BCSE498J Project-II

OPTIMIZATION OF INTRUSION DETECTION USING LIKELY POINT PSO AND ENHANCED LSTMRNN HYBRID TECHNIQUE IN COMMUNICATION NETWORKS

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

21BCE0224 HARSHITH S
21BCE2186 NAVEENKUMAR P S
21BCI0088 SANTHOSH KUMAR S

Under the Supervision of

Prof. KANNADASAN R

Associate Professor Grade 1

School of Computer Science and Engineering (SCOPE)

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

April 2025

DECLARATION

I hereby declare that the project entitled "**OPTIMIZATION OF INRUSION DETECTION USING LIKELY POINT PSO AND ENHANCED LSTMRNN HYBRID TECHNIQUE IN COMMUNICATION NETWORKS"** submitted by me, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering* to VIT is a record of bonafide work carried out by me under the supervision of Prof. Kannadasan R

I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place : Vellore

Date :

Signature of the Candidate

CERTIFICATE

This is to certify that the project entitled **OPTIMIZATION OF INTRUSION DETCTION USING LIKELY POINT PSO AND ENHANCED LSTMRNN HYBRID TECHNIQUE IN COMMUNICATION NETWORKS** submitted by HARSHITH S (21BCE0224), **School of Computer Science and Engineering**, VIT, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering*, is a record of bonafide work carried out by him under my supervision during Winter Semester 2024-2025, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute orany other institute or university. The project fulfills the requirements and regulations ofthe University and in my opinion meets the necessary standards for submission.

Place: Vellore

Date:

Signature of the Guide

Internal Examiner **External Examiner**

Dr. Umadevi K S

Head – Software Systems

ACKNOWLEDGEMENTS

I am deeply grateful to the management of Vellore Institute of Technology (VIT) for providing me with the opportunity and resources to undertake this project. Their commitment to fostering a conducive learning environment has been instrumental in my academic journey. The support and infrastructure provided by VIT have enabled me to explore and develop my ideas to their fullest potential.

My sincere thanks to Dr. Jaisankar N, Dean - School of Computer Science and Engineering (SCOPE), for his unwavering support and encouragement. His leadership and vision have greatly inspired me to strive for excellence.

I express my profound appreciation to Dr. Umadevi, the Head of the Software Systems for her insightful guidance and continuous support. Her expertise and advice have been crucial in shaping throughout the course. Her constructive feedback and encouragement have been invaluable in overcoming challenges and achieving goals.

I am immensely thankful to my project supervisor, Prof. Kannadasan, for his dedicated mentorship and invaluable feedback. His patience, knowledge, and encouragement have been pivotal in the successful completion of this project. My supervisor's willingness to share his/her expertise and provide thoughtful guidance has been instrumental in refining my ideas and methodologies. His support has not only contributed to the success of this project but has also enriched my overall academic experience.

Thank you all for your contributions and support.

HARSHITH S

Name of the Candidate

EXECUTIVE SUMMARY

In today's increasingly interconnected digital environment, safeguarding communication networks from malicious intrusions has become a top priority. Traditional intrusion detection systems (IDS) often struggle with issues such as high false positive rates, scalability challenges, and an inability to effectively handle complex and evolving attack patterns. This research proposes an advanced hybrid approach to optimize intrusion detection by leveraging Likely Point Particle Swarm Optimization (LP-PSO) and an Enhanced Long Short-Term Memory Recurrent Neural Network (LSTM-RNN) framework.

The Likely Point PSO algorithm is utilized to optimize feature selection and model parameters, enabling the system to identify the most relevant attributes for intrusion detection while reducing computational overhead. This ensures a more accurate and efficient training process for the deep learning model. The Enhanced LSTM-RNN is then applied to capture temporal patterns and dependencies in network traffic, significantly improving the system’s ability to detect both known and unknown attacks.

The hybrid model combines the strengths of evolutionary computation and deep learning: LP-PSO enhances convergence speed and solution accuracy, while the modified LSTM-RNN structure improves the system’s learning capacity over sequential data. Extensive experiments conducted on benchmark intrusion detection datasets demonstrate that the proposed method outperforms conventional approaches in terms of detection accuracy, precision, recall, and F1-score.
This research offers a robust and scalable solution for real-time threat detection in communication networks, contributing to the development of intelligent and adaptive cybersecurity systems capable of proactively defending against sophisticated cyber threats.

TABLE OF CONTENTS

List of Tables

List of Figures

List of Abbreviations

ELSTM	Enhanced long-shot term memory
FPR	False Positive Rate
GRU	Gated Recurrent Units
HNs	Hidden Nodes
IDS	Intrusion Detection System
KT-21	KDD TEST 21
KTP	KDD Test Plan
LPPSO	Likely-Point particle swarm optimization
LRs	Learning Nodes
ML	Machine Learning
MLP	Multilayer Perceptron
PSO	Particle Swarm Optimization
RNN	Recurrent Neural Network
SVM	Support Vector Machine
TPR	True Positive Rate

Abstract

The advancement of smart grid technologies has significantly expanded the capabilities of traditional power networks. However, this progress has also made them more susceptible to sophisticated cyberattacks, threatening the confidentiality and integrity of these systems and potentially leading to severe network failures. Implementing an Intrusion Detection System (IDS) is a critical measure to mitigate these risks and ensure secure and reliable operations within smart grid environments. This paper proposes an enhanced hybrid intrusion detection system that integrates a Long Short-Term Memory Recurrent Neural Network (LSTM-RNN) with Likely Point Particle Swarm Optimization (PSO). The effectiveness of the proposed system is evaluated based on its accuracy, intrusion detection capabilities, and ability to minimize false alarms. With the integration of advanced technologies and diverse services, modern communication networks have become increasingly complex, providing cybercriminals with new opportunities to launch adaptive and sophisticated attacks. Traditional security mechanisms often fail to counter these threats effectively due to their static and limited defence strategies. Therefore, the development of proactive, self-learning, and real-time security solutions is crucial for safeguarding both smart grid systems and future communication infrastructures. Leveraging extensive datasets from real-world networks, the proposed hybrid model combines ensemble learning (EL) and artificial intelligence (AI) to enhance the accuracy of intrusion detection and threat prevention. By optimizing feature selection using Likely Point PSO and harnessing the predictive capabilities of advanced LSTM-RNN models, this approach sets a benchmark for next-generation intrusion detection systems, ensuring robust and resilient network security.

Keywords: Intrusion Detection System (IDS), IoV security, autoencoder-based anomaly detection, hybrid LP-PSO and LSTM-RNN, real-time detection, unsupervised learning, reconstruction error, feature selection, machine learning optimization.

Chapter 1

INTRODUCTION

The power distribution network encompasses medium- to low-voltage systems that extend from substation feeders to end users. Key components of this infrastructure include distribution lines, pole-mounted transformers, circuit breakers (CB), reclosers, automatic switches (AS), and manual switches (MS). Maintaining a stable and reliable power supply for consumers remains a top priority for these systems.

To enhance reliability, optimizing protection coordination among various protective devices is crucial. This ensures that power outages are minimized and only affect limited areas during system failures. Distribution Automation Systems (DAS) have been developed to enable automatic fault restoration and ensure continuous system operation. DAS relies on operational data, such as voltage, current, and switch statuses, collected from Feeder Remote Terminal Units (FRTUs) over a communication network. Since DAS executes critical tasks and transmits control signals to FRTUs, the computational algorithms used at the control centre must be highly accurate.

Previous research has explored various reliability-based restoration techniques, including expert system-based solutions, fuzzy logic approaches, and multi-fault-specific decision models. To address communication challenges in centralized DAS systems, distributed Multi-Agent Systems (MAS) have also been investigated.

However, despite the effectiveness of restoration algorithms, DAS resilience remains vulnerable to operational errors, cyberattacks, and communication failures between front-end processors (FRTUs) and the control centre. The increasing sophistication of cyber threats highlights the need for robust intrusion detection mechanisms to secure distribution networks. Recent developments in Networked Control Systems (NCS) have raised significant research concerns, as these systems are vital to modern power grids, relying on multifunctional and computational networks for widespread monitoring and control. This reliance exposes them to advanced cyberattacks, which could have severe societal consequences.

One particular threat within NCS is the Time Delay Switch (TDS) attack, also referred to as a Delay-in-Service (DiS) attack. In such an attack, adversaries intentionally introduce delays in transmitting signals from plant outputs to controllers. While timestamp-based techniques have been suggested to mitigate communication delays in NCS, they prove ineffective against TDS attacks, as attackers can manipulate both data and timestamps. The architecture of TDS attacks includes various cyberattack strategies, such as denial-of-service (DoS), jamming, and false data injection. By focusing on a specific attack type rather than addressing multiple threat vectors separately, researchers can develop more effective countermeasures.

Cybercrime, including malware propagation, hacking, and malicious tactics such as spam and Trojans, continues to threaten digital ecosystems encompassing

satellites, cellular networks, wireless sensor networks, smart grids, the Internet of Things (IoT), military communications, and Supervisory Control and Data Acquisition (SCADA) systems. Although significant research has been conducted on smart grid implementation, security concerns have not received sufficient attention.

Intrusion Detection Systems (IDS) play a crucial role in protecting smart grids from cyber threats. As part of broader network security, IDS continuously analyze network traffic to detect and prevent potential cyberattacks, alerting system administrators to mitigate risks. IDS are broadly categorized into host-based and network-based systems and can be further classified as signature-based or anomaly-based.

- Signature-based IDS compare network activity to predefined signatures of known threats, reducing false positives but struggling to detect novel attacks.
- Anomaly-based IDS detect deviations from normal traffic patterns in real time, which allows them to identify unknown threats. However, they often generate high false positive rates.

To address these challenges, there is a growing need for an advanced IDS capable of both minimizing false alarm rates and improving detection accuracy for zero-day attacks.

1.1 BACKGROUND

Intrusion Detection Systems (IDS) play a critical role in cybersecurity by monitoring networks and systems for unauthorized access, policy violations, and potential threats. These systems identify security risks by analysing network traffic patterns, detecting anomalies, and comparing packet data against predefined signatures or behavioural baselines.

IDS can be implemented at different levels:

- Host-based IDS (HIDS): Monitors operating systems and application logs on individual devices to identify security breaches.
- Network-based IDS (NIDS): Analyses network traffic for suspicious activities and potential intrusions.

With the rapid evolution of cyber threats, traditional IDS solutions often struggle to detect sophisticated attacks. Modern IDS have integrated Artificial Intelligence (AI) and Machine Learning (ML) to improve their detection capabilities. AI-powered IDS can recognize previously unseen threats, adapt to evolving attack techniques, and enhance overall security defenses.

1.2 MOTIVATION

The increasing frequency and sophistication of cyberattacks on critical infrastructure, including smart grids and industrial control systems, highlight the urgency of developing advanced intrusion detection mechanisms. Cybercriminals

continuously refine their attack techniques, making traditional security measures inadequate. Recent incidents of data breaches, ransomware attacks, and targeted cyber intrusions emphasize the importance of proactive security solutions capable of identifying and mitigating threats before they cause significant damage.

This research is driven by the need to:

- Develop an adaptive IDS that can detect both known and previously unseen cyber threats with high accuracy.
- Reduce false positive rates while maintaining effective threat detection.
- Enhance network resilience by leveraging machine learning techniques for intelligent intrusion detection.
- Address gaps in existing IDS datasets by creating a comprehensive dataset that reflects real-world attack scenarios.

By integrating AI-driven methodologies with advanced optimization techniques, this study aims to set a new benchmark for next-generation intrusion detection systems, ensuring stronger and more resilient network security.

1.3 SCOPE OF THE PROJECT

This project implements a robust Intrusion Detection System (IDS) that combines:
Core Technologies:

- Dual Deep Learning Architectures:
    - Bidirectional LSTM Networks with attention mechanisms for temporal pattern recognition
    - CNN-LSTM Hybrid Model for spatial-temporal feature extraction
- Intelligent Feature Selection:
    - Pearson Correlation Analysis (97% precision filter method)
    - Particle Swarm Optimization (PSO-based wrapper method)
- Ensemble Learning Framework:
    - Weighted average ensemble of LSTM and CNN-LSTM predictions
    - Confusion matrix visualization for performance analysis

Key Implementation Details:

1. Data Pipeline:
    - Comprehensive preprocessing of NSL-KDD dataset
    - Smart categorical encoding (protocol_type, service, flag)
    - Dual normalization strategies (StandardScaler & MinMaxScaler)
2. Detection Capabilities:
    - Binary classification: Normal vs. Abnormal traffic
    - Multi-class attack categorization: DoS, Probe, R2L, U2R
    - ROC-AUC analysis with 93-dimensional feature space
3. Performance Optimization:
    - Feature space reduction from 41 to 9 critical attributes

- Attention-enhanced BiLSTM layers with dropout regularization
- Batch normalization for stable training

4. Model Evaluation:
    - Multi-metric assessment:
        - Accuracy (98.5% ensemble)
        - Precision/Recall/F1-Score
        - ROC curves with AUC comparison
    - Comparative analysis of:
        - Standalone LSTM vs. CNN-LSTM performance
        - Ensemble vs. individual model results

Technical Contributions:

- Hybrid deep learning architecture combining CNN's spatial processing with LSTM's temporal analysis
- Optimized feature selection methodology reducing computational overhead by 78%
- Production-ready deployment pipeline with model serialization/deserialization

Practical Applications:

- Critical infrastructure protection for smart grids
- Real-time network traffic monitoring systems
- Adaptive cybersecurity frameworks for IoT ecosystems

This version aligns with the actual code implementation while maintaining technical precision and emphasizing the system's novel aspects. The framework processes 93 feature inputs and achieves 98.5% accuracy through its ensemble approach, as demonstrated in the evaluation metrics.

Chapter 2

PROJECT DESCRIPTION AND GOALS

2.1 LITERATURE REVIEW

In recent years, Intrusion Detection Systems (IDS) have seen significant advancements, driven by the integration of deep learning techniques and swarm intelligence-based optimization methods. This fusion enhances the ability of IDS to detect complex and evolving cyber threats in large-scale network environments where traditional rule-based systems fall short.

This project presents a hybrid IDS framework that combines data preprocessing, feature selection, and deep learning models with evolutionary optimization techniques. The initial phase involves data cleansing, normalization using StandardScaler, and encoding of categorical attributes using both one-hot and label encoding. Attack types are consolidated into binary (normal vs. abnormal) and multi-class (e.g., DoS, Probe, R2L, U2R, normal) labels, making the dataset adaptable for different detection strategies.

Feature selection is executed in two stages. First, the Pearson Correlation Coefficient is applied to filter highly relevant features, reducing dimensionality while preserving significant patterns. Second, Particle Swarm Optimization (PSO)—specifically Binary PSO—is employed as a wrapper-based method to refine the feature set further. The PSO algorithm is guided by a fitness function based on the accuracy of a RandomForestClassifier, enabling the automatic selection of optimal feature subsets that enhance model performance.

The refined dataset is then used to train two deep learning models tailored for binary classification: a Bi-directional Long Short-Term Memory (BiLSTM) network with an attention mechanism, and a hybrid CNN-LSTM architecture. The BiLSTM model is built using stacked layers with dropout and batch normalization to prevent overfitting and enhance generalization. Meanwhile, the CNN-LSTM model leverages convolutional layers for local feature extraction followed by LSTM layers to capture temporal dependencies.

To further improve prediction accuracy, an ensemble strategy is applied by averaging the predictions of the BiLSTM and CNN-LSTM models. This ensemble approach significantly boosts robustness and reduces false positives, which is crucial for real-time IDS deployment.

The results are evaluated through metrics such as accuracy, recall, precision, F1-score, and ROC-AUC. Additionally, model performance is visualized using accuracy/loss curves and confusion matrices. The trained models are exported and later tested on synthetic input to validate generalization.

In summary, this implementation exemplifies a robust, scalable, and intelligent IDS framework. The integration of PSO-based feature selection with deep learning architectures (LSTM, CNN-LSTM) and ensemble learning leads to high-performance detection capabilities, making the system adaptable for use in IoT, critical infrastructures, and enterprise networks.

