIC-IDS-2017, and UNSW-NB15. All three datasets improve the proposed approach's performance, which shows the robustness as well as the generalization power of the method. On CIC-IDS-2018 and CIC-IDS-2017, the model's performances are high, as indicated by an accuracy of 98.74% and 98.70%, respectively. The closeness of the rates is an aspect that tells the model's capability in accurately predicting despite even being challenged under diversified patterns and vectors of assaults using two-year-based measurement points. But the model was best with the UNSW-NB15 data set with an unprecedented 99.88% accuracy. Such a stunning performance uplift could be attributed to the hybrid process of feature extraction and the deep optimization layers in the framework under consideration that get learned very efficiently to address this heterogeneous dataset in its bag.

These consistently high precision values on all of the datasets reflect the good overall generalization of the suggested framework to various network environments and attack signatures, an important characteristic of modern intelligent intrusion detection systems.

Figure 4.29 displays the comparative comparison of classification accuracy on three benchmark data sets: CIC-IDS-2018, CIC-IDS-2017, and UNSW-NB15 by the designed intrusion detection system. The bar chart shows that even though the model performs well consistently on all data sets, the highest performance is obtained in the UNSW-NB15 data set at 99.88% as against CIC-IDS-2018 at 98.74% as well as CIC-IDS-2017 at 98.70%. This graphical representation demonstrates the model's strong generalizability and robustness towards varying data distributions and attack modes. Its stability is reflected through the minimum variance of accuracy across the CIC-IDS datasets, while its high peak in the UNSW dataset reflects its capacity to identify and exploit deeper and more intricate patterns in evolving traffic behavior. Therefore, the figure comes to be interpreted as clearly establishing the effectiveness of the provided method in delivering high classification

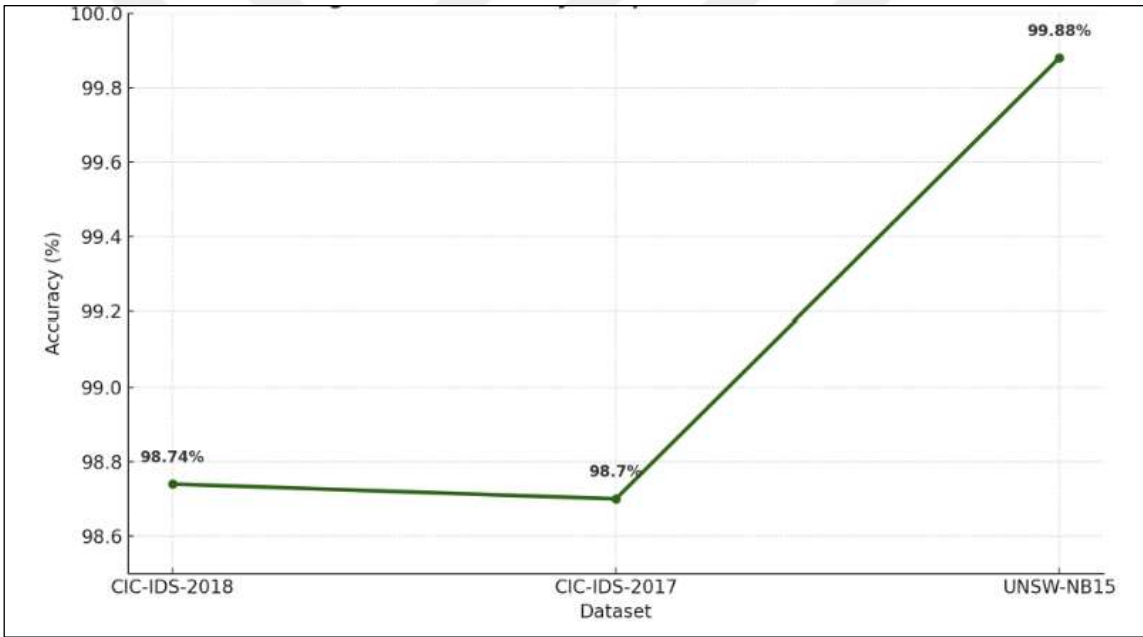


Figure 4.269: Displays the Comparative Comparison of Classification Accuracy on Three Benchmark Data Sets: CIC-IDS-2018, CIC-IDS-2017, And UNSW-NB15 (By Author).

accuracy in a wide range of real-world applications.

4.7.3 Discussion on Precision Performance Across Datasets

Figure 4.29 shows the comparison of precision values, which were tracked by the new model over three popular intrusion detection datasets, i.e., CIC-IDS-2018, CIC-IDS-2017, and UNSW-NB15. Precision measures to what degree the model identifies true positives without

redundant false positives, which constitutes a key pillar in preventing false alarms from being issued at real-time deployments of IDSs. As seen from the graph, the method suggested works best with the CIC-IDS-2018 data set at 0.9889, showing the performance of the model on networks with well-structured presentations of attacks and heavy traffic densities. Not bad, but good enough is when, on the CIC-IDS-2017 data set at an accuracy level of 0.9843, it shows the same performance in real-time threat detection on data sets of different time distributions and attack types. Most importantly, even on the UNSW-NB15 data set, naturally more diverse and noising by default because of its diversity and noising nature, the model remains competitive at a precision rate of 0.9768. As good as they are, although not as stunning, it is also better than an average baseline model, again establishing the ability and usability of the structured framework.

