# INDUSTRIAL ORIENTED MINI PROJECT

## Report

On

# HYBRID APPROACH FOR NETWORK INTRUSION DETECTION SYSTEM USING MACHINE LEARNING

Submitted in partial fulfilment of the requirements for the award of the degree of

**BACHELOR OF TECHNOLOGY**

**In**

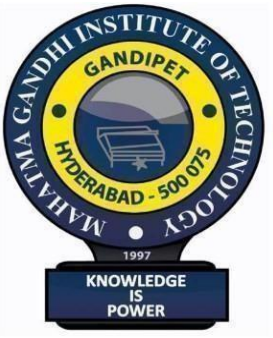
**INFORMATION TECHNOLOGY**

**By**

**Suragani Revathi-22261A1253**
**Patloori Durga-22261A1246**

Under the guidance of

**Mrs. Chinnakka Sudha**
Assistant Professor, Department of IT




**DEPARTMENT OF INFORMATION TECHNOLOGY**

**MAHATMA GANDHI INSTITUTE OF TECHNOLOGY (AUTONOMOUS)**

**(Affiliated to JNTUH, Hyderabad; Eight UG Programs Accredited by NBA; Accredited by NAAC with 'A++' Grade)**

**Gandipet, Hyderabad, Telangana, Chaitanya Bharati (P.O), Ranga Reddy District, Hyderabad– 500075, Telangana**

**2024-2025**

# CERTIFICATE

This is to certify that the **Industrial Oriented Mini Project** entitled **HYBRID APPROACH FOR NETWORK INTRUSION DETECTION SYSTEM USING MACHINE LEARNING** submitted by **Suragani Revathi-(22261A1253), Patloori Durga- (22261A1246)** in partial fulfillment of the requirements for the Award of the Degree of Bachelor of Technology in Information Technology as specialization is a record of the bonafide work carried out under the supervision of **Mrs. Chinnakka Sudha**, and this has not been submitted to any other University or Institute for the award of any degree or diploma.

**Internal Supervisor:**

**Mrs. Chinnakka Sudha**
Assistant Professor
Dept. of IT

**IOMP Supervisor:**

**Dr. U. Chaitanya**
Assistant Professor
Dept. of IT

**EXTERNAL EXAMINAR**

**Dr. D. Vijaya Lakshmi**
Professor and HOD
Dept. of IT

# DECLARATION

We hear by declare that the **Industrial Oriented Mini Project** entitled **HYBRID APPROACH FOR NETWORK INTRUSION DETECTION SYSTEM USING MACHINE LEARNING** is an original and bonafide work carried out by us as a part of fulfilment of Bachelor of Technology in Information Technology, Mahatma Gandhi Institute of Technology, Hyderabad, under the guidance of **Mrs. Chinnakka Sudha , Assistant Professor**, Department of IT, MGIT.

No part of the project work is copied from books /journals/ internet and wherever the portion is taken, the same has been duly referred in the text. The report is based on the project work done entirely by us and not copied from any other source.

**Suragani Revathi-22261A1253**
**Patloori Durga-22261A1246**

# ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success. They have been a guiding light and source of inspiration towards the completion of the **Industrial Oriented Mini Project**.

We would like to express our sincere gratitude and indebtedness to our project guide **Mrs.Chinnakka Sudha,** Assistant Professor, Dept. of IT, who has supported us throughout our project with immense patience and expertise.

We are also thankful to our honorable Principal of MGIT **Prof. G. Chandramohan Reddy** and **Dr. D. Vijaya Lakshmi**, Professor and HOD, Department of IT, for providing excellent infrastructure and a conducive atmosphere for completing this **Industrial Oriented Mini Project** successfully.

We are also extremely thankful to our IOMP supervisor **Dr. U. Chaitanya**, Assistant Professor, Department of IT, and senior faculty **Mrs. B. Meenakshi,** Assistant Professor Department of IT for their valuable suggestions and guidance throughout the course of this project.

We convey our heartfelt thanks to the lab staff for allowing us to use the required equipment whenever needed.

Finally, we would like to take this opportunity to thank our families for their support all through the work. We sincerely acknowledge and thank all those who gave directly or indirectly their support for completion of this work.

**Suragani Revathi-22261A1253**

**Patloori Durga-22261A1246**

# ABSTRACT

Intrusion Detection Systems (IDS) are essential for keeping networks safe by detecting unauthorized access and malicious behavior. Conventional signature-based IDS methods are capable of identifying known attacks but are unable to identify new or infrequent attack types, like Remote-to-Local (R2L) and User-to-Root (U2R), because they have unpredictable and ever- changing patterns. To overcome these weaknesses, this paper introduces a Double Layered Hybrid Approach (DLHA) that strengthens detection using a hybrid of machine learning and deep learning models.