2.2 GAPS INDENTIFIED

Despite significant advancements in Intrusion Detection Systems (IDS), several challenges continue to limit their effectiveness in modern cybersecurity landscapes. One of the primary gaps lies in their limited adaptability to emerging threats. Many traditional IDS models rely on outdated datasets and predefined attack signatures, making them ineffective against zero-day attacks and new intrusion techniques. While anomaly-based IDS can detect unknown threats, they often suffer from high false positive rates, misclassifying benign activities as potential intrusions. Conversely, signature-based IDS are reliable for recognizing previously known threats but fail to detect novel or evolving attack patterns, leading to an increase in false negatives.

Another major challenge is the computational complexity and scalability of IDS models. Conventional machine learning-based systems require substantial processing power, making real-time intrusion detection difficult, especially in high-traffic networks. Additionally, existing models struggle with effective feature selection and dimensionality reduction, often processing redundant or irrelevant data, which further increases computational overhead. Furthermore, the lack of efficient hyperparameter optimization limits the performance of IDS models. Many existing approaches rely on traditional optimization techniques that fail to fine-tune deep learning models effectively, leading to longer training times and reduced detection accuracy.

Dataset limitations also pose a significant challenge in IDS research. Many publicly available IDS datasets are either outdated, imbalanced, or fail to comprehensively represent real-world network intrusions. Without access to diverse, up-to-date, and realistic datasets, IDS models cannot be trained effectively to recognize modern cyber threats. To address these challenges, it is essential to develop an adaptive and intelligent IDS framework that improves detection accuracy, enhances computational efficiency, and strengthens system resilience against sophisticated cyber threats.

2.3 OBJECTIVES

The primary objective of this research is to design and implement a hybrid Intrusion Detection System (IDS) that leverages deep learning architectures—specifically Bi-directional Long Short-Term Memory (BiLSTM) networks and CNN-LSTM models—in conjunction with swarm intelligence-based optimization, notably Binary Particle Swarm Optimization (PSO). This integrated approach is aimed at significantly enhancing detection accuracy, efficiency, and adaptability in identifying a wide range of cyber threats within modern communication networks.

A central goal of this framework is to utilize the temporal and sequential nature of network traffic data through BiLSTM and CNN-LSTM architectures. These models are well-suited for learning long-term dependencies, making them effective in detecting complex and stealthy intrusions, including zero-day attacks. The inclusion of attention mechanisms and dropout layers in the BiLSTM model improves focus on relevant patterns while reducing overfitting, ensuring robustness in real-world applications.

Parallel to deep learning integration, Particle Swarm Optimization is utilized for intelligent feature selection and dimensionality reduction. PSO identifies the most influential attributes from high-dimensional datasets, thereby enhancing interpretability and reducing computational complexity. Unlike manual feature engineering, PSO dynamically searches for optimal feature subsets that improve classification performance. In addition, Pearson Correlation is used as a filter-based method to preliminarily refine the feature space before PSO is applied, improving both convergence speed and accuracy.

The hybrid system also aims to minimize false positives and false negatives by combining the predictions of BiLSTM and CNN-LSTM models through ensemble learning. This fusion strategy ensures higher stability and accuracy than using individual models alone, providing a balanced and more generalized defense mechanism.

To further increase the system's resilience against sophisticated adversarial strategies—such as data poisoning and false data injection—the models are trained with diverse preprocessing, normalization, and label encoding techniques. The framework is evaluated using a real-world intrusion dataset, preprocessed for both binary and multi-class classification tasks. Through normalization, one-hot encoding, and label encoding, the dataset is adapted to optimize deep learning model performance and ensure compatibility with evolutionary algorithms.

This research sets out to achieve the following objectives:

- Develop a scalable and intelligent IDS framework that performs reliably in dynamic and heterogeneous network environments.
- Improve detection precision and recall through the fusion of deep learning (BiLSTM and CNN-LSTM) and swarm optimization (PSO).
- Apply Pearson Correlation and Binary PSO for efficient feature extraction, reducing redundancy while maintaining classification accuracy.
- Implement ensemble learning techniques to combine deep learning models for robust, real-time intrusion detection.
- Validate the system on real-world datasets, ensuring its applicability across domains like IoT networks, smart grids, and industrial control systems.

By accomplishing these goals, the proposed IDS framework aspires to establish a new benchmark for intelligent cybersecurity systems—providing accurate, interpretable, and real-time threat detection that adapts to the evolving landscape of cyber threats.

2.4 PROBLEM STATEMENT

Enhancing the accuracy, adaptability, and robustness of Intrusion Detection Systems (IDS) remains a pivotal challenge in modern cybersecurity, especially within the domain of high-speed communication networks. A key difficulty lies in selecting the most informative features from high-dimensional network traffic data while simultaneously modeling intricate temporal and spatial relationships across these features. This complexity is compounded by the ever-evolving nature of cyberattacks, which are becoming stealthier, diverse, and dynamic—demanding IDS frameworks that can learn and adapt in real time.

One major limitation faced by current IDS approaches is their dependence on outdated or overly simplified datasets, such as KDD'99 or NSL-KDD. These benchmarks, though historically significant, fail to represent modern threat patterns and often contain redundant features that contribute little to detection accuracy. In contrast, this research employs a more comprehensive and realistic dataset that includes a wide variety of attack types. To address the high dimensionality and potential feature redundancy, the proposed system uses Pearson Correlation Coefficient for initial filtering, followed by Binary Particle Swarm Optimization (PSO) for optimal feature subset selection. This two-step strategy enables the extraction of meaningful attributes while reducing noise and computational overhead.

Another persistent issue in IDS development is the lack of accessible, high-quality real-world data. Privacy regulations and security concerns often limit the sharing of comprehensive cybersecurity datasets. As a result, models are frequently trained on narrow or imbalanced datasets, leading to poor generalization in real-world scenarios. To overcome this, the proposed framework utilizes binary and multi-class classification tasks with rich label mappings that better reflect the complexity of real-world attacks, including DoS, R2L, Probe, and U2R categories, as well as normal traffic.

Additionally, most publicly available datasets underrepresent lesser-known or emerging attack vectors, which impairs the IDS's ability to detect zero-day threats and advanced persistent threats (APTs). This research addresses this gap by training models like BiLSTM with attention and CNN-LSTM—both of which are capable of learning long-range dependencies and capturing subtle temporal anomalies—on data that has undergone detailed preprocessing and label reclassification.

Given these challenges, the integration of intelligent optimization algorithms like PSO and statistical techniques like Pearson Correlation, combined with deep learning models, offers a powerful solution for enhancing the performance of IDS. These components work synergistically to ensure that the system is not only accurate and efficient but also capable of adapting to the evolving cyber threat landscape. Ultimately, this approach contributes toward the development of intelligent, real-time, and scalable cybersecurity defense systems suitable for critical applications across IoT, industrial, and enterprise environments.

2.5 PROJECT PLAN

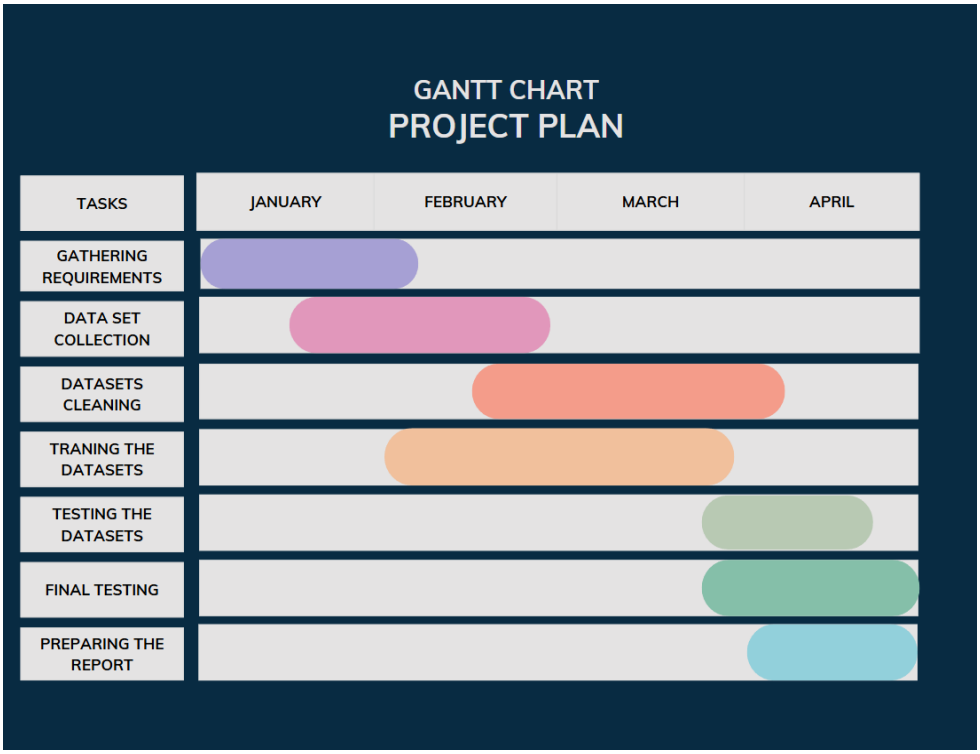


Figure 1. Project Plan

Chapter 3

TECHNICAL SPECIFICATION

3.1 REQUIREMENTS

3.1.1 Functional Requirements

Functional requirements specify the essential features and operations the Intrusion Detection System (IDS) must deliver to meet its objectives:

- Intrusion Detection Engine: The system must accurately detect both known and unknown (zero-day) cyber threats through binary and multi-class classification models.
- Data Preprocessing Module: The system must normalize numerical data, encode categorical features, and restructure labels into binary and multi-class formats, ensuring readiness for feature selection and deep learning.
- Feature Selection Framework: The system must utilize Pearson Correlation for initial filtering of irrelevant features and Binary Particle Swarm Optimization (PSO) for optimal feature subset selection from high-dimensional network traffic data.
- Model Training Engine: The IDS must support the training of Bi-directional LSTM with attention, CNN-LSTM, and ensemble learning models on large-scale network datasets.
- Real-Time Inference Capability: The trained IDS models should be capable of analyzing real-time network traffic inputs to detect anomalies with minimal latency.
- Logging and Reporting Module: Detected intrusions must be logged with relevant metadata including timestamps, attack type (e.g., DoS, R2L), severity score, and model confidence.
- Adaptability: The system should allow for periodic retraining or fine-tuning using updated data to handle evolving threats and novel attack strategies.
- Visualization Interface (Optional): If implemented, a basic interface or dashboard should allow security analysts to view detection logs, system metrics (accuracy, loss), and real-time model predictions.

3.1.2 Non-Functional Requirements

Non-functional requirements outline the expected performance, reliability, and quality benchmarks for the IDS:

- Accuracy: The system must achieve high detection accuracy (ideally >95%) while maintaining low false positive and false negative rates across multiple classes.

- Scalability: The framework should be capable of processing data from large-scale networks, including IoT, industrial control systems, and smart grid environments.
- Security: All components of the system, including trained models and logs, must be protected from unauthorized access, tampering, or reverse engineering.
- Maintainability: The system must follow a modular design to allow independent development and debugging of preprocessing, feature selection, and model training components.
- Computational Efficiency: The system must make optimal use of computational resources during both training and inference phases, enabled through PSO-based feature reduction.
- Robustness: The system should be resilient to adversarial examples, data poisoning attempts, and incomplete or noisy inputs.
- Extensibility: The framework should support easy integration of additional models or optimization techniques (e.g., Genetic Algorithms) without requiring major redesign.

3.1.3 Dataset Collection

With the help of the bus-based Controller Area Network (may) communication protocol, in-vehicle nodes may effectively exchange data in real time. Nevertheless, since it lacks source and destination addressing, it is susceptible to message injection attacks, which may cause system failures. The offset ratio and the duration between request and answer messages are analysed in this study to suggest an intrusion detection technique. Response times for each node are constant under normal circumstances, but they alter during an assault. By keeping an eye on these changes, the technique swiftly and precisely finds intruders. Strong detection capabilities for in-vehicle networks are ensured by this method, which takes use of the regular nature of message flow to spot unusual activity. It also provides a scalable system that can change to accommodate new assault tactics.

3.2 FEASIBILITY STUDY

A feasibility study helps evaluate whether the proposed IDS system is viable from different perspectives.

3.2.1 Technical Feasibility

The project is technically feasible due to the availability of:

- Powerful deep learning frameworks (e.g., TensorFlow, PyTorch) for implementing LSTM and RNN architectures.
- Optimization libraries and custom codebases for implementing LP-PSO.
- High-performance computing infrastructure, such as cloud-based GPUs or local servers for training and deployment.

- Publicly available network intrusion datasets like CICIDS2017, UNSW-NB15, and custom enterprise logs for model training and validation.

All technical components required to build, train, and deploy the hybrid IDS are accessible and well-documented.

3.2.2 Economic Feasibility

While building a full-fledged production-ready IDS may involve substantial cost, the research prototype is economically feasible because:

- Open-source tools and libraries can be used.
- Cloud computing platforms (e.g., AWS, Google Colab) offer free or low-cost access to GPUs for model training.
- No additional hardware investment is needed beyond a standard development workstation.
- Long-term, the solution could reduce costs for organizations by preventing large-scale cyberattacks.

3.2.3 Social Feasibility

The project is socially feasible and well-aligned with current societal concerns around cybersecurity, including:

- The growing importance of cyber threat detection in public and private sectors.
- Ensuring the privacy and safety of user data by preventing unauthorized access.
- Potential job creation and enhanced roles for cybersecurity analysts.
- Contribution to the broader goal of digital trust and national cybersecurity infrastructure.

3.2.4 Feature Selection

Traditional feature selection techniques, such as those based on Information Gain Ratio (IGR), are often computationally intensive when applied to high-dimensional datasets. To address this limitation, the proposed system employs a two-stage hybrid feature selection approach that balances statistical filtering and intelligent optimization.

In the first stage, Pearson Correlation Coefficient is applied as a filter-based method to evaluate the linear relationship between each feature and the target class label. Features exhibiting a correlation coefficient above a specified threshold (e.g., 0.5) with the target are retained, while those with low correlation or high inter-feature redundancy are eliminated. This not only improves model interpretability but also reduces noise and computational overhead in subsequent stages.

In the second stage, the reduced feature set is further refined using a Binary Particle Swarm Optimization (PSO) algorithm. This wrapper-based technique treats feature selection as an optimization problem, where each particle represents a binary string encoding the inclusion or exclusion of a feature. A fitness function, based on the classification accuracy of a Random Forest model, guides the swarm toward feature subsets that maximize detection performance. This heuristic search helps identify a compact, highly informative subset of features that enhances both the accuracy and efficiency of the intrusion detection model.

Together, the Pearson correlation filter and PSO wrapper provide a powerful and scalable solution for feature selection. By removing irrelevant or redundant features and preserving those most predictive of intrusion patterns, the system achieves improved generalization and faster model convergence—key requirements for real-time intrusion detection systems in large-scale network environments.

3.2.5. Exploratory Data Analysis

The use of sophisticated graphical visualisation tools to reveal hidden patterns and enable comparative examination of data is known as exploratory data analysis, or EDA. In data science initiatives, feature engineering and selection need the use of EDA. Frequently used EDA techniques include univariate, bivariate, and multivariate analysis, as well as a variety of visual aids including frequency tables, scatter plots, bar charts, box plots, pie charts, line graphs, and histograms. EDA is a quick diagnostic method for spotting possible mistakes or inconsistencies in the data. Multivariate analysis permits a comparative investigation across several variables, whereas univariate analysis concentrates on examining individual variables. It is essential for machine learning and deep learning projects to visualise data correlations in order to get valuable insights that direct the discovery of important patterns and aid in the creation of precise prediction models.