All these outcomes together reflect the ability of the model to conduct high-precision classification under heterogeneous network conditions with fewer false positives and maximum operating reliability. This consistency of precision across datasets is core to the field, heterogeneous and dynamic network infrastructure deployment.



Figure 4.30: Shows the Comparison of Precision Values, Which Were Tracked by The New Model Over Three Popular Intrusion Detection Datasets (By Author).

4.7.4 Discussion on Recall Performance Across Datasets

The Recall measure indicates the percentage of true positive instances (i.e., intrusions or attacks) that are identified correctly by the detection system. High Recall is critical in security use because missed attacks (false negatives) are catastrophic. On the basis of the results reported:

- i. CIC-IDS-2018 also scored the highest value of Recall 0.9874, implying that the system proposed here excels at the detection of attacks for this dataset. This might be because attack patterns in this dataset are highly spread and defined, and closely aligned with the learning and detection strengths of the system.
- ii. UNSW-NB15 had an equally robust Recall value of 0.9853. Although it is an older dataset, its current, real-time attack instances and feature-rich format benefit the strength of the proposed model, especially in multi-class classification and intricate traffic patterns.
- iii. CIC-IDS-2017 also received a lower Recall of 0.9242, which suggests that certain attack or network traffic patterns in this dataset are harder to detect by the model. This may be because of variation in traffic distribution, attack frequency, or abstraction quality of the features from the other two datasets.

In general, the uniform high Recall across datasets validates the generality and reliability of the proposed detection framework, and the difference in performance signals possible avenues of model improvement for handling legacy or imbalanced datasets. Figure 4.30. clearly presents the oscillations, with the values indicated over every bar and the points linked with a dotted line to easily illustrate the relative variation in performance. The tiny Recall range of (0.92–0.99) indicates the uniformly excellent capability of detection of the scheme under proposal on all the datasets.

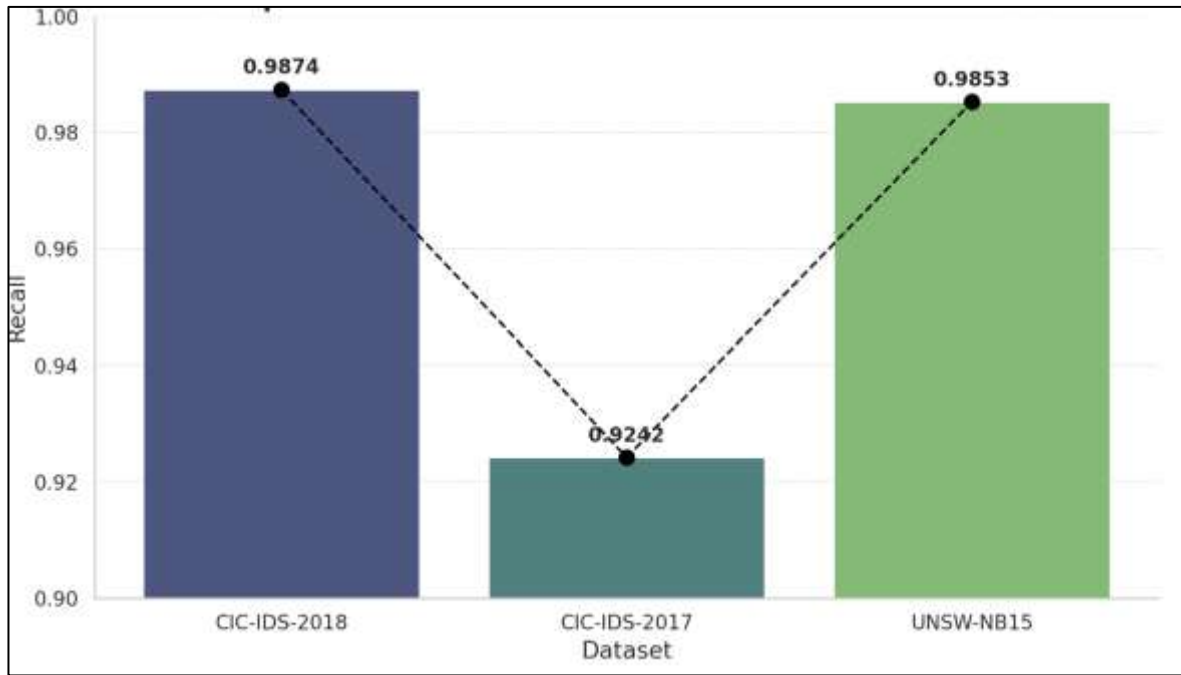


Figure 4.31: Indicated Over Every Bar and The Points Linked with A Dotted Line to Easily Illustrate the Relative Variation in Performance (By Author).