The introduced DLHA incorporates Naive Bayes, Support Vector Machines (SVM), and an enhanced Bidirectional Long Short-Term Memory (BLSTM) network. Naive Bayes is used in the first layer to effectively detect common attack types such as Denial of Service (DoS) and Probe attacks. In the second level, SVM is utilized with BLSTM to separate the less common and more insidious R2L and U2R attacks from normal behaviors. For enhancing the accuracy of the model and limiting dimensionality, Principal Component Analysis (PCA) is utilized in the preprocessing phase, which identifies significant features and fluctuations in the data.

By exploiting the sequential learning capacity of BLSTM, the system successfully identifies temporal relationships in network traffic and thus more accurately detects sophisticated and infrequent intrusions. Experimental results show that the combined model significantly enhances overall detection accuracy and offers a more complete solution to emerging network security issues.

# LIST OF FIGURES

# LISTOF TABLES

# TABLE OF CONTENTS

# 1.INTRODUCTION

## 1.1 MOTIVATION

In today's digitally connected world, network security has become a critical concern for organizations and individuals alike. With the growing dependency on internet-based services and cloud computing, the frequency and sophistication of cyberattacks have also increased dramatically. Intrusion Detection Systems (IDS) are a vital component of modern cybersecurity infrastructure, as they help monitor network traffic and identify suspicious activities that may signal security breaches.

Traditionally, IDS technologies have relied heavily on signature-based detection methods, which work by comparing observed behaviors with a database of known attack patterns. While these systems are effective at identifying well-known threats, they struggle to detect novel or rare attack types, such as Remote-to-Local (R2L) and User-to-Root (U2R) intrusions. These types of attacks often mimic legitimate user behavior and exhibit subtle, unpredictable patterns, making them difficult to identify using conventional methods. Furthermore, attackers are constantly evolving their techniques, which further reduces the effectiveness of static, rule- based systems.

To address these challenges, Machine Learning (ML) and Deep Learning (DL) techniques have been explored for their ability to learn from data and generalize to unseen patterns. ML models can detect unknown intrusions by identifying deviations from normal behavior, while DL models, especially Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks, are capable of learning complex temporal and sequential patterns in data. However, relying solely on one type of model can limit the effectiveness of the IDS, particularly when dealing with highly imbalanced datasets or varied attack categories.

This project introduces a Double Layered Hybrid Approach (DLHA) for intrusion detection, which strategically combines the strengths of multiple models to enhance detection accuracy. The architecture consists of two layers:

Layer 1 employs Naive Bayes, a simple yet powerful probabilistic classifier, to detect more frequent and easily distinguishable attacks such as Denial of Service (DoS) and Probe attacks.

Layer 2 integrates Support Vector Machines (SVM) with an enhanced Bidirectional Long Short-Term Memory (BLSTM) network. This layer is specifically designed to capture the subtle patterns of less common and more sophisticated threats like R2L and U2R attacks.

To further improve the performance of the model and reduce computational complexity, Principal Component Analysis (PCA) is applied during the preprocessing stage. PCA helps in extracting the most significant features from the dataset and eliminates redundant or less relevant information, leading to more accurate and faster classification.

The proposed hybrid model leverages the sequential learning capabilities of BLSTM, the margin-based classification strength of SVM, and the simplicity of Naive Bayes to offer a robust, efficient, and scalable intrusion detection system. By combining these techniques, the DLHA can adapt to various types of network traffic and accurately distinguish between normal behavior and both common and rare attack patterns.

This paper details the architecture, implementation, and evaluation of the DLHA system using the widely-used NSL-KDD dataset, demonstrating significant improvements in detection rates and classification accuracy over traditional and single-model IDS approaches.

## 1.2 PROBLEM STATEMENT

With the growing sophistication of cyber threats, traditional Intrusion Detection Systems (IDS) based on signature matching are increasingly ineffective—especially against rare attack types like Remote-to-Local (R2L) and User-to-Root (U2R). These attacks often imitate legitimate user behavior , making them hard to detect with conventional methods.

To address these challenges, this paper proposes a Double Layered Hybrid Approach (DLHA) that combines Machine Learning (ML) and Deep Learning (DL) models for more robust intrusion detection.

Layer 1 uses the Naive Bayes classifier to efficiently detect common attack types such as

Denial of Service (DoS) and Probe attacks.

Layer 2 integrates Support Vector Machines (SVM) with an enhanced Bidirectional Long Short-Term Memory (BLSTM) network to identify subtle and less frequent attacks like R2L and U2R.

To improve accuracy and reduce computational load, Principal Component Analysis (PCA) is applied during preprocessing to select the most relevant features.