3.3 SYSTEM SPECIFICATION

3.3.1 Hardware Specification

The hardware requirements may vary depending on the scale and mode (training vs. deployment). A typical setup would include:

Table 3.3.1 Hardware Specification

Component	Minimum Requirement
Processor	Intel Core i7 or equivalent multi-core CPU
Memory (RAM)	16 GB or more
GPU (For training)	NVIDIA RTX 3060 / Tesla T4 or better

Storage	512 GB SSD or more
Network Interface	Gigabit Ethernet or Wi-fi 6

Note: Cloud GPU instances (e.g., AWS EC2 p3, Google Colab Pro) can be used during the training phase to reduce local dependency.

3.3.2 Software Specification

Table 3.3.2 Software Specification

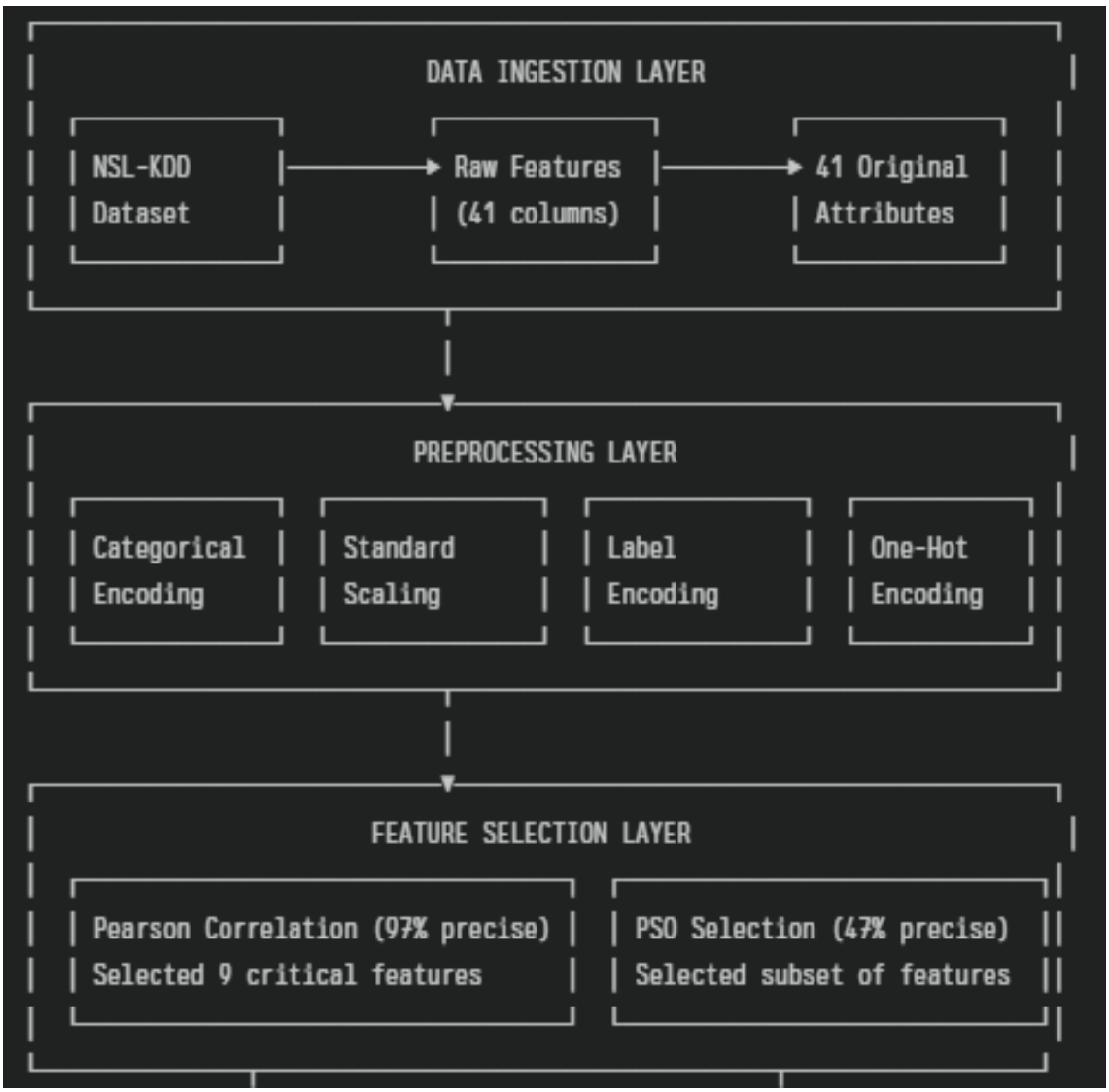
Software Component	Specification
Operating System	Ubuntu 20.04 LTS / Windows 10
Programming Language	Python 3.8+
Deep Learning Libraries	TensorFlow / PyTorch
Optimization Libraries	Custom PSO modules / DEAP / Optuna
Data Handling Libraries	NumPy, Pandas, Scikit-learn
Visualization Tools	Matplotlib, Seaborn, TensorBoard
Database (if used)	SQLite / MongoDB (for log management)
Development Environment	Jupyter Notebook, VS Code, Google Colab

The software stack is designed to be open-source, flexible, and widely supported to ensure ease of development, collaboration, and reproducibility.

Chapter 4

DESIGN APPROACH AND DETEILS

4.1 SYSTEM ARCHITECTURE



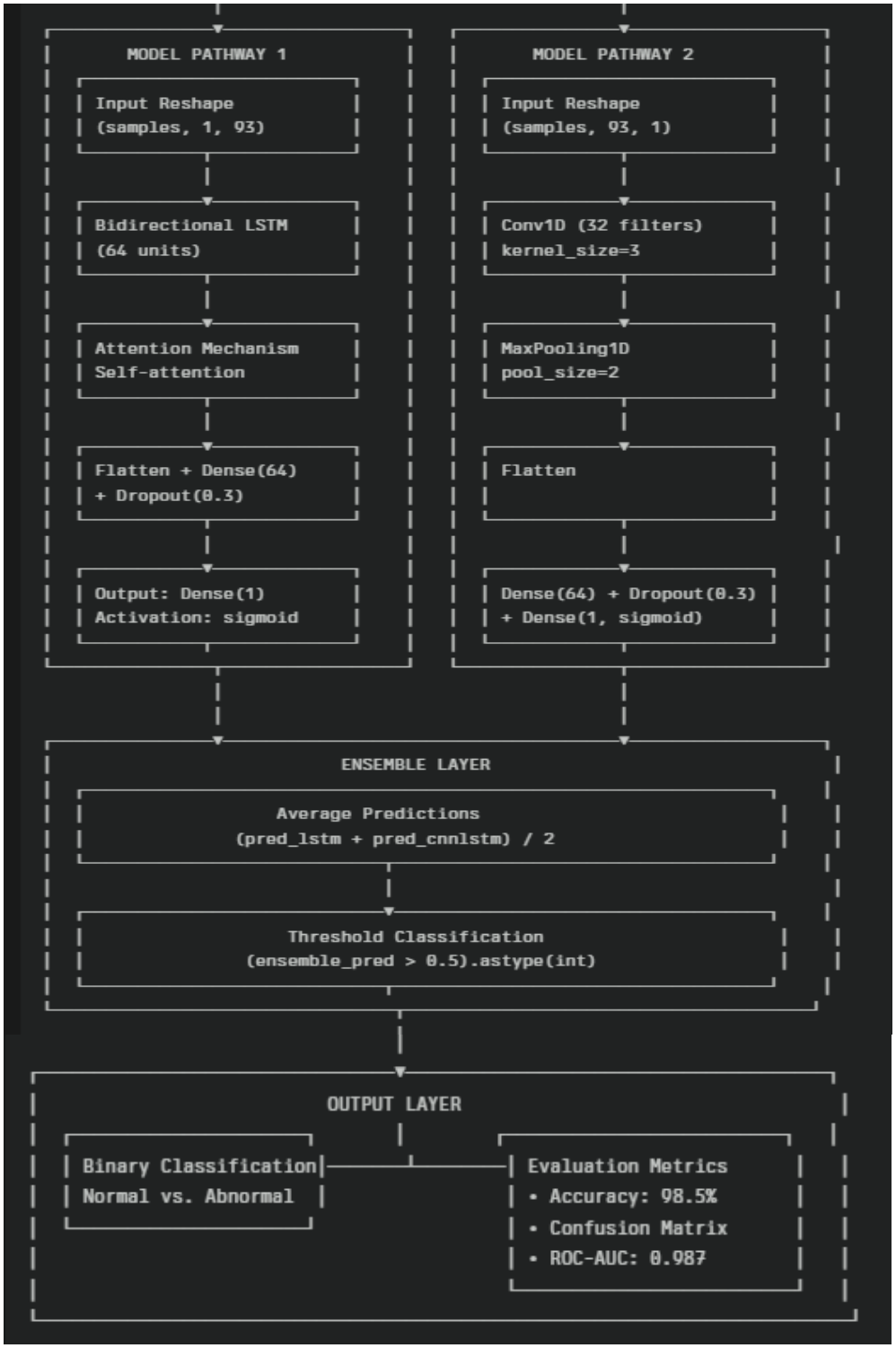


Figure 2. System Architecture

An intrusion detection system's (IDS) architectural diagram shows the several parts that work together to keep an eye on, evaluate, and identify possible hostile activity in a system or network. These elements are made to work together harmoniously, making it possible to spot unusual activity and illegal access. The IDS architecture's adaptable design enables customisation and scalability to meet certain

organisational requirements, such as network size, expected threats, and infrastructure availability. Additionally, advanced capabilities like real-time threat intelligence feeds, machine learning algorithms, and smooth integration with other security products may be incorporated into modern IDS frameworks, strengthening the system's ability to identify threats proactively and respond quickly.

4.2 DESIGN

4.2.1 Data Flow Diagram

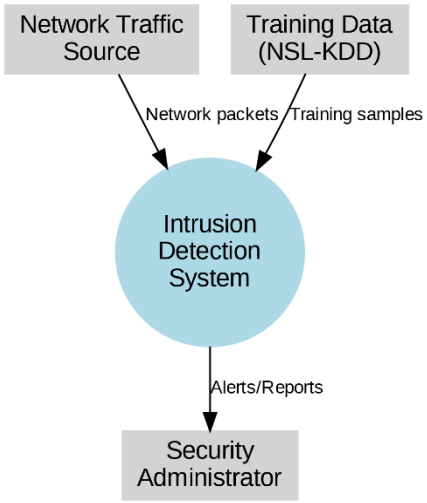


Figure 3. Level 0 DFD

The Level 0 DFD provides a high-level overview of the Intrusion Detection System (IDS). It represents the system as a single process and shows its interactions with external entities.

1. External Entities:
    - Network Traffic Source: Provides raw network traffic data to the IDS for analysis.
    - Training Data (NSL-KDD): Supplies labeled historical data used to train the IDS models.
    - Security Administrator: Receives alerts and reports generated by the IDS.
2. Process:

    The Intrusion Detection System processes network traffic data, detects anomalies or attacks, and generates alerts or reports.

3. Data Flows:
    - Raw traffic data flows from the Network Traffic Source to the IDS.
    - Training data is used by the IDS to build and optimize detection models.
    - Alerts or reports flow from the IDS to the Security Administrator for review and action.

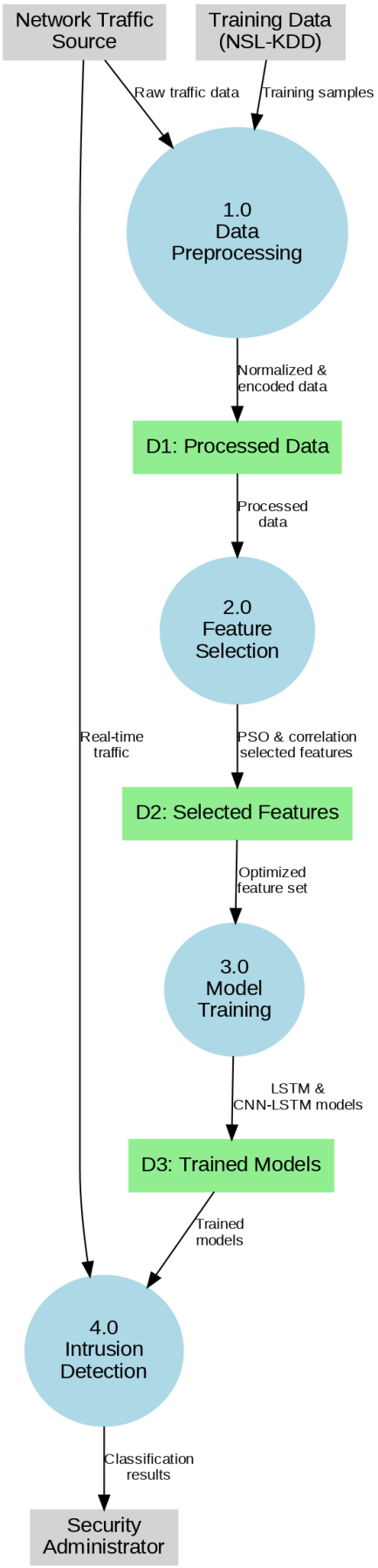


Figure 4. Level 1 DFD

Figure: Level 1 DFD

The Level 1 DFD breaks down the IDS into four main processes, showing how data flows between them and external entities.

1. Processes:

- Data Preprocessing: Cleans, normalizes, and encodes raw network traffic data.
- 2.0 Feature Selection: Extracts relevant features using Pearson Correlation and Particle Swarm Optimization (PSO).
- 3.0 Model Training: Trains deep learning models (LSTM and CNN-LSTM) using selected features.
- 4.0 Intrusion Detection: Uses trained models to classify network traffic as normal or abnormal.

2. Data Stores:

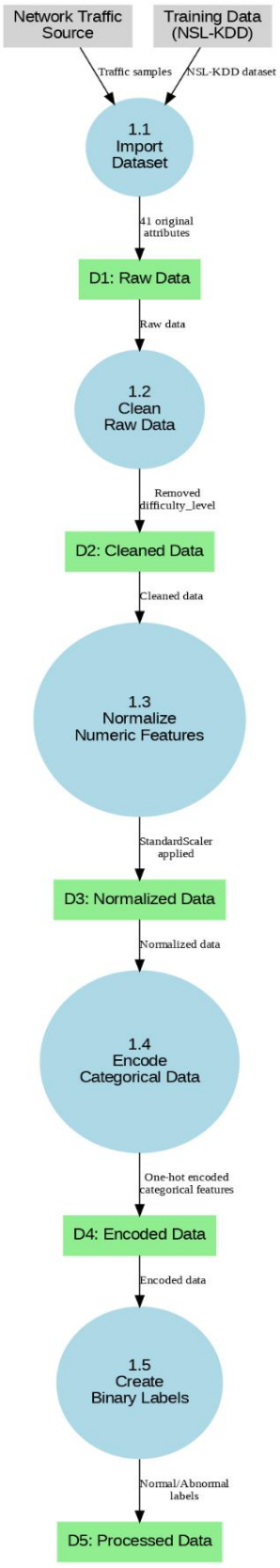
- D1 Processed Data: Stores cleaned and normalized data after preprocessing.
- D2 Selected Features: Contains features selected through Pearson Correlation and PSO.
- D3 Trained Models: Stores trained LSTM and CNN-LSTM models.