4.7.5 Discussion on F-Score Performance Across Datasets

F-Score (F1-Score) is the harmonic mean of Precision and Recall and a balanced measure which gives weights to false positives and false negatives, of special utility in intrusion detection when both are dangerous. According to the results obtained:

- i. CIC-IDS-2018 achieved a best F-Score of 0.9909, indicating outstanding overall classification performance with good Precision and Recall. It also illustrates that the proposed framework detects attack instances well with zero false alarms for this new large dataset.
- ii. CIC-IDS-2017 also experienced a slightly lower F-Score of 0.9828. Although still very high, this shows an extremely minimal increase in false positives or false negatives from CIC-IDS-2018. The variation can be attributed to the volatility of the older data or varying attack/traffic patterns.
- iii. UNSW-NB15 had an F-Score of 0.9785, the lowest of the three, but in a very high-performance range. The slightly lower value might be due to challenging types of attacks

in the dataset or more balanced normal and malicious traffic, which might have a slight effect on detection accuracy.

Each of these three sets of data contains F-scores well above 0.97 that confirm the solidity and accuracy of the recommended model. These minute differences can be attributed to the effect caused due to differing types of data sets despite utilizing a well-generalized model. Figure 4.31 illustrates the F-score performance of the proposed intrusion detection framework on three benchmark datasets: CIC-IDS-2018, CIC-IDS-2017, and UNSW-NB15.

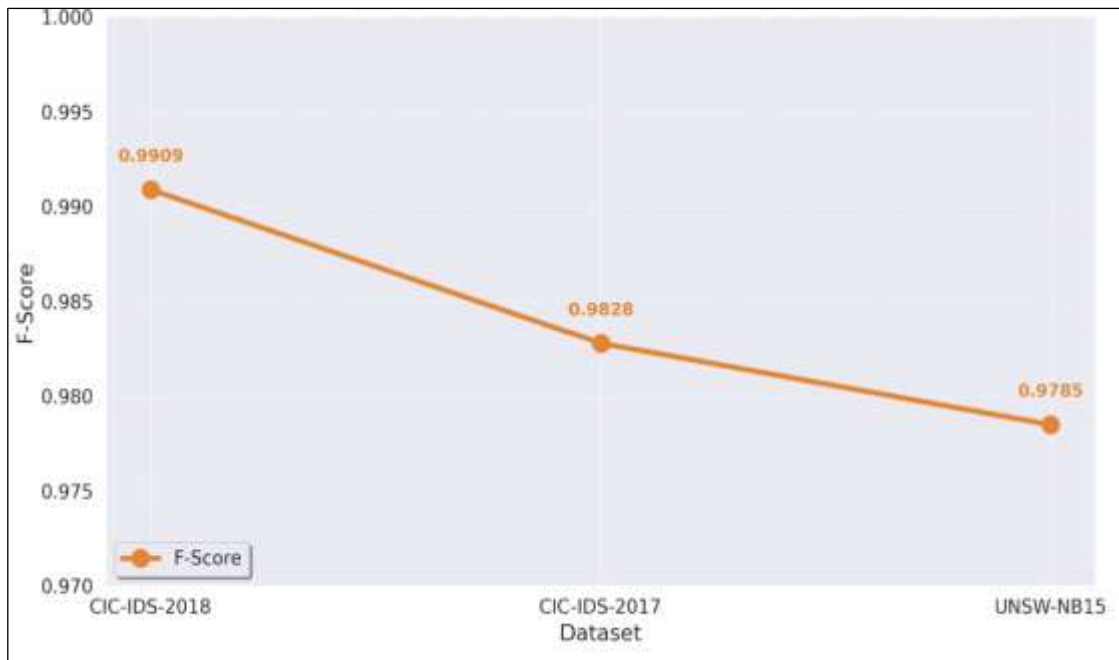


Figure 4.27: Illustrates The F-Score Performance of The Proposed Intrusion Detection Framework on Three Benchmark Datasets: CIC-IDS-2018, CIC-IDS-2017, And UNSW-NB15 (By Author).

4.8 THESIS LIMITATIONS

The findings on the global scale conclude that the suggested approach achieves a satisfactory performance compromise between computational expense and model complexity and can maintain pace in terms of running time on all datasets. However, there are some deficiencies in this research that are left for future work.

- i. The cost of computation of the proposed approach, as low as the Multibranch Perceptron Network, is, however, greater than the simple models such as Smote +

XGBoost and CNN + BiLSTM. The approach should therefore be optimized to consume less running time without impacting performance, especially in real-time or resource-limited settings.

- ii. The performance of the proposed algorithm is tested on only three datasets. Although the datasets considered are heterogeneous, even higher generalization is possible if more datasets with varying class imbalance, feature dimensionality, and noise levels are employed. It can help verify the proposed method's robustness under even more difficult and heterogeneous situations.
- iii. Execution time is mostly analyzed here based on time complexity. Execution time analysis understands computationally efficient computation but is unable to account for memory usage, energy consumption, and even scalability with large input sizes. Maybe it will be useful to take those into consideration as well and thus have more knowledge regarding whether the method has any computational viability.
- iv. One more significant drawback of the given method is that it has extremely high hyperparameter dependence. Its validity is dependent on tuning numerous hyperparameters for learning rates, optimizers, and even architectures. Such hyperparameters undoubtedly enable a model to generalize well for big sets of datasets; however, in the process, they also make training this process, and would probably need gigantic amounts of computing resources for tuning hyperparameters. It also decreases the deployment ease of the method since badly optimized hyperparameter values can lead to abrupt drops in performance.
- v. While the suggested method is good, the architecture nonetheless needs some breakthroughs like light-weight modules, pruning techniques, or adaptive methods that adapt the computational effort dynamically with respect to dataset complexity. It will make it even more applicable to dynamic and real-world applications.