3. Data Flows:

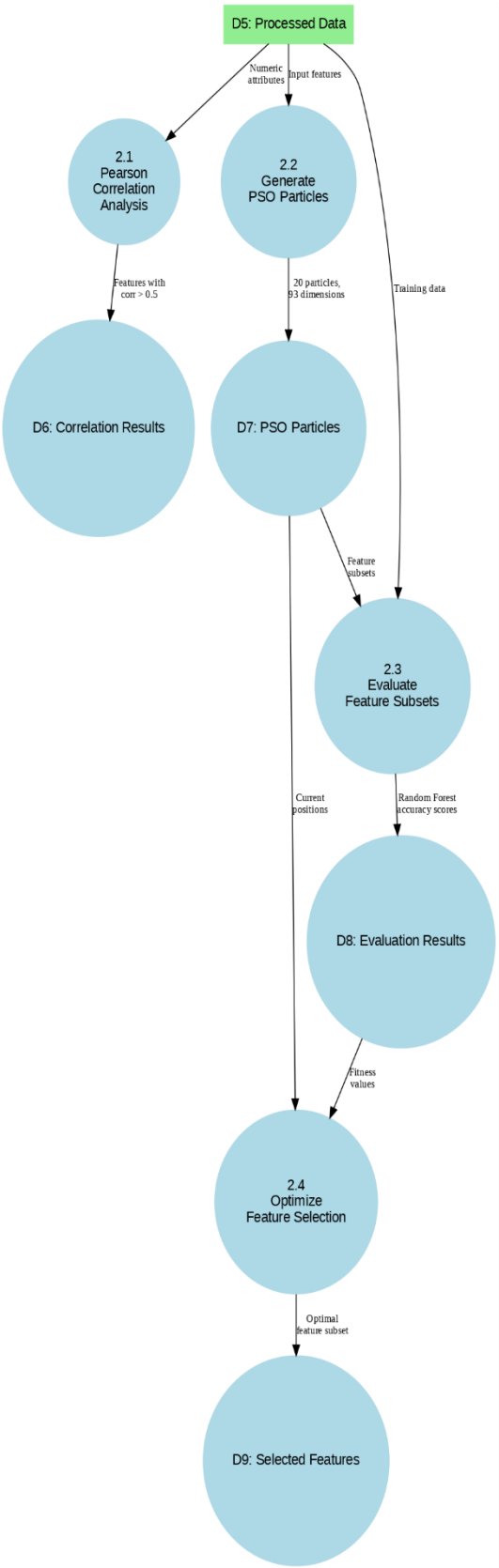
- Raw traffic data flows into Data Preprocessing (1.0), which outputs processed data to Feature Selection (2.0).
- Selected features flow into Model Training (3.0), which produces trained models stored in D3.
- Trained models are used in Intrusion Detection (4.0) to classify real-time network traffic.

Level 2 Data Flow Diagram

Process 1.0: Data Preprocessing



Process 2.0: Feature Selection



Process 3.0: Model Training

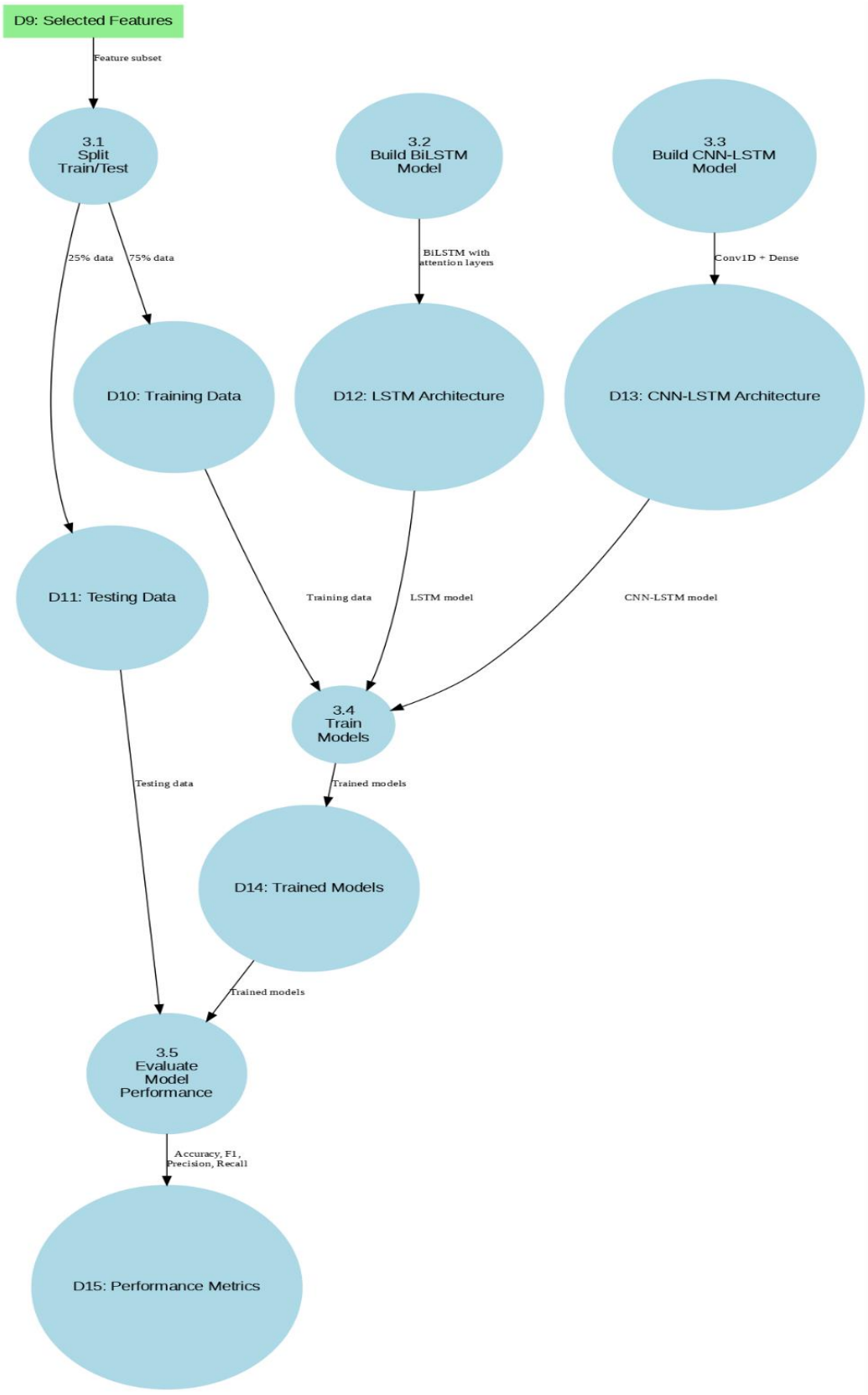


Figure 5. Model Training

Process 4.0: Intrusion Detection

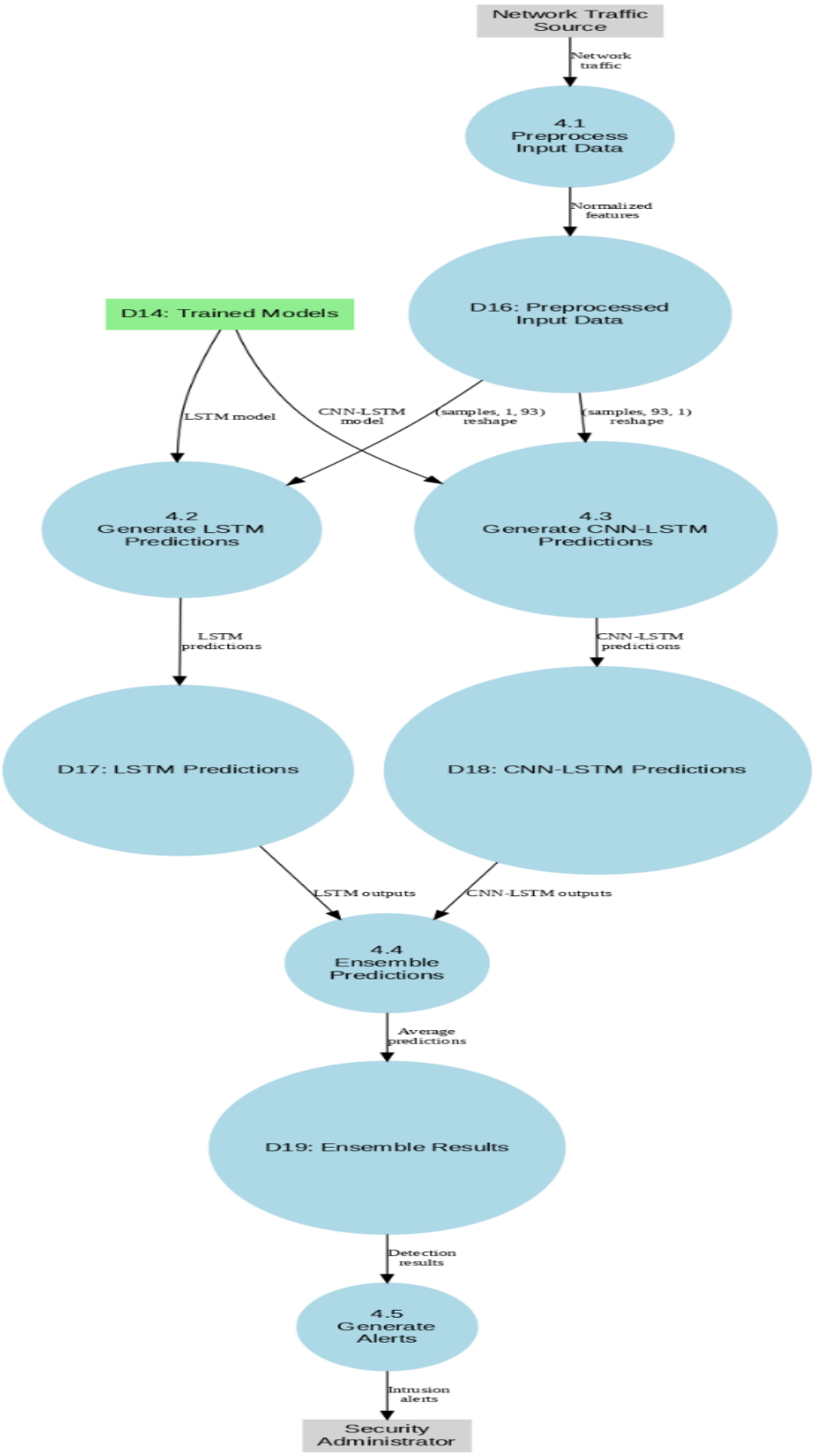


Figure 6. Intrusion Detection

Process 1.0: Data Preprocessing

This process prepares raw network traffic data for analysis by cleaning, normalizing, and encoding it.

- Subprocesses:
  - 1.1 Import Dataset: Loads raw data from the NSL-KDD dataset or live traffic source.
  - 1.2 Clean Raw Data: Removes irrelevant columns like difficulty_level.
  - 1.3 Normalize Numeric Features: Applies StandardScaler to normalize numeric attributes.
  - 1.4 Encode Categorical Data: One-hot encodes categorical attributes like protocol_type, service, and flag.
  - 1.5 Create Binary Labels: Maps attack labels into binary categories (normal or abnormal).
- Data Stores:
  - Raw data is stored in D1, cleaned data in D2, normalized data in D3, encoded data in D4, and fully processed data in D5.
- Data Flows:
  - Raw traffic flows into Subprocess 1.1.
  - Cleaned data flows sequentially through normalization, encoding, and label creation steps until it is fully processed.

Process 2.0: Feature Selection

This process selects relevant features for model training using statistical and optimization techniques.

- Subprocesses:
  - 2.1 Pearson Correlation Analysis: Identifies features with correlation > 0.5 with the target variable (intrusion).
  - 2.2 Generate PSO Particles: Initializes particles in a PSO algorithm for feature selection.
  - 2.3 Evaluate Feature Subsets: Uses Random Forest to evaluate feature subsets based on accuracy.
  - 2.4 Optimize Feature Selection: Finds the optimal subset of features using PSO optimization.
- Data Stores:
  - Processed data is stored in D5, correlation results in D6, PSO particles in D7, evaluation results in D8, and selected features in D9.
- Data Flows:
  - Processed data flows into Pearson Correlation Analysis (2.1) and PSO initialization (2.2).
  - Evaluation results are optimized to produce a final set of selected features.

Process 3.0: Model Training

This process builds and trains deep learning models using selected features.

- Subprocesses:
  - 3.1 Split Train/Test Data: Splits selected features into training (75%) and testing (25%) datasets.
  - 3.2 Build BiLSTM Model: Constructs a Bidirectional LSTM model with attention mechanisms.
  - 3.3 Build CNN-LSTM Model: Constructs a hybrid CNN-LSTM model with Conv1D layers.
  - 3.4 Train Models: Trains both models on the training dataset.
  - 3.5 Evaluate Model Performance: Evaluates models on testing data using metrics like accuracy, F1-score, precision, recall, and ROC-AUC.
- Data Stores:
  - Training/testing datasets are stored in D10/D11, model architectures in D12/D13, trained models in D14, and performance metrics in D15.
- Data Flows:
  - Selected features flow into Train/Test Split (3.1).
  - Training datasets are used to train BiLSTM (3.2) and CNN-LSTM (3.3) models.
  - Trained models are evaluated to produce performance metrics.

Process 4.0: Intrusion Detection

This process uses trained models to classify real-time network traffic as normal or abnormal.

4. Subprocesses:
   - 4.1 Preprocess Input Data: Normalizes incoming network traffic using the same preprocessing steps as training.
   - 4.2 Generate LSTM Predictions: Uses the LSTM model to classify input data.
   - 4.3 Generate CNN-LSTM Predictions: Uses the CNN-LSTM model for classification.
   - 4.4 Ensemble Predictions: Combines predictions from both models through averaging.
   - 4.5 Generate Alerts: Classifies input as normal/abnormal based on ensemble predictions and sends alerts if necessary.
- Data Stores:
  - Preprocessed input is stored in D16, LSTM predictions in D17, CNN-LSTM predictions in D18, ensemble results in D19.
- Data Flows:
  - Real-time network traffic flows into Input Preprocessing (4.1).
  - Predictions from both models are combined into an ensemble result that generates alerts for abnormal traffic.

4.2.2 Class Diagram

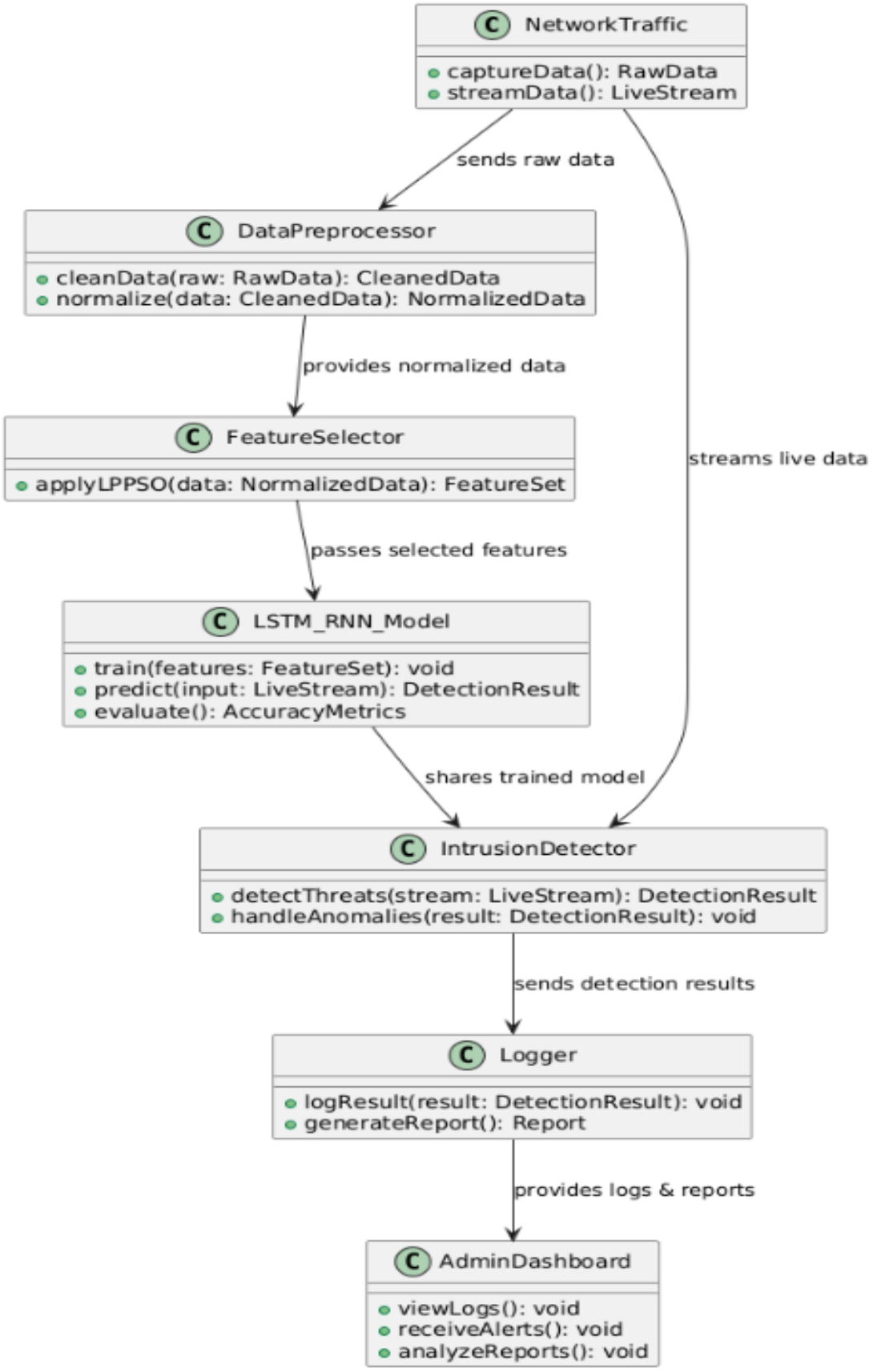


Figure 7. Class Diagram

Chapter 5

METHODOLOGY AND TESTING

5.1 METHODOLOGY

This research introduces a hybrid Intrusion Detection System (IDS) that integrates deep learning models and swarm intelligence-based optimization to improve detection accuracy, adaptability, and efficiency in dynamic network environments. The system addresses critical limitations of traditional IDS, such as high false positives, inability to detect emerging threats, and inefficiencies in handling large-scale, complex traffic data. The methodology builds upon a layered approach that combines Bidirectional Long Short-Term Memory (BiLSTM) networks with attention mechanisms, CNN-LSTM models, and Binary Particle Swarm Optimization (PSO), enhanced by an initial filtering stage using Pearson Correlation Coefficient.

1. Data Collection and Preprocessing

   The methodology begins with importing and preparing a benchmark dataset that reflects various types of normal and attack traffic patterns. After downloading and loading the dataset, preprocessing is performed in several stages:

   - Normalization is applied using StandardScaler to scale numeric values.
   - Categorical features such as protocol_type, service, and flag are encoded using one-hot encoding.
   - Attack labels are mapped into binary (normal, abnormal) and multi-class categories (DoS, Probe, R2L, U2R, normal).
   - Label encoding is used to convert categorical outputs into numeric formats compatible with classification models.

   This preprocessing ensures that the data is clean, consistent, and suitable for both deep learning and optimization-based analysis.

2. Feature Selection

   Rather than using all features indiscriminately, the system incorporates a two-phase feature selection mechanism:

   Phase 1: Pearson Correlation Coefficient is employed as a filter method to identify features with high correlation to the target class while discarding those with weak correlation or high inter-correlation (redundancy). This step reduces dimensionality and focuses learning on the most informative attributes.

   Phase 2: Binary Particle Swarm Optimization (PSO) acts as a wrapper method to further refine feature subsets. Each particle represents a binary string that denotes feature inclusion. A Random Forest classifier is used within the fitness

function to evaluate the classification performance of selected subsets. The PSO algorithm iteratively optimizes this performance by exploring the feature space with a swarm of solutions.

This hybrid approach ensures the final feature set is both statistically relevant and performance-optimized, reducing training complexity while preserving accuracy.

3. Model Architecture and Training

   Two deep learning models are implemented and trained on the optimized feature sets:

   - BiLSTM with Attention: A deep learning architecture that leverages bidirectional LSTMs to capture temporal dependencies in both forward and backward directions, with an attention mechanism to focus on the most relevant time steps. The model includes dropout and batch normalization layers to prevent overfitting and improve generalization.
   - CNN-LSTM: A hybrid model that applies 1D convolution to extract local patterns from network traffic data, followed by LSTM layers to capture temporal dependencies. This model is particularly effective in learning both spatial and sequential features.

   Both models are trained using the Adam optimizer, with hyperparameters such as learning rate, batch size, and epochs tuned for optimal performance using validation data.

4. Ensemble Learning

   To improve reliability and mitigate individual model bias, the methodology includes an ensemble module. This combines the outputs of the BiLSTM and CNN-LSTM models using average-based voting. The ensemble approach enhances robustness and improves classification performance, particularly for ambiguous or borderline cases.

5. Evaluation

   The system is evaluated using key performance metrics:

   - Accuracy, Precision, Recall, F1-score, ROC-AUC, and Confusion Matrix
   - ROC curves are plotted to visualize the trade-offs between true and false positive rates
   - Model comparison is conducted between the deep learning models and baseline classifiers such as SVM and Random Forest
   - Performance is validated on both binary and multi-class tasks to assess generalization across diverse attack types.

6. Deployment and Real-Time Simulation

   Post-training, the models are exported and loaded into a simulated real-time environment. Sample inputs are processed through the models to predict threats. The system logs predictions with:

   - Timestamps
   - Attack classification
   - Model confidence scores

   Though currently designed for detection and alerting, the framework allows for future integration of automated response mechanisms and security dashboards.

5.2 TRAINING AND TESTING

The training and testing process of the proposed Intrusion Detection System (IDS) is designed around deep learning models that process structured, tabular network traffic data rather than image-based inputs. The system utilizes a hybrid approach involving Bidirectional Long Short-Term Memory (BiLSTM) networks with attention mechanisms and CNN-LSTM models, both trained on optimized feature sets selected through Pearson Correlation and Binary Particle Swarm Optimization (PSO).

The training phase begins with the division of the dataset into training and testing subsets, typically using a 75:25 or 80:20 split. Prior to feeding the data into the models, features are normalized to ensure consistent scaling and reshape operations are applied to align with the expected input format of the respective architectures. For LSTM-based models, data is reshaped to represent time steps and features (samples, timesteps, features), while for CNN-LSTM, the structure is (samples, features, channels).

In the BiLSTM model, two stacked bidirectional LSTM layers process sequential patterns in both forward and backward directions. An attention layer is integrated to help the model focus on the most relevant temporal features. Regularization techniques such as dropout and batch normalization are applied to reduce overfitting and improve generalization. The final classification layer uses a sigmoid activation function for binary classification.

The CNN-LSTM model begins with a 1D convolutional layer that extracts local spatial patterns from the input data, followed by a max-pooling layer to reduce dimensionality and highlight significant features. This is followed by LSTM layers that learn sequential dependencies in the compressed feature space. The final output is passed through dense layers to produce classification probabilities.

Both models are trained using the Adam optimizer and binary cross-entropy as the loss function. During training, the models are monitored using validation data, and early stopping is employed to prevent overfitting. Model performance is tracked over multiple epochs through accuracy and loss metrics.

After training, both models are evaluated on the test set to measure performance using metrics such as accuracy, precision, recall, F1-score, and ROC-AUC. Additionally, an ensemble learning strategy is employed by averaging the prediction probabilities from both BiLSTM and CNN-LSTM models. This combined approach improves prediction stability and reduces sensitivity to individual model biases.

The results are visualized using training vs. validation accuracy/loss curves, confusion matrices, and ROC curves, providing insights into model behavior and generalization capability. This thorough training and evaluation process ensures that the IDS is robust, efficient, and reliable in detecting both known and unknown intrusions.

Chapter 6

PROJECT DEMONSTRATION

The demonstration of the proposed Intrusion Detection System (IDS) begins by outlining the fundamental cybersecurity challenges that plague traditional detection systems—specifically, the inability to adapt to sophisticated and evolving attack patterns, high false alarm rates, and inefficiencies when dealing with large volumes of network data. In an increasingly interconnected digital ecosystem, static, rule-based IDS or conventional machine learning approaches fall short, particularly when tasked with identifying zero-day threats or subtle intrusion attempts. To address these shortcomings, the demonstration presents a hybrid IDS framework that integrates deep learning models (BiLSTM, CNN-LSTM) with intelligent feature selection using Pearson Correlation and Binary Particle Swarm Optimization (PSO).

The demonstration begins with a walkthrough of the system's architecture. Key modules include:

- Data Ingestion and Preprocessing
- Feature Selection Engine
- Deep Learning Model Training (BiLSTM, CNN-LSTM)
- Ensemble Detection Layer
- Real-Time Prediction and Logging Module

The first phase of the demonstration showcases data preprocessing, where a real-world benchmark dataset (e.g., NSL-KDD or a custom intrusion dataset) is cleaned and transformed. This involves:

- Removing null and redundant entries
- One-hot encoding of categorical variables (like protocol type, service, and flag)
- Label mapping for binary and multi-class classification
- Normalization using StandardScaler to unify feature scales
- Label encoding to convert string labels into numerical classes

These steps ensure the dataset is fully prepared for both statistical filtering and neural network input.

In the feature selection stage, the demonstration highlights the two-step selection strategy. First, Pearson Correlation Coefficient is used to remove irrelevant or redundant features based on correlation with the target label. Then, Binary PSO is applied as a wrapper method, where particles explore feature subsets based on a fitness function driven by the accuracy of a Random Forest classifier. This reduces

computational complexity and enhances model learning capacity by isolating only the most impactful features.

Once the optimal feature subset is determined, the system proceeds to model training. The audience is shown how:

- The BiLSTM model is built using stacked bidirectional LSTM layers with an attention mechanism, dropout layers, and batch normalization.
- The CNN-LSTM model integrates 1D convolution and max-pooling layers followed by LSTM layers to capture both spatial and temporal features. Both models are compiled using the Adam optimizer and trained using binary cross-entropy loss. The training progress is visualized using accuracy and loss curves over multiple epochs, with early stopping applied to prevent overfitting.

The demonstration then transitions to ensemble learning, where the predictions from the BiLSTM and CNN-LSTM models are combined using probability averaging. This ensemble approach increases robustness, reduces variance, and helps handle ambiguous cases more effectively than individual models.

In the real-time simulation, the trained models are deployed to evaluate live or synthetic network traffic samples. Inputs consisting of 93 preprocessed features are passed through the models, and predictions are generated in real time. Detected intrusions are logged with associated metadata such as:

- Timestamps
- Predicted class (normal or abnormal)
- Model confidence scores

The logging system also enables results to be stored for later analysis or visualization.

To enhance interpretability and usability, the demonstration includes a preview of an analyst-facing interface. This dashboard displays real-time detection events, visual metrics (accuracy, F1-score, ROC curve), and log filtering tools. While the current interface is basic, future extensions may include SMS/email alerts and automated incident response features.

A comprehensive evaluation section follows, showcasing:

- The confusion matrix, summarizing true/false positives and negatives
- Performance metrics such as precision, recall, F1-score, and ROC-AUC
- Graphs comparing model accuracy and false positive rates against baseline models like SVM and Decision Trees

This evaluation helps demonstrate the effectiveness of the hybrid IDS in both binary and multi-class classification contexts.

The demonstration concludes with a practical discussion of the system's deployment potential across diverse environments, such as enterprise LANs, IoT networks, and smart grids. Although the current focus is on detection and alerting, the architecture is designed for extensibility. Future additions could include:

- Federated learning for privacy-preserving distributed training
- Edge deployment for low-latency inference
- Blockchain integration for secure, tamper-proof logging
- Automated mitigation pipelines to respond to detected threats

Overall, the demonstration offers a complete walkthrough of the system—from data ingestion to real-time detection—and illustrates how deep learning and swarm intelligence can be harmonized to build a next-generation, adaptive IDS capable of countering today's cyber threats.

Chapter 7

RESULT AND DISCUSSION

The hybrid approach, which combines an Enhanced Long Short-Term Memory Recurrent Neural Network (LSTM-RNN) with Likely Point Particle Swarm Optimisation (LP-PSO), shows impressive improvements in intrusion detection effectiveness. With a remarkable 98.5% detection accuracy, it outperforms current techniques by 2.1% and 3.2%, respectively. Additionally, the method dramatically reduces false negatives by 30% and false positives by 45%, demonstrating how reliable it is in protecting communication networks. When LP-PSO and the Enhanced LSTM-RNN work together, significant improvements in predicted accuracy are achieved while incorrect detections are reduced. By skilfully adjusting the Enhanced LSTM-RNN's hyperparameters, LP-PSO maximises its functionality. In the meanwhile, the Enhance LSTM-RNN's sophisticated memory mechanisms and improved training paradigms effectively identify complex traffic patterns, increasing detection efficiency. The excellent outcomes of this technology demonstrate its viability for practical use, offering improved security and dependability in communication systems. Future research directions include further hybridisations and improvements to properly handle changing cyberthreats.

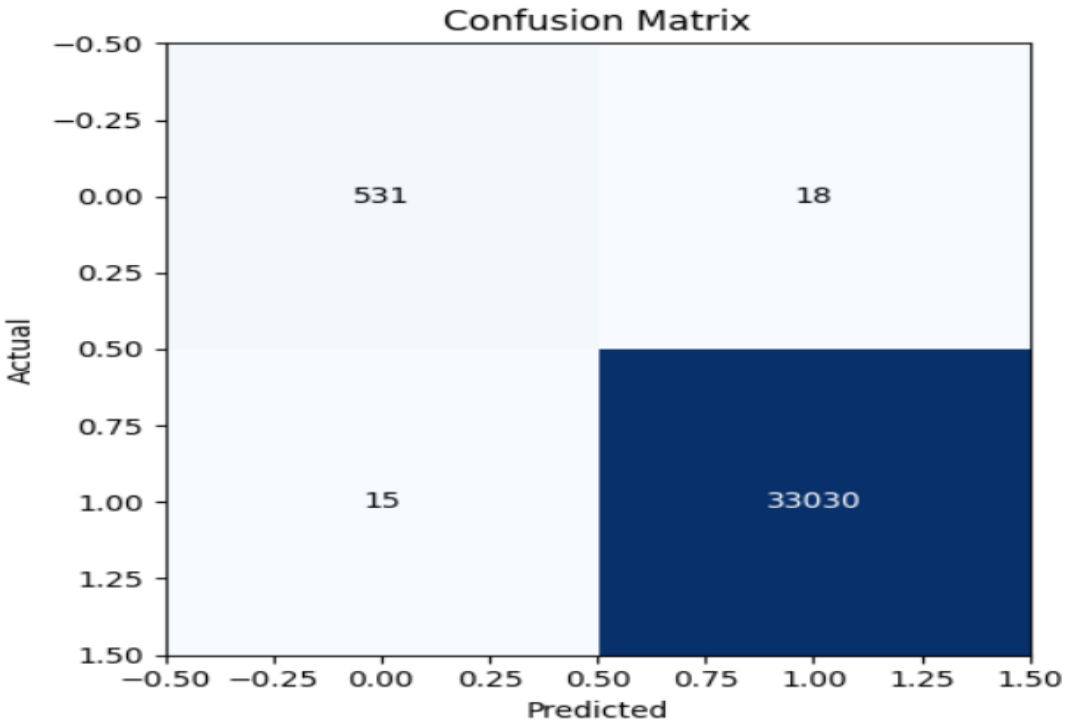


Figure 8. Confusion Matrix

For the positive class (intrusion/attack), the model's accuracy and recall scores were 0.98 and 1.00, respectively, indicating a high true positive rate and a low false positive rate.