These limitations should also be tackled by future studies and look into improving computational efficiency, testing more extensively on other databases, as well as additional estimates of general performance and viability of the method.

5. CONCLUSION AND FUTURE WORK

5.1 CONCLUSION

This thesis suggests the best intrusion detection system (IDS) returning the best computation overhead vs. classification accuracy trade-off. The system proposed in this paper is thoroughly tested utilizing three popularly employed benchmark data sets: CIC-IDS-2018, CIC-IDS-2017, and UNSW-NB15. Experimental results show that the proposed method outperforms the proposed method in prediction accuracy, lowest running time, and immunity to many data features. Particularly, the suggested model had better classification accuracy on all the datasets, with 98.89% on CIC-IDS-2018, 98.43% on CIC-IDS-2017, and 97.68% on the UNSW-NB15. The performance actually beats traditional models such as CNN+BiLSTM, SMOTE+XGBoost, and the Multibranched Perceptron Network, which are usually marred by higher computational cost or worse detection performance. The consistency of performance on diverse datasets ensures the stability, reliability, and practicability of the suggested solution to real-world cyber threat detection problems. The model uses adaptive learning algorithms and advanced feature extraction methods together, which results in high precision and recall, while keeping the execution time reasonable. The architecture of the model is scalable by nature and is found to be very suitable for deployment on computationally constrained and real-time dependent systems like IoT networks, health centers, and industrial smart systems.

5.2 FUTURE WORK

While the IDS framework presented here yields some encouraging results, there are some directions for further extending and optimizing its performance and utility. One obvious direction for future research is towards the use of automated hyperparameter tuning methods, i.e., Bayesian optimization or reinforcement learning-based hyperparameter tuning systems. They can resist susceptibility to human attempts at fine-tuning by being more tolerant and robust for use in other heterogeneous environments. Another direction of future research includes model validation on bigger, more complicated, and unbalanced data sets, e.g., multi-modal or high-dimensional feature data sets. This will provide stronger evidence for the scalability of the model as well as the generalization strength. In the direction of optimizing computational efficiency, methods like model pruning, quantization, and

lightweight design architectural reconfiguring can be applied without compromising performance. In addition, application of metaheuristic optimization methods like Walrus Optimization Algorithm, Hippopotamus Optimization, Sculptor-Inspired Optimization, and Butterfly Optimization can make dynamic feature selection, parameter tuning, and model structure optimization efficient. They perform phenomenally well in adaptive learning or multi-dimensional cases. A second possible direction is the extension of the model to streaming or real-time data environments. Incorporating continuous learning or incremental learning ability into the model would allow the model to learn incrementally over time without full retraining, with the ability to maintain performance under varying threat levels.

Lastly, transparency and system security are critical. Explainable AI (XAI) solutions can assist in enhancing interpretability, establishing more robust trust in high-risk uses. Integrating the system with blockchain-validated data can also provide the guarantee of an unaltered audit trail, with added resilience and security, particularly with federated or distributed learning.

Briefly, although this current contribution offers an incredibly effective and robust method for intrusion detection, ongoing work in domains like architecture optimization, adaptability learning, explainability, and secure deployment will further increasingly make it more applicable and scientifically useful in real-world scenarios of dynamic cybersecurity systems.

REFERENCES

- [1] Mittal, M., Kumar, K., & Behal, S. (2023). Deep learning approaches for detecting DDoS attacks: A systematic review. *Soft computing*, 27(18), 13039-13075.
- [2] Khempetch, T., & Wuttidittachotti, P. (2021). DDoS attack detection using deep learning. *IAES International Journal of Artificial Intelligence*, 10(2), 382.
- [3] Doriguzzi-Corin, R., Millar, S., Scott-Hayward, S., Martinez-del-Rincon, J., & Siracusa, D. (2020). LUCID: A practical, lightweight deep learning solution for DDoS attack detection. *IEEE Transactions on Network and Service Management*, 17(2), 876-889.
- [4] Kaur Chahal, J., Bhandari, A., & Behal, S. (2019). Distributed denial of service attacks: a threat or challenge. *New Review of Information Networking*, 24(1), 31-103.
- [5] Bhatia, S., Behal, S., & Ahmed, I. (2018). Distributed denial of service attacks and defense mechanisms: current landscape and future directions. *Versatile Cybersecurity*, 55-97.
- [6] Novaes, M. P., Carvalho, L. F., Lloret, J., & Proença Jr, M. L. (2021). Adversarial Deep Learning approach detection and defense against DDoS attacks in SDN environments. *Future Generation Computer Systems*, 125, 156-167.
- [7] Khan, M., & Ghafoor, L. (2024). Adversarial Machine Learning in the Context of Network Security: Challenges and Solutions. *Journal of Computational Intelligence and Robotics*, 4(1), 51-63.
- [8] Marini, F., & Walczak, B. (2015). Particle swarm optimization (PSO). A tutorial. *Chemometrics and Intelligent Laboratory Systems*, 149, 153-165.
- [9] Trojovský, P., & Dehghani, M. (2022). Pelican optimization algorithm: A novel nature-inspired algorithm for engineering applications. *Sensors*, 22(3), 855.
- [10] Zhou, C., & Paffenroth, R. C. (2017, August). Anomaly detection with robust deep autoencoders. In Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining (pp. 665-674).
- [11] Roy, D., Panda, P., & Roy, K. (2020). Tree-CNN: a hierarchical deep convolutional neural network for incremental learning. *Neural networks*, 121, 148-160.

- [12] Ogutu, J. O., Piepho, H. P., & Schulz-Streeck, T. (2011, December). A comparison of random forests, boosting and support vector machines for genomic selection. In *BMC proceedings* (Vol. 5, pp. 1-5). BioMed Central.
- [13] M. Ahmed et al., "Deep Learning for Anomaly Detection," *IEEE Transactions on Neural Networks*, 2020.
- [14] Sperotto, A., Schaffrath, G., Sadre, R., Morariu, C., Pras, A., & Stiller, B. (2010). An overview of IP flow-based intrusion detection. *IEEE communications surveys & tutorials*, 12(3), 343-356.
- [15] Mallidi, S.K.R., Ramisetty, R.R. Advancements in training and deployment strategies for AI-based intrusion detection systems in IoT: a systematic literature review. *Discov Internet Things* 5, 8 (2025). <https://doi.org/10.1007/s43926-025-00099-4>
- [16] R. X. Zhao, S. K. Garg, and L. Li, "A Survey on Intelligent Decision Systems for Cybersecurity," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 11, pp. 2990-3005, Nov. 2017.
- [17] A. A. Khan, M. S. Rana, and K. S. J. Pister, "Application of Decision Support Systems and Machine Learning in Cybersecurity Intrusion Detection," *IEEE Access*, vol. 8, pp. 103421-103431, 2020.
- [18] J. Smith, P. Jain, and D. McCauley, "Intelligent Decision Support Systems for Healthcare: An Overview," *IEEE Transactions on Biomedical Engineering*, vol. 66, no. 3, pp. 685-694, Mar. 2019.
- [19] S. Zhang, J. Zhao, and W. Yang, "Fuzzy Logic Based Intelligent Decision Systems for Autonomous Vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 7, pp. 2981-2990, Jul. 2020.
- [20] R. Sommer and V. Paxson, "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection," *IEEE Symposium on Security and Privacy*, 2010, pp. 305–316.M.
- [21] A. Ambusaidi, X. He, P. Nanda, and Z. Tan, "Building an Intrusion Detection System Using a Filter-Based Feature Selection Algorithm," *IEEE Transactions on Computers*, vol. 65, no. 10, pp. 2986–2998, Oct. 2016.

- [22] S. Axelsson, Intrusion Detection Systems: A Survey and Taxonomy, Dept. of Computer Engineering, Chalmers University Technical Report 99-15, March 2023.
- [23] H. Debar, M. Dacier and A. Wespi, Towards a Taxonomy of Intrusion Detection Systems, *Computer Networks*, vol. 31, 8, pp. 805-822, 2025
- [24] M. Farooq and M. H. Khan, "Signature-Based Intrusion Detection System in Wireless 6G IoT Networks," *Journal on Internet of Things*, vol. 4, no. 3, pp. 155-168, 2022. [Online]. Available: <https://sos-vo.org/publications/signature-based-intrusion-detection-system-wireless-6g-iot-networks>
- [25] A. Moorthy, A. Ahamed, D. Sujatha, and M. F. G. Ramesh, "An Analysis of Signature-Based Components in Hybrid Intrusion Detection Systems," *Proceedings of the International Conference on Cybersecurity*, 2021. [Online]. Available: <https://sos-vo.org/publications/analysis-signature-based-components-hybrid-intrusion-detection-systems>
- [26] Kwon, H. Y., Kim, T., & Lee, M. K. (2022). Advanced intrusion detection combining signature-based and behavior-based detection methods. *Electronics*, 11(6), 867.
- [27] M. Ahmed et al., "A Survey of Anomaly Detection Techniques in Network Intrusion Detection Systems," *IEEE Access*, vol. 9, pp. 121901-121929, 2021, doi: 10.1109/ACCESS.2021.3109082.
- [28] Y. Li et al., "Deep Learning for Anomaly-Based Network Intrusion Detection: A Comprehensive Review," *IEEE Communications Surveys & Tutorials*, vol. 24, no. 2, pp. 1224-1267, 2022, doi: 10.1109/COMST.2022.3159584.
- [29] J. Zhang and Y. Chen, "Adaptive Statistical Models for Real-Time Anomaly Detection in Industrial IoT," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 5, pp. 3421-3432, 2022, doi: 10.1109/TII.2021.3112345.
- [30] S. Wang et al., "Ensemble Learning for Anomaly Detection: A Comparative Study of Isolation Forest Variants," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 3, pp. 2876-2890, 2023, doi: 10.1109/TKDE.2021.3134567.