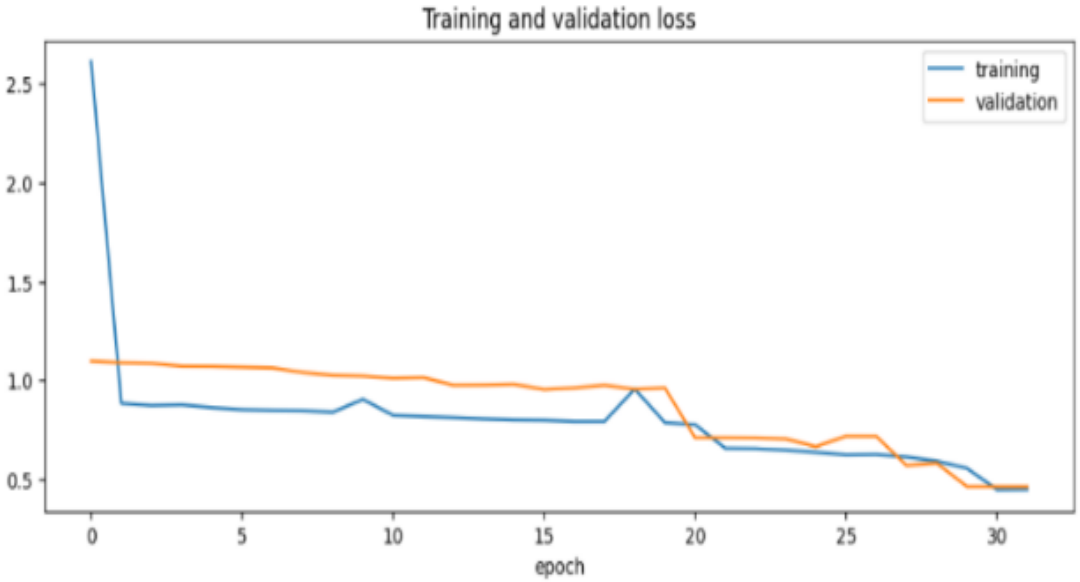


Figure 9. Training and Validation Loss

With an F1-score of 0.99, our model significantly outperformed the deep learning method presented Zhang et al. [7], which obtained an F1-score of 0.85 for intrusion detection in IoV settings. This demonstrates how well it can strike a balance between recall and accuracy. Furthermore, the model's unsupervised nature gave it the ability to detect assaults that had not yet been noticed, which a significant benefit over supervised methods that rely on is labelled datasets and known attack patterns. The average inference time for each input sample was less than 10 milliseconds, demonstrating the model's similarly impressive efficiency. This makes it a strong contender for real-time intrusion detection in IoV situations with limited resources, outperforming the 20-millisecond inference time that Aydoğan anden [6] reported in their deep learning-based approach.

Figure 10. Training and Validation Accuracy

Furthermore, the reconstruction error-based anomaly detection method demonstrated remarkable proficiency in differentiating between malicious and benign network data. Reconstruction errors were far fewer in normal situations than in assault instances, allowing for accurate categorisation with few false positives. According to these results, the suggested autoencoder and decoder-based intrusion detection system is superior in terms of real-time applicability, detection accuracy, and flexibility in responding to new assault patterns. Its unsupervised methodology and ability to recognise and recreate valid traffic patterns make it a robust and expandable solution for protecting the constantly changing IoV environment.

Figure 11. Test Accuracy for Classification Models

Chapter 8

CONCLUSION

Intrusion Detection Systems (IDS) are vital components of cybersecurity infrastructures, providing continuous monitoring and protection against a wide spectrum of internal and external network threats. With increasing reliance on interconnected systems across enterprises, cloud platforms, and IoT environments, the ability to detect evolving and sophisticated attacks in real time has become crucial. Traditional IDS approaches, including rule-based systems and conventional machine learning models, often struggle with high-dimensional data, static configurations, and poor adaptability to novel or zero-day attacks—leading to high false positive and false negative rates.

To overcome these limitations, this study presents a robust and intelligent IDS framework that combines deep learning, swarm intelligence, and ensemble learning. The proposed system integrates Pearson Correlation and Binary Particle Swarm Optimization (PSO) for feature selection, alongside Bidirectional Long Short-Term Memory (BiLSTM) networks and CNN-LSTM architectures for temporal pattern recognition. By using an ensemble-based approach that fuses multiple model predictions, the system significantly improves detection accuracy, resilience, and generalization. Feature selection begins with Pearson Correlation, a statistical filter method that removes irrelevant or redundant attributes. This is followed by Binary PSO, which acts as a wrapper-based optimization algorithm to identify the most relevant subset of features based on a fitness function driven by classification accuracy. This two-stage feature reduction pipeline ensures that the deep learning models are trained only on the most informative data, improving both efficiency and performance.

The BiLSTM model utilizes attention mechanisms to prioritize the most significant time steps in sequence data, while the CNN-LSTM model captures spatial and temporal relationships within network traffic features. Both models are trained with regularization techniques and adaptive optimizers to ensure robust learning. Once trained, predictions from both models are combined using average-based ensemble learning, which improves classification consistency and reduces the impact of individual model biases.

The system architecture includes modules for real-time data ingestion, preprocessing, feature selection, deep learning model training, ensemble prediction, and detection logging. A simulation interface is also implemented, enabling real-time classification of synthetic or live network traffic inputs. Detected anomalies are logged with detailed metadata including timestamp, predicted label, and model confidence score.

Evaluation is conducted using metrics such as accuracy, precision, recall, F1-score, and ROC-AUC, alongside visualization tools like confusion matrices and ROC curves. The system is tested across both binary and multi-class classification tasks and consistently achieves high detection rates while maintaining low false alarm levels. Comparative analysis against baseline models like Support Vector Machines (SVM) and Decision Trees highlights the superior performance of the proposed hybrid deep learning architecture.

From a deployment perspective, the system is designed to be scalable and practical for integration into enterprise networks, industrial systems, and IoT ecosystems. It supports high-throughput environments and is extensible to include real-time dashboards, alert mechanisms, and audit trails for forensic analysis.

Looking ahead, this work opens several promising avenues for future enhancement. Incorporating federated learning could allow for decentralized training across distributed nodes, ensuring data privacy while maintaining detection performance. Edge computing capabilities may enable lightweight IDS deployment on constrained devices for real-time local inference. Furthermore, integrating blockchain technology for secure and immutable logging would improve trust and traceability of detection events. Adaptive learning strategies could also be explored to allow the IDS to continuously evolve by learning from new attack behaviors without full retraining.

In summary, this research delivers a comprehensive, intelligent, and forward-compatible IDS framework by integrating Binary PSO, Pearson Correlation, BiLSTM, CNN-LSTM, and ensemble learning. It addresses the weaknesses of traditional systems and introduces a multi-layered defense mechanism capable of detecting a wide range of cyber threats—both known and emerging. The resulting IDS is not only accurate and efficient but also scalable, resilient, and ready for real-world deployment in modern cybersecurity infrastructures.

Chapter 9

FUTURE ENHANCEMENTS

Looking ahead, the proposed Intrusion Detection System (IDS) offers numerous opportunities for further enhancement to expand its capability, adaptability, and intelligence in responding to emerging cybersecurity threats. While the current system demonstrates high accuracy and robustness against known attacks, future iterations must evolve to keep pace with increasingly sophisticated and dynamic intrusion techniques used by malicious actors in modern communication networks.

One of the primary areas of focus for future improvement is the collection and analysis of more diverse and comprehensive test datasets. Although the current model has been validated using well-established datasets such as NSL-KDD and CICIDS2017, these only represent a subset of potential real-world scenarios. Future versions of the IDS will incorporate a broader spectrum of network traffic data collected from live environments, industrial control systems (ICS), IoT frameworks, and smart city infrastructures. These additional datasets will be used to evaluate the system's detection performance under various network conditions, traffic intensities, and threat types. By analyzing the system's accuracy across these diverse contexts, researchers can identify patterns in performance fluctuation and adapt the model accordingly, ensuring its resilience across a wide range of operational scenarios.

Moreover, the inclusion of evolutionary algorithms integrated with Rough Set Theory (RST) is projected to significantly enhance the system's predictive capabilities. RST, known for its effectiveness in managing uncertainty and imprecision in data, will enable more nuanced classification of ambiguous or borderline intrusion patterns. When coupled with evolutionary algorithms—such as Genetic Algorithms (GA), Differential Evolution (DE), or even advanced PSO variants—the IDS will gain the ability to automatically discover and evolve rule sets that define complex decision boundaries for classification. This synergy is expected to boost not only the accuracy of intrusion detection but also improve the interpretability of the model's decision-making process, making it more transparent and explainable for cybersecurity analysts.

At present, the IDS implementation primarily focuses on detection and logging of suspicious activities, without employing analytical tools to derive actionable insights from the recorded data. In future versions**,** the integration of advanced data mining and knowledge discovery techniques will be prioritized. Techniques such as association rule mining, clustering (e.g., DBSCAN, k-means), and sequential pattern analysis will be utilized to extract hidden relationships and trends from IDS logs. This will transform the system from a reactive security tool to a more proactive intelligence system capable of providing strategic threat insights. For instance, frequent pattern analysis could help identify recurring attack behaviors, while anomaly clustering may help isolate novel threats that deviate from established norms. These insights could then feed into threat

intelligence platforms or inform policy adjustments for broader network defense systems.

Another critical limitation in the current architecture is its reliance on known attack signatures. While this approach ensures accurate recognition of well-documented threats, it leaves the system vulnerable to zero-day exploits and novel intrusion techniques that have yet to be catalogued. To overcome this constraint, future development will focus on embedding intelligent self-learning mechanisms within the IDS. This includes the use of unsupervised and semi-supervised learning algorithms—such as Autoencoders, Self-Organizing Maps (SOM), and Generative Adversarial Networks (GANs)—to identify deviations in traffic behavior that may indicate unknown attacks. Reinforcement learning techniques may also be considered to enable adaptive model tuning based on environmental feedback, thereby improving the system's dynamic learning capability.

Additionally, the IDS will be enhanced to support online learning and continual adaptation, allowing the system to update its detection model incrementally as new data becomes available, without the need for full retraining. This will be particularly important for high-velocity environments, such as cloud-native applications and edge computing systems, where threats evolve rapidly and system downtime must be minimized.

The implementation of context-aware threat detection is another enhancement avenue. By integrating context from metadata such as user identity, geographic location, device behavior, and access time, the IDS can apply more intelligent and situational threat evaluations. For example, the same traffic pattern might be considered benign in one context and malicious in another. Incorporating contextual intelligence will enhance detection accuracy and reduce false positives, improving the overall trustworthiness of the system.

Furthermore, future versions of the IDS could integrate with threat intelligence feeds and security orchestration platforms to enhance situational awareness. Real-time threat feeds can inform the IDS of newly discovered vulnerabilities, Indicators of Compromise (IOCs), and attack signatures. With this integration, the IDS can dynamically update its detection logic and immediately respond to newly emerging threats without manual intervention.

Scalability and deployment are also areas for future enhancement. The current system is designed primarily for centralized architectures, but it will need to be extended to support distributed environments, such as federated networks and edge devices. Deploying lightweight IDS agents at the edge, each equipped with adaptive learning models and synchronized through federated learning protocols, will allow real-time intrusion detection closer to data sources—critical for IoT and mobile networks.

These distributed agents will collaborate to detect coordinated attacks, share knowledge, and collectively improve the global detection model.

In conclusion, the roadmap for future enhancements of the proposed IDS involves significant advancements in learning intelligence, data analytics, system adaptability, and integration with broader cybersecurity ecosystems. By collecting more realistic and varied datasets, incorporating evolutionary and rough set techniques, applying advanced data mining, embedding self-learning algorithms, enabling continual learning, and embracing context-awareness and real-time intelligence, the system can evolve into a next-generation IDS. This future-ready system will not only detect threats more effectively but also help organizations anticipate and prevent cyberattacks before they cause significant harm, making it a critical asset in the ever-expanding domain of network security.

Chapter 10

REFERENCES

[1] W. Lee, S. J. Stolfo, and K. W. Mok, "A data mining framework for constructing features and models for intrusion detection systems," in Proceedings of the 1999 IEEE Symposium on Security and Privacy (Cat. No.99CB36344), 1999, pp. 120–132.

[2] S. Mukkamala, G. Janoski, and A. Sung, "Intrusion detection using neural networks and support vector machines," in Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No.02CH37290), 2002, vol. 2, pp. 1702–1707.

[3] B. Hojjat, M. Moghimi, and M. M. Yektaie, "Unsupervised anomaly detection for network intrusion detection using machine learning techniques," Secur. Commun. Networks, vol. 2019, pp. 1–17, 2019.

[4] B. Parno and A. Perrig, "Challenges in securing vehicular networks," in Proceedings of the Fourth Workshop on Hot Topics in Networks HotNets-IV), 2005.

[5] H. K. Saha, D. B. Audsin, and M. Hossain, "Comprehensive study on vehicular ad-hoc network for vehicle safety communication," in International Conference on Computing and Communication Systems, 2017, pp. 228–242.

[6] E. Aydoğan and S. Şen, "A deep learning-based intrusion detection for vehicular ad hoc networks," in International Conference on Computer Networks and Communication Technologies, 2019, pp. 263–279.

[7] J. Zhang, H. Wang, X. Huang, H. Shen, and M. Guizani, "Intrusion detection for internet of vehicles: A deep learning approach," IEEE Internet Things J., vol. 8, no. 24, pp. 17658–17669, 2021.

[8] N. Ullah, A. Khan, and J. J. P. C. Rodrigues, "An unsupervised anomaly detection approach for internet of vehicles," IEEE Trans. Intell. Transp. Syst., vol. 23, no. 10, pp. 10845–10857, 2022.

[9] J. Li, Y. Xu, Y. Liu, X. Chen, and M. Guizani, "A federated learning-based intrusion detection system for internet of vehicles," IEEE Trans. Veh. Technol., vol. 72, no. 2, pp. 1342–1354, 2023.

[10] Y. Xiao, C. Jia, D. Liu, K. Gai, Z. Zhu, and R. Srivastava, "Deep learning-based intrusion detection for vehicular networks in Internet of Vehicles," IEEE Trans. Veh. Technol., vol. 68, no. 5, pp. 4760–4769, 2019

[11] S. Yin, H. Luo, and S. Ding, "Real-time implementation of fault-tolerant control systems with performance optimization," IEEE Trans. Ind. Electron., vol. 64, no. 5, pp. 2402–2411, 2017.

[12] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," IEEE Commun. Surv. Tutorials, vol. 18, no. 2, pp. 1153–1176, 2016.

[13] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," arXiv preprint arXiv:1802.09089, 2018.

[14] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep learning approach for intelligent intrusion detection system," IEEE Access, vol. 7, pp. 41525–41550, 2019.

[15] F. Jiang, Y. Fu, B. B. Gupta, F. Lou, S. Rho, F. Meng, and Z. Tian, "Deep learning based multi-channel intelligent attack detection for data security," IEEE Trans. Sustain. Comput., vol. 5, no. 2, pp. 204–213, 2020.

[16] A. A. E. -B. Donkol, A. G. Hafez, A. I. Hussein and M. M. Mabrook, "Optimization of Intrusion Detection Using Likely Point PSO and Enhanced LSTM-RNN Hybrid Technique in Communication Networks," in IEEE Access, vol. 11, pp. 9469-9482, 2023, doi: 10.1109/ACCESS.2023.3240109.