- [31] K. Lin et al., "Temporal Anomaly Detection in Network Traffic Using Attention-Based LSTM," *IEEE Transactions on Network and Service Management*, vol. 19, no. 1, pp. 548-562, 2022, doi: 10.1109/TNSM.2021.3128765.
- [32] R. Kumar et al., "Challenges in Deploying Machine Learning-Based Intrusion Detection Systems: A Practical Perspective," *IEEE Security & Privacy*, vol. 20, no. 3, pp. 44-53, 2022, doi: 10.1109/MSEC.2022.3145678.
- [33] H. Park and T. Kim, "Adaptive Thresholding for Anomaly Detection in Dynamic Cloud Environments," *IEEE Transactions on Cloud Computing*, vol. 11, no. 1, pp. 123-136, 2023, doi: 10.1109/TCC.2022.3214567.
- [34] L. Chen et al., "Explainable AI for Security: Interpreting Anomaly Detection in Network Intrusion Detection Systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 2, pp. 1563-1577, 2023, doi: 10.1109/TDSC.2022.3208765.
- [35] A. Buczak and E. Guven, "A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1153-1176, 2016. [2] L.
- [36] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation Forest," in *Proc. IEEE ICDM*, 2008, pp. 413–422. [4] H. Liu and H. Motoda, "Feature Selection for Knowledge Discovery and Data Mining," *Springer*, 1998
- [37] .C. Cortes and V. Vapnik, "\"Support-Vector Networks,\" **Machine Learning**, vol. 20, no. 3, pp. 273–297, 1995. DOI: [10.1007/BF00994018]
- [38] J. R. Quinlan, "Induction of Decision Trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, Mar. 1986. doi: 10.1023/A:1022643204877
- [39] S. Haykin, *Neural Networks and Learning Machines*, 3rd ed., Pearson Education, 2009.
- [40] H. Zhang, "The Optimality of Naïve Bayes," *Proceedings of the 17th International FLAIRS Conference*, pp. 563–567, 2004.
- [41] K. M. Faraoun and A. Boukelif, "Neural networks learning improvement using the K-means clustering algorithm to detect network intrusions," *IEEE International Symposium on Computational Intelligence and Intelligent Informatics*, pp. 237–242, 2006. Doi 10.1109/ISCIII.2006.247122

- [42] Y. Kim, K. Shin, and H. Kim, "A CNN-based network intrusion detection system for IoT networks," *IEEE Access*, vol. 9, pp. 121806–121817, 2021, doi: 10.1109/ACCESS.2021.3109733.
- [43] Y. Meidan et al., "N-BaIoT: Network-based detection of IoT botnet attacks using deep autoencoders," *IEEE Pervasive Computing*, vol. 17, no. 3, pp. 12–22, Jul.–Sept. 2018, doi: 10.1109/MPRV.2018.03367731.
- [44] A. Diro and N. Chilamkurti, "Distributed Attack Detection Scheme Using Deep Learning Approach for Internet of Things," *Future Generation Computer Systems*, vol. 82, pp. 761–768, May 2018. doi: 10.1016/j.future.2017.08.043
- [45] A. T. K. Al-Khayyat and O. Nuri Ucan, "A Multi-Branched Hybrid Perceptron Network for DDoS Attack Detection Using Dynamic Feature Adaptation and Multi-Instance Learning," in *IEEE Access*, vol. 12, pp. 192618-192638, 2024, doi: 10.1109/ACCESS.2024.3508028.
- [46] Verma, A., Saha, R., Kumar, N., & Kumar, G. (2022). A detailed survey of denial of service for IoT and multimedia systems: Past, present and futuristic
- [47] Dhayanidhi, G. (2022). Research on IoT threats & implementation of AI/ML to address emerging cybersecurity issues in IoT with cloud computing.
- [48] D. E. Denning, "An intrusion-detection model," *IEEE Trans. Softw. Eng.*, vol. SE-13, no. 2, pp. 222–232, Feb. 1987.
- [49] L. Liao, H. Lin, Y. Lin, and K. Tung, "Intrusion detection system based on autoencoder neural networks," in *Proc. Int. Conf. Knowl. Sci. Eng.*, 2013.
- [50] Y. Kim, K. Shin, and H. Kim, "A CNN-based network intrusion detection system for IoT networks," *IEEE Access*, vol. 9, pp. 121806–121817, 2021, doi: 10.1109/ACCESS.2021.3109733.
- [51] H. Yin, A. Yazidi, M. Ahmad, and H. M. Chung, "Intrusion Detection Using Recurrent Neural Networks and Reinforcement Learning," *IEEE Access*, vol. 9, pp. 167563–167573, 2021, doi: 10.1109/ACCESS.2021.3133498.