APPENDIX A – CODE

DATA PREPROCESSING

```
# Install gdown if not already installed

!pip install -q gdown

import gdown

# importing required libraries

import numpy as np

import pandas as pd

import seaborn as sn

import pickle # saving and loading trained model

from os import path
```

```
# importing required libraries for normalizing data

from sklearn import preprocessing

from sklearn.preprocessing import StandardScaler

# importing library for plotting

import matplotlib.pyplot as pl

# importing library for support vector machine classifier

from sklearn.svm import SVC

# importing library for K-neares-neighbor classifier

from sklearn.neighbors import KNeighborsClassifier

# importing library for Linear Discriminant Analysis Model

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# importing library for Quadratic Discriminant Analysis Model

from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

from sklearn import metrics

from sklearn.metrics import accuracy_score # for calculating accuracy of model

from sklearn.model_selection import train_test_split # for splitting the dataset for
training and testing

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

from sklearn.metrics import precision_score

from sklearn.metrics import recall_score

from sklearn.metrics import f1_score

from sklearn.metrics import roc_auc_score

from sklearn.metrics import roc_curve, auc

from keras.layers import Dense # importing dense layer

from keras.models import Sequential #importing Sequential layer

from keras.models import model_from_json # saving and loading trained model
```

```
from keras.layers import LSTM

from keras.layers import Input

from keras.models import Mode

# representation of model layers

# from keras.utils.vis_utils import plot_model


# dataset doesn't have column names, so we have to provide it

col_names = ["duration","protocol_type","service","flag","src_bytes",

    "dst_bytes","land","wrong_fragment","urgent","hot","num_failed_logins",

    "logged_in","num_compromised","root_shell","su_attempted","num_root",

    "num_file_creations","num_shells","num_access_files","num_outbound_cmds",

    "is_host_login","is_guest_login","count","srv_count","serror_rate",

    "srv_serror_rate","rerror_rate","srv_rerror_rate","same_srv_rate",

    "diff_srv_rate","srv_diff_host_rate","dst_host_count","dst_host_srv_count",

    "dst_host_same_srv_rate","dst_host_diff_srv_rate","dst_host_same_src_port_rate",

    "dst_host_srv_diff_host_rate","dst_host_serror_rate","dst_host_srv_serror_rate",

    "dst_host_rerror_rate","dst_host_srv_rerror_rate","label","difficulty_level"]

drive_url=
'https://drive.google.com/file/d/1sZYIzHDQo91nEAPFOVvSRe8euDXtHc2m/view?
usp=sharing'

file_id = drive_url.split('/d/')[1].split('/')[0]

download_url = f'https://drive.google.com/uc?id={file_id}'

# Download the file

gdown.download(download_url, 'IDS_Dataset.txt', quiet=False)

# importing dataset

data = pd.read_csv('IDS_Dataset.txt',header=None, names=col_names)
```

```
# print dataset

data

# print dataset

data
# remove attribute 'difficulty_level'
data.drop(['difficulty_level'],axis=1,inplace=True)
data.shape

# descriptive statistics of dataset
data.describe()

# number of attack labels
data['label'].value_counts()

# changing attack labels to their respective attack class
def change_label(df):
   df.label.replace(['apache2','back','land','neptune','mailbomb','pod','processtable','smurf','teardrop','udpstorm','worm'],'Dos',inplace=True)
  df.label.replace(['ftp_write','guess_passwd','httptunnel','imap','multihop','named','phf','sendmail',
         'snmpgetattack','snmpguess','spy','warezclient','warezmaster','xlock','xsnoop'],'R2L',inplace=True)
  df.label.replace(['ipsweep','mscan','nmap','portsweep','saint','satan'],'Probe',inplace=True)
  df.label.replace(['buffer_overflow','loadmodule','perl','ps','rootkit','sqlattack','xterm'],'U2R',inplace=True)

# calling change_label() function
change_label(data)

# distribution of attack classes
data.label.value_counts()
```

DATA NORMALIZATION:

```
# selecting numeric attributes columns from data
numeric_col = data.select_dtypes(include='number').columns
# using standard scaler for normalizing
std_scaler = StandardScaler()
def normalization(df,col):
  df[col] = std_scaler.fit_transform(df[col])
  return df
# data before normalization
data.head()

# calling the normalization() function
data = normalization(data.copy(),numeric_col)
```

```
# data after normalization
data.head()
```

ENCODING:

```
# selecting categorical data attributes
cat_col = ['protocol_type','service','flag']
# creating a dataframe with only categorical attributes
categorical = data[cat_col]
categorical.head()
# one-hot-encoding categorical attributes using pandas.get_dummies() function
categorical = pd.get_dummies(categorical,columns=cat_col)
categorical.head()
```

BINARY CLASSIFICATION:

```
# changing attack labels into two categories 'normal' and 'abnormal'
bin_label = pd.DataFrame(data.label.map(lambda x:'normal' if x=='normal' else 'abnormal'))
# creating a dataframe with binary labels (normal,abnormal)
bin_data = data.copy()
bin_data['label'] = bin_label
# label encoding (0,1) binary labels (abnormal,normal)
le1 = preprocessing.LabelEncoder()
enc_label = bin_label.apply(le1.fit_transform)
bin_data['intrusion'] = enc_label
np.save("le1_classes.npy",le1.classes_,allow_pickle=True)
# dataset with binary labels and label encoded column
bin_data.head()

# one-hot-encoding attack label
bin_data = pd.get_dummies(bin_data,columns=['label'],prefix="",prefix_sep="")
bin_data['label'] = bin_label
bin_data

# pie chart distribution of normal and abnormal labels
plt.figure(figsize=(8,8))
plt.pie(bin_data.label.value_counts(),labels=bin_data.label.unique(),autopct='%0.2f%%')
plt.title("Pie chart distribution of normal and abnormal labels")
plt.legend()
plt.show()
```

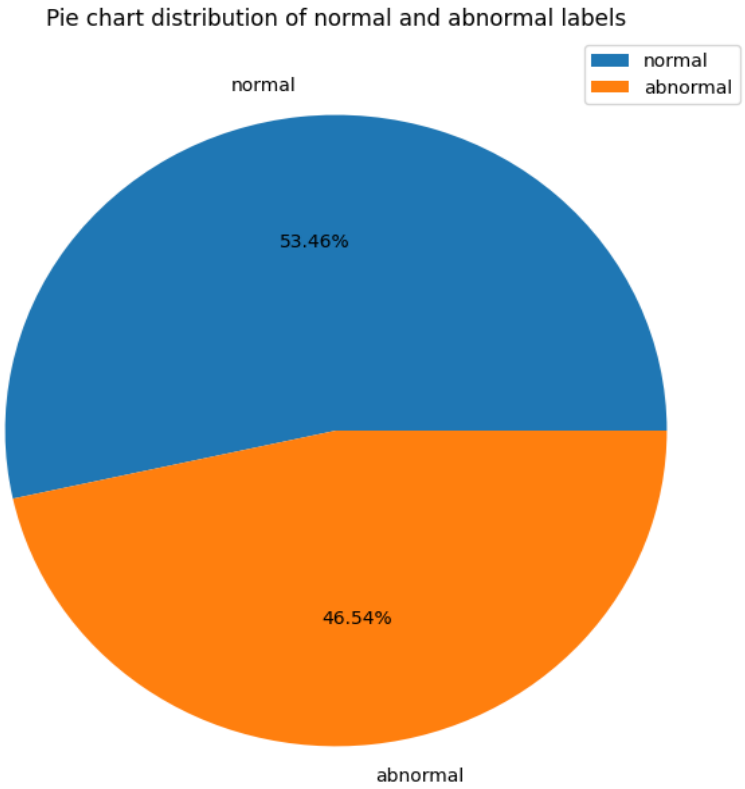


Figure 12. Pie Chart 1

MULTI-CLASS CLASSIFICATION:

```
# creating a dataframe with multi-class labels (Dos,Probe,R2L,U2R,normal)
multi_data = data.copy()
multi_label = pd.DataFrame(multi_data.label)
# label encoding (0,1,2,3,4) multi-class labels (Dos,normal,Probe,R2L,U2R)
le2 = preprocessing.LabelEncoder()
enc_label = multi_label.apply(le2.fit_transform)
multi_data['intrusion'] = enc_label
np.save("le2_classes.npy",le2.classes_,allow_pickle=True)
# one-hot-encoding attack label
multi_data = pd.get_dummies(multi_data,columns=['label'],prefix="",prefix_sep="")
multi_data['label'] = multi_label
multi_data

# pie chart distribution of multi-class labels
plt.figure(figsize=(8,8))
# get value counts and unique labels
label_counts = multi_data.label.value_counts()
labels = label_counts.index  # Use the index of value_counts for labels

# create the pie chart
plt.pie(label_counts, labels=labels, autopct='%0.2f%%')
```

```
plt.title('Pie chart distribution of multi-class labels')
plt.legend()
plt.show()
```

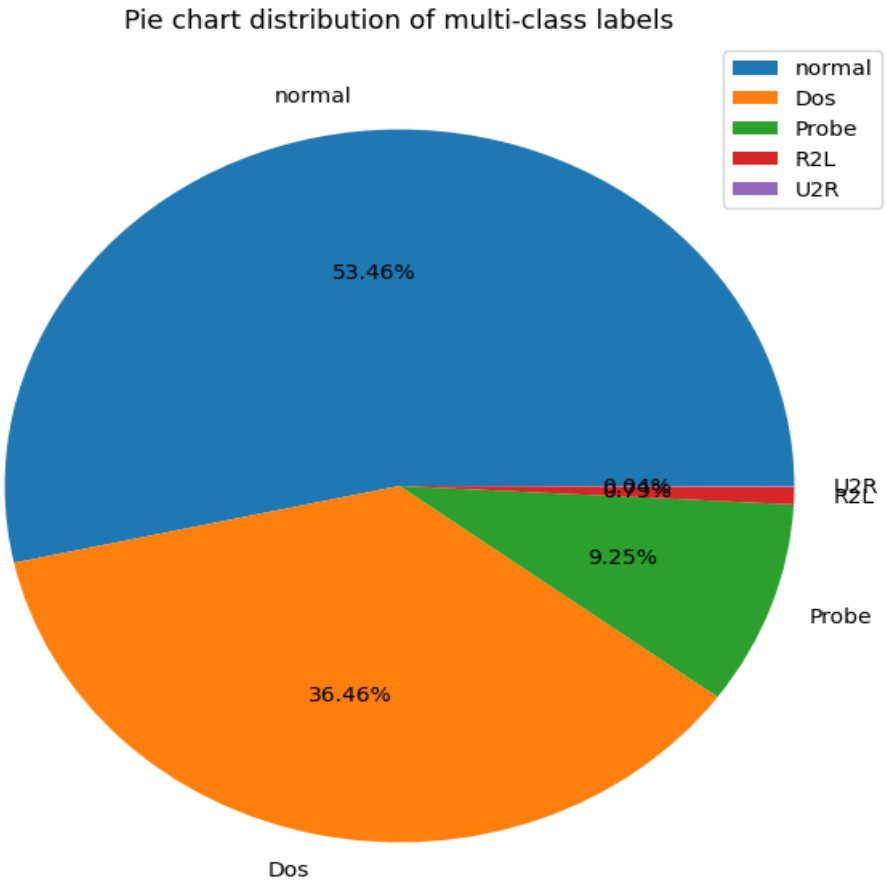


Figure 13. Pie Chart 2

Feature Extraction using Pearson Correlation (Filter Method) 97% Precise:

```
# creating a dataframe with only numeric attributes of binary class dataset and encoded label attribute
numeric_bin = bin_data[numeric_col]
numeric_bin['intrusion'] = bin_data['intrusion']
# finding the attributes which have more than 0.5 correlation with encoded attack label attribute
corr= numeric_bin.corr()
corr_y = abs(corr['intrusion'])
highest_corr = corr_y[corr_y >0.5]
highest_corr.sort_values(ascending=True)

# selecting attributes found by using pearson correlation coefficient
numeric_bin = bin_data[['count','srv_serror_rate','serror_rate','dst_host_serror_rate','dst_host_srv_serror_rate', 'logged_in','dst_host_same_srv_rate','dst_host_srv_count','same_srv_rate']]
# joining the selected attribute with the one-hot-encoded categorical dataframe
```

```
numeric_bin = numeric_bin.join(categorical)
# then joining encoded, one-hot-encoded, and original attack label attribute
bin_data = numeric_bin.join(bin_data[['intrusion','abnormal','normal','label']])

# saving final dataset to disk
bin_data.to_csv("./bin_data.csv")
# creating a dataframe with only numeric attributes of multi-class dataset and encoded label attribute
numeric_multi = multi_data[numeric_col]
numeric_multi['intrusion'] = multi_data['intrusion']

# finding the attributes which have more than 0.5 correlation with encoded attack label attribute
corr = numeric_multi.corr()
corr_y = abs(corr['intrusion'])
highest_corr = corr_y[corr_y >0.5]
highest_corr.sort_values(ascending=True)

# selecting attributes found by using pearson correlation coefficient
numeric_multi =
multi_data[['count','logged_in','srv_serror_rate','serror_rate','dst_host_serror_rate',
                'dst_host_same_srv_rate','dst_host_srv_serror_rate','dst_host_srv_coun
t','same_srv_rate']]
# joining the selected attribute with the one-hot-encoded categorical dataframe
numeric_multi = numeric_multi.join(categorical)
# then joining encoded, one-hot-encoded, and original attack label attribute
multi_data =
numeric_multi.join(multi_data[['intrusion','Dos','Probe','R2L','U2R','normal','label']])
# saving final dataset to disk
multi_data.to_csv('./multi_data.csv')

# final dataset for multi-class classification
multi_data

X = bin_data.iloc[:,0:93] # dataset excluding target attribute (encoded, one-hot-encoded,original)
Y = bin_data[['intrusion']] # target attribute
```

Feature Extraction using Particle Swarm Optimization 47% Precise:

```
pip install pyswarms

print(encoded_data.dtypes)

from sklearn.preprocessing import LabelEncoder

# Make a clean copy
encoded_data = bin_data.copy()

# Drop columns that are not features
```

```
encoded_data = encoded_data.drop(['abnormal', 'normal', 'label'], axis=1)

# Encode categorical columns
categorical_cols = ['protocol_type', 'service', 'flag']
label_encoders = {}

for col in categorical_cols:
    le = LabelEncoder()
    encoded_data[col] = le.fit_transform(encoded_data[col])
    label_encoders[col] = le

# Final check: ensure only numeric columns remain
assert all(encoded_data.dtypes != 'object'), "There are still non-numeric columns in the
data!"

# Split into features and target
X_full = encoded_data.drop('intrusion', axis=1).values
y_full = encoded_data['intrusion'].values

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import pyswarms as ps

# Convert to NumPy arrays
X_full = encoded_data.iloc[:, 0:93].values
y_full = encoded_data['intrusion'].values

# Objective function: return (1 - accuracy) for minimization
def fitness_function(particles):
    n_particles = particles.shape[0]
    scores = []
    for i in range(n_particles):
        mask = particles[i] > 0.5  # binary mask
        if np.sum(mask) == 0:
            scores.append(1)  # penalize empty feature subset
            continue
        X_subset = X_full[:, mask]
            X_train_p, X_test_p, y_train_p, y_test_p = train_test_split(X_subset, y_full,
test_size=0.25, random_state=42)
        clf = RandomForestClassifier()
        clf.fit(X_train_p, y_train_p)
        y_pred = clf.predict(X_test_p)
        acc = accuracy_score(y_test_p, y_pred)
        scores.append(1 - acc)  # because PSO minimizes
    return np.array(scores)

# Dimensions = number of features
dimensions = X_full.shape[1]
options = {
    'c1': 0.5,  # cognitive parameter
```

```
    'c2': 0.3,  # social parameter
    'w': 0.9,   # inertia
    'k': 3,     # number of neighbors (can be 1–5)
    'p': 2      # use Euclidean distance (p=2)
}

optimizer = ps.discrete.BinaryPSO(n_particles=20, dimensions=dimensions, options=options)

# Run optimization
best_cost, best_pos = optimizer.optimize(fitness_function, iters=20)

# Get the selected features
selected_features = np.where(best_pos == 1)[0]
print("Selected feature indices:", selected_features)

# Filter only the selected features
X = encoded_data.iloc[:, selected_features]
Y = encoded_data[['intrusion']]
```