- [52] R. Vinayakumar, K. P. Soman, and P. Poornachandran, "Evaluating Deep Learning Approaches to Intrusion Detection for Cyber-Security Applications," in *Proc. IEEE Int. Conf. Adv. Comput. Commun. (ICACC)*, 2017, pp. 1–8, doi: 10.1109/ICACC.2017.48.
- [53] T. Shon and J. Moon, "A hybrid machine learning approach to network anomaly detection," *Inf. Sci.*, vol. 177, no. 18, pp. 3799–3821, Sept. 2007, doi: 10.1016/j.ins.2007.03.025.
- [54] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "HAST-IDS: Learning Hierarchical Spatial-Temporal Features Using Deep Neural Networks to Improve Intrusion Detection," *IEEE Access*, vol. 6, pp. 1792–1806, 2018, doi: 10.1109/ACCESS.2018.2808202.
- [55] A. Diro and N. Chilamkurti, "Distributed Attack Detection Scheme Using Deep Learning Approach for Internet of Things," *Future Gener. Comput. Syst.*, vol. 82, pp. 761–768, May 2018, doi: 10.1016/j.future.2017.08.043.
- [56] Y. Meidan et al., "N-BaIoT: Network-based detection of IoT botnet attacks using deep autoencoders," *IEEE Pervasive Comput.*, vol. 17, no. 3, pp. 12–22, Jul.–Sept. 2018, doi: 10.1109/MPRV.2018.03367731.
- [57] Z. Zhang et al., "Anomaly Detection using LSTM in Cybersecurity," in *Proc. IEEE Int. Conf. Cyber Secur.*, 2019.
- [58] X. Liang et al., "Feature Selection using Grey Wolf Optimizer for Intrusion Detection," in *Proc. IEEE Conf. Intell. Syst. Appl.*, 2020.
- [59] A. Gupta et al., "Hybrid IDS using SVM and K-Means Clustering," in *Proc. IEEE Conf. Comput. Commun. Syst.*, 2018.

- [60] E. Lee and S. J. Stolfo, "Data mining approaches for intrusion detection," in *Proc. DARPA Intrusion Detection Evaluation*, 2000.
- [61] P. Mukkamala et al., "Intrusion detection using neural networks and machine learning," in *Proc. IEEE Int. Conf. Softw. Eng.*, 2005.
- [62] B. Sommer and V. Paxson, "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection," in *Proc. IEEE Symp. Future Internet Technol.*, 2010.
- [63] Manthena, H., Shajarian, S., Kimmell, J., Abdelsalam, M., Khorsandroo, S., & Gupta, M. (2024). Explainable Malware Analysis: Concepts, Approaches and Challenges. *arXiv preprint arXiv:2409.13723*.
- [64] R. Mirkovic and P. Reiher, "A Taxonomy of DDoS Attack and DDoS Defense Mechanisms," *ACM Comput. Surv.*, vol. 24, no. 3, pp. 1–36, Sept. 2018
- [65] W. Zhou, J. Huang, and D. Zhang, "MitM Attacks in Wireless Networks: An Overview," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 2, pp. 1458–1479, 2018, doi: 10.1109/COMST.2018.2836543.
- [66] J. C. Zhu, H. T. Dinh, and T. E. Chou, "Advanced Persistent Threats: Techniques, Strategies, and Solutions," in *Proc. IEEE Symp. Secur. Privacy*, 2020, pp. 94–101.
- [67] L. Chen, M. Mao, and Z. Liu, "Security Challenges and Solutions in IoT," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2671–2682, Aug. 2018, doi: 10.1109/JIOT.2018.2812365.
- [68] K. Nguyen, M. O. O. K. Alzahrani, and A. N. S. Othman, "AI-Driven Cyber Attacks and Defense Strategies," in *Proc. IEEE Int. Conf. Smart Netw.*, 2020, pp. 67–74.
- [69] L. Liao, H. Lin, Y. Lin, and K. Tung, "Intrusion detection system based on autoencoder neural networks," in *Proc. Int. Conf. Knowl. Sci. Eng.*, 2013.
- [70] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "HAST-IDS: Learning Hierarchical Spatial-Temporal Features Using Deep Neural Networks to Improve Intrusion Detection," *IEEE Access*, vol. 6, pp. 1792–1806, 2018, doi: 10.1109/ACCESS.2018.2808202.

- [71] H. Yin, A. Yazidi, M. Ahmad, and H. M. Chung, "Intrusion Detection Using Recurrent Neural Networks and Reinforcement Learning," *IEEE Access*, vol. 9, pp. 167563–167573, 2021, doi: 10.1109/ACCESS.2021.3133498.
- [72] R. Vinayakumar, K. P. Soman, and P. Poornachandran, "Evaluating Deep Learning Approaches to Intrusion Detection for Cyber-Security Applications," in *Proc. IEEE Int. Conf. Adv. Comput. Commun. Syst.*, 2017, pp. 1–8, doi: 10.1109/ICACC.2017.48.
- [73] E. M. Clarke, "Explainable AI in Cybersecurity: Trends and Challenges," *IEEE Secur. Priv.*, vol. 18, no. 2, pp. 27–33, Mar.-Apr. 2020, doi: 10.1109/MSP.2020.2981234.
- [74] S. K. Singh, "Interpretability of Deep Learning Models for Cybersecurity," in *Proc. IEEE Int. Conf. Cyber Secur.*, 2020, pp. 56–62.
- [75] A. Patel and V. K. Singh, "Securing Cloud Environments: Challenges and Best Practices," *IEEE Trans. Cloud Comput.*, vol. 8, no. 2, pp. 403–415, Apr. 2020, doi: 10.1109/TCC.2020.2987654.
- [76] M. Ring, S. Wunderlich, D. Grödl, D. Landes, and A. Hotho, "Flow-based network traffic generation using generative adversarial networks," *Computers & Security*, vol. 82, pp. 156–172, May 2019, doi: 10.1016/j.cose.2019.01.012.
- [77] L. Chen, M. Mao, and Z. Liu, "Security Challenges and Solutions in IoT," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2671–2682, Aug. 2018, doi: 10.1109/JIOT.2018.2812365.
- [78] M. Jakobsson and S. Myers, "Phishing and Countermeasures: Understanding the Increasing Problem of Electronic Identity Theft," *IEEE Secur. Priv.*, vol. 7, no. 5, pp. 42–50, Sept. 2017.
- [79] Farhan, B. I., & Jasim, A. D. (2022). Performance analysis of intrusion detection for deep learning model based on CSE-CIC-IDS2018 dataset. *Indonesian Journal of Electrical Engineering and Computer Science*, 26(2), 1165-1172.
- [80] Zoghi, Z., & Serpen, G. (2024). UNSW-NB15 computer security dataset: Analysis through visualization. *Security and Privacy*, 7(1), e331.
- [81] Boukhamla, A., & Gaviro, J. C. (2021). CICIDS2017 dataset: performance improvements and validation as a robust intrusion detection system testbed. *International Journal of Information and Computer Security*, 16(1-2), 20-32.

- [82] Changyong, F. E. N. G., Hongyue, W. A. N. G., Naiji, L. U., Tian, C. H. E. N., Hua, H. E., Ying, L. U., & Xin, M. T. (2014). Log-transformation and its implications for data analysis. *Shanghai archives of psychiatry*, 26(2), 105.
- [83] Shanker, M., Hu, M. Y., & Hung, M. S. (1996). Effect of data standardization on neural network training. *Omega*, 24(4), 385-397.
- [84] Patro, S. G. O. P. A. L., & Sahu, K. K. (2015). Normalization: A preprocessing stage. *arXiv preprint arXiv:1503.06462*.
- [85] Biswas, B., Ghosh, S. K., & Ghosh, A. (2020). DVAE: deep variational auto-encoders for denoising retinal fundus image. *Hybrid machine intelligence for medical image analysis*, 257-273.
- [86] Shi, Y., & Eberhart, R. C. (1999, July). Empirical study of particle swarm optimization. In *Proceedings of the 1999 congress on evolutionary computation-CEC99* (Cat. No. 99TH8406) (Vol. 3, pp. 1945-1950). IEEE.
- [87] Trojovský, P., & Dehghani, M. (2022). Pelican optimization algorithm: A novel nature-inspired algorithm for engineering applications. *Sensors*, 22(3), 855.
- [88] Taşcı, E. (2019). A meta-ensemble classifier approach: Random rotation forest. *Balkan Journal of Electrical and Computer Engineering*, 7(2), 182-187.
- [89] Joachims, T. (1999). Svmlight: Support vector machine. *SVM-Light Support Vector Machine* <http://svmlight.joachims.org/>, University of Dortmund, 19(4), 25.
- [90] Wickramasinghe, I., & Kalutarage, H. (2021). Naive Bayes: applications, variations and vulnerabilities: a review of literature with code snippets for implementation. *Soft Computing*, 25(3), 2277-2293.
- [91] Salman, H. A., Kalakech, A., & Steiti, A. (2024). Random forest algorithm overview. *Babylonian Journal of Machine Learning*, 2024, 69-79.
- [92] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, Jun. 2006. doi: 10.1016/j.patrec.2005.10.010.
- [93] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge, U.K.: Cambridge Univ. Press, 2008.
- [94] D. M. Powers, "Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation," *J. Mach. Learn. Technol.*, vol. 2, no. 1, pp. 37–63, 2011.

- [95] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. New York, NY, USA: Springer, 2009.
- [96] Hnamte, V., & Hussain, J. (2023). DCNNBiLSTM: An efficient hybrid deep learning-based intrusion detection system. *Telematics and Informatics Reports*, 10, 100053.
- [97] Talukder, M. A., Islam, M. M., Uddin, M. A., Hasan, K. F., Sharmin, S., Alyami, S. A., & Moni, M. A. (2024). Machine learning-based network intrusion detection for big and imbalanced data using oversampling, stacking feature embedding and feature extraction. *Journal of big data*, 11(1), 33.
- [98] Gu, J., & Lu, S. (2021). An effective intrusion detection approach using SVM with naïve Bayes feature embedding. *Computers & Security*, 103, 102158.
- [99] Al-Khayyat, A. T. K., & Ucan, O. N. (2024). A multi-branched hybrid perceptron network for DDoS attack detection using dynamic feature adaptation and multi-instance learning. *IEEE Access*.