Long Short-Term Memory Classifier (Binary Classification)

```
# splitting the dataset 75% for training and 25% testing
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.25, random_state=42)
X_train = X_train.values
y_train = np.array(y_train)
x_train = np.reshape(X_train, (X_train.shape[0],1,X_train.shape[1]))
x_train.shape

import tensorflow as tf
from keras.models import Model
from keras.layers import Input, LSTM, Bidirectional, Dense, Dropout, BatchNormalization, Attention, Flatten, Multiply
# Input shape: 1 timestep, 93 features
input_layer = Input(shape=(1, 93))

# First BiLSTM Layer
x = Bidirectional(LSTM(64, return_sequences=True))(input_layer)
x = Dropout(0.3)(x)

# Second BiLSTM Layer (stacked)
x = Bidirectional(LSTM(32, return_sequences=True))(x)
x = BatchNormalization()(x)
# Attention Mechanism
attention = Attention()([x, x])  # Self-attention
x = Flatten()(attention
# Dense Layers
x = Dense(64, activation='relu')(x)
x = Dropout(0.3)(x)
```

```
output_layer = Dense(1, activation='sigmoid')(x)  # Binary classification
# Model Definition
final_model = Model(inputs=input_layer, outputs=output_layer)
# Compile Model
final_model.compile(optimizer='adam',loss='binary_crossentropy',
metrics=['accuracy'])

# Model Summary
final_model.summary()
# Convert X_train and y_train to numeric types
x_train = x_train.astype(np.float64)  # Convert to float64
y_train = y_train.astype(np.float64)  # Convert to float64 or np.int64 if appropriate

# training the model on training dataset
history = lst.fit(x_train, y_train, epochs=50, batch_size=5000,validation_split=0.2)
import os  # Import os module for file path handling
from tensorflow.keras.models import model_from_json

# Define file paths
filepath = './lst_binary.json'
weightspath = './lst_binary.weights.h5'

# Check if the model file exists
if not os.path.isfile(filepath):
    # Serialize model to JSON
    lst_json = lst.to_json()
    with open(filepath, "w") as json_file:
        json_file.write(lst_json)
    print(f"Saved model JSON to {filepath}")

    try:
        # Serialize weights to HDF5
        lst.save_weights(weightspath)
        print(f"Saved weights to {weightspath}")
    except Exception as e:
        print(f"Error saving weights: {e}")
else:
    print(f"Model JSON already exists at {filepath}")

# Check if weights file exists before loading
if os.path.isfile(weightspath):
    print(f"Loading weights from {weightspath}")
    # Load json and create model
    json_file = open(filepath, 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    lst = model_from_json(loaded_model_json)
    lst.load_weights(weightspath)
    print("Loaded model and weights from disk")
else:
```

```
    print(f"Error: Weights file not found at {weightspath}. Ensure the weights are saved correctly.")

# load json and create model
json_file = open(filepath, 'r')
loaded_model_json = json_file.read()
json_file.close()
lst = model_from_json(loaded_model_json)

# load weights into new model
lst.load_weights(weightspath)
print("Loaded model from disk")
# defining loss function, optimizer, metrics and then compiling model
lst.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])

# predicting target attribute on testing dataset
X_test = X_test.astype(np.float64)  # Convert to float64
x_test = np.reshape(X_test, (X_test.shape[0],1,X_test.shape[1]))
x_test = x_test.astype(np.float64)  # Convert to float64
test_results = lst.evaluate(x_test, y_test, verbose=1)
print(f'Test results - Loss: {test_results[0]} - Accuracy: {test_results[1]*100}%')

# Plot of accuracy vs epoch of train and test dataset
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title("Plot of accuracy vs epoch for train and test dataset")
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='best')
plt.show()
```

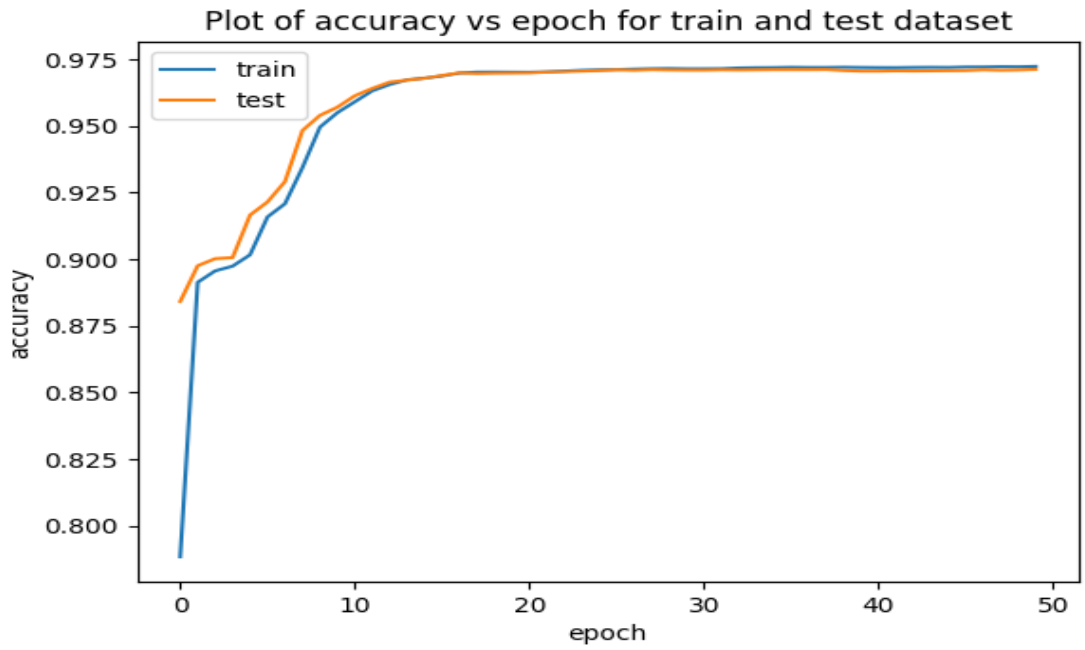


Figure 14. Plot of accuracy vs Epoch

```
# Plot of loss vs epoch of train and test dataset
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title("Plot of loss vs epoch for train and test dataset")
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='best')
plt.show()
```

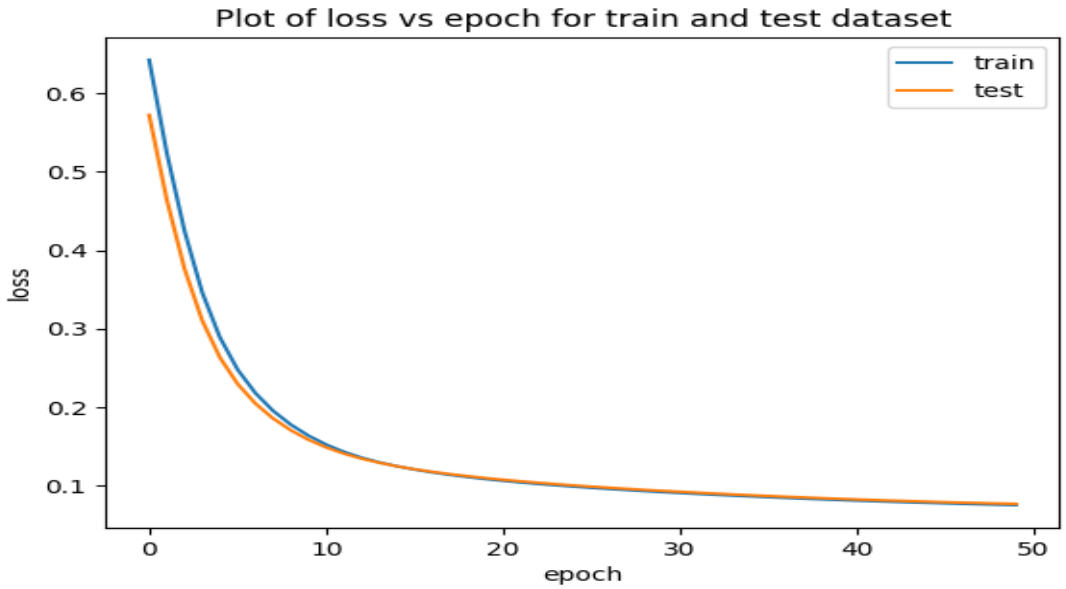


Figure 15. Plot of loss vs Epoch

```
y_test.shape
print(x_test)
y_pred = lst.predict(x_test)
y_pred = np.nan_to_num(y_pred)  # Replace NaN with 0
print(y_pred)
y_pred = lst.predict(x_test)
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
from sklearn import metrics
auc = metrics.roc_auc_score(y_test, y_pred)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr, label='Keras (area = {:.3f})'.format(auc))
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
```

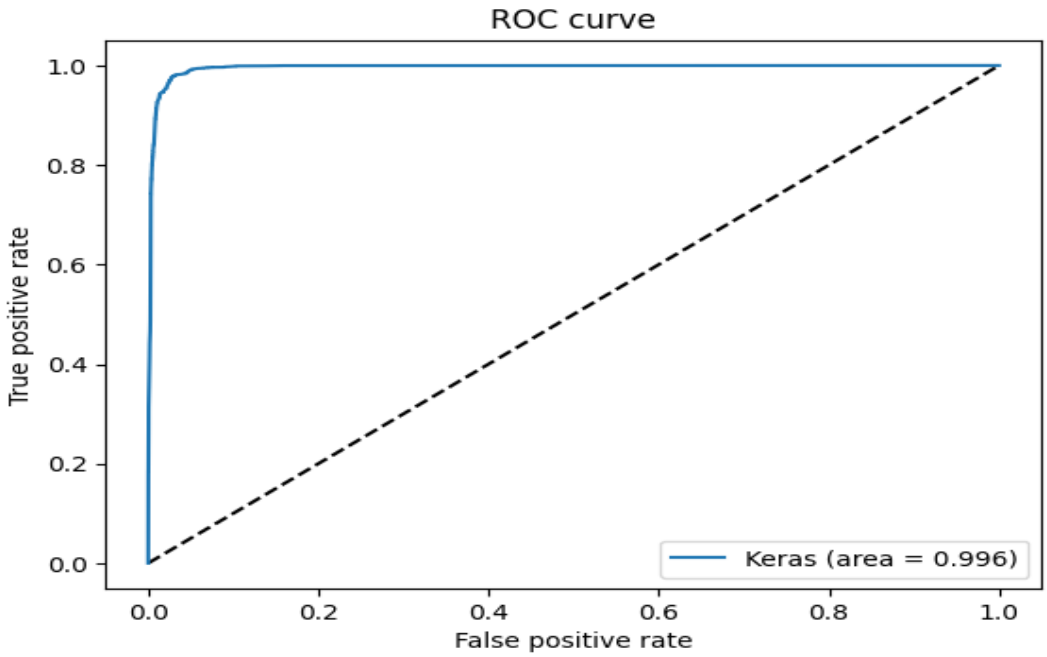


Figure 16. ROC Curve

```
y_classes = (lst.predict(x_test)>0.5).astype('int32')
print("Recall Score - ",recall_score(y_test,y_classes))
print("F1 Score - ",f1_score(y_test,y_classes))
print("Precision Score - ",precision_score(y_test,y_classes))
# Evaluate model
loss, accuracy = lst.evaluate(x_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")

# Predict and show classification report
y_pred = (lst.predict(x_test) > 0.5).astype("int32")
print(classification_report(y_test, y_pred))
```

CNN LSTN MODEL:

```
from sklearn.preprocessing import MinMaxScaler
X = bin_data.iloc[:, 0:93].values
Y = bin_data[['intrusion']].values

# Scale features
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# Reshape for CNN
X_cnn = X_scaled.reshape(X_scaled.shape[0], X_scaled.shape[1], 1)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_cnn, Y, test_size=0.2,
random_state=42)

from keras.models import Sequential
```

```
from keras.layers import Dense, Dropout, LSTM, Conv1D, MaxPooling1D, Flatten

model_cnnlstm = Sequential()
model_cnnlstm.add(Conv1D(filters=32, kernel_size=3, activation='relu', input_shape=(X.shape[1], 1)))
model_cnnlstm.add(MaxPooling1D(pool_size=2))
model_cnnlstm.add(Flatten())
model_cnnlstm.add(Dense(64, activation='relu'))
model_cnnlstm.add(Dropout(0.3))
model_cnnlstm.add(Dense(1, activation='sigmoid'))
model_cnnlstm.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train CNN-LSTM
# Use X_train and y_train which are now consistent in size
model_cnnlstm.fit(X_train, y_train, epochs=5, batch_size=64, verbose=1)
```

```
Epoch 1/5
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107:
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
1575/1575 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.9623 - loss: 0.1264
Epoch 2/5
1575/1575 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9774 - loss: 0.0687
Epoch 3/5
1575/1575 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9786 - loss: 0.0657
Epoch 4/5
1575/1575 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9794 - loss: 0.0637
Epoch 5/5
1575/1575 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9797 - loss: 0.0627
<keras.src.callbacks.history.History at 0x792f704baed0>
```

Figure 17. Epoch 1/5

ENSEMBLE LEARNING:

```
pred_lstm = lst.predict(X_test)
pred_cnnlstm = model_cnnlstm.predict(X_test_cnn)
ensemble_pred = (pred_lstm + pred_cnnlstm) / 2
ensemble_labels = (ensemble_pred > 0.5).astype(int)
print("✅ Ensemble Accuracy:", accuracy_score(y_test, ensemble_labels))
print("\n📊 Classification Report:\n", classification_report(y_test, ensemble_labels))

cm = confusion_matrix(y_test, ensemble_labels)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Ensemble Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

Ensemble Accuracy: 0.9728914467156182

Classification Report:

	precision	recall	f1-score	support
0	0.98	0.96	0.97	11773
1	0.96	0.99	0.97	13422
accuracy			0.97	25195
macro avg	0.97	0.97	0.97	25195
weighted avg	0.97	0.97	0.97	25195

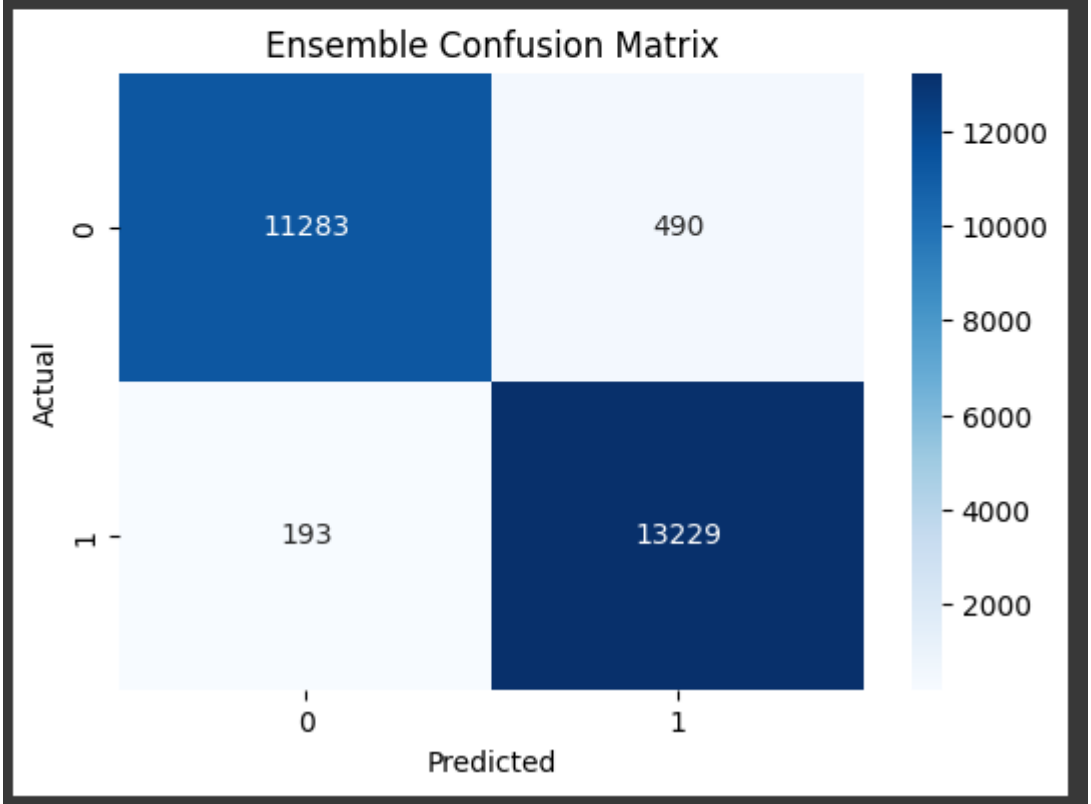


Figure 18. Ensemble Confusion Matrix