The combination of probabilistic modeling, sequence learning, and feature selection allows DLHA to outperform traditional IDS techniques. Experimental results show that this hybrid model achieves higher detection accuracy across diverse and imbalanced attack categories, making it a scalable and adaptable solution for modern cybersecurity needs.

## 1.3 EXISTING SYSTEM

The Traditional Intrusion Detection Systems (IDS) primarily rely on signature-based and anomaly-based techniques.

**Signature-Based IDS:**

These systems detect attacks by matching incoming traffic patterns with a predefined database of known threat signatures.

While highly accurate for known attacks, they fail to detect zero-day or novel attacks, such as R2L (Remote-to-Local) and U2R (User-to-Root).Examples: Snort, Suricata.Anomaly-Based IDS: These systems learn patterns of normal behavior and flag deviations as potential intrusions. Although better at detecting unknown threats, they often suffer from high false positive rates, especially when distinguishing rare attacks from normal behavior.

**Machine Learning-Based IDS:**

To further boost performance and reduce dimensionality, Principal Component Analysis (PCA) is applied during preprocessing. PCA selects the most relevant features, enabling more accurate and faster classification.

By leveraging the strengths of Naive Bayes, SVM, and BLSTM, DLHA offers a robust, scalable, and adaptive IDS solution. Experimental results demonstrate improved detection of both frequent and rare intrusions, making it a strong candidate for modern network security challenges.

ML models like Decision Trees, KNN, and SVM are used to classify network traffic.

While they improve adaptability, they are sensitive to imbalanced datasets (e.g., very few R2L and

U2R samples), leading to poor detection of minority classes.

**Deep Learning-Based IDS:**

Models like LSTM, CNN, and Autoencoders are used to detect complex and time-dependent threats. These approaches offer better accuracy but are computationally expensive and require large datasets and training time—making real-time detection difficult.

Limitations of the Existing System:

Poor detection of rare attacks (R2L, U2R). High false negatives for new or subtle intrusions.

Over-reliance on static features or outdated signatures. Lack of balance between detection accuracy and real-time performance.

## 1.3.1 LIMITATIONS

**High Computational Cost in Layer 2**

SVM and BLSTM require more processing power and memory, limiting scalability in real-time or large networks.

**Dependency on Quality and Balance of Dataset**

Performance is sensitive to imbalanced or outdated datasets, which may affect detection of new or rare attack patterns.

**Potential Loss of Important Features with PCA**

PCA may remove features that are crucial for detecting certain complex or unknown attacks.

**Limited Real-Time Deployment Testing**

While designed for efficiency, the system hasn't been extensively validated under real-time or high-speed network conditions.

**Lack of Auto-Adaptation to Evolving Threats**

The system does not learn continuously; it needs retraining to recognize newly emerging attack types .No Built-in Automated Response System.

The IDS detects attacks but does not include mechanisms for real-time response or threat mitigation.

## 1.4 PROPOSED SYSTEM

To overcome the limitations of traditional and standalone intrusion detection approaches, this project proposes a Double Layered Hybrid Intrusion Detection System (DLHA) that integrates Machine Learning and Deep Learning techniques with Principal Component Analysis (PCA) for enhanced efficiency and accuracy.

**Key Components of the Proposed System:**

**Principal Component Analysis (PCA):**

Used for dimensionality reduction to extract the most significant features from the dataset. Reduces noise and improves model training speed without losing critical attack information.

**Layer 1 – Naïve Bayes Classifier:**

Handles detection of frequent and linearly separable attacks such as Denial of Service (DoS) and Probe.

Naïve Bayes is chosen for its speed and efficiency on high-volume data.

**Layer 2 – Support Vector Machine (SVM) + Bidirectional Long Short-Term Memory (BLSTM):**

Focuses on the detection of rare and complex attacks such as Remote-to-Local (R2L) and User-to-Root (U2R).

SVM provides high-dimensional boundary separation, while BLSTM captures temporal and sequential patterns in network behavior for better detection of stealthy attacks.

**Hybrid Layered Architecture:**

By separating detection tasks across two specialized layers, the system achieves higher accuracy, lower false positives, and better resource efficiency.

This design allows real-time intrusion detection with improved coverage across both frequent and rare attack types.

Advantages of the Proposed System:

Improved detection of both frequent and rare attacks. Reduced false positives and false negatives.

Efficient handling of imbalanced datasets.

Enhanced detection of time-dependent patterns in network traffic. Suitable for real-time deployment in modern cybersecurity environments.

### 1.3.1 ADVANTAGES

- The hybrid layered approach significantly enhances the detection of both frequent (DoS, Probe) and rare (R2L, U2R) attacks.
- Reduced False Positives and False Negatives
- Layered specialization ensures that each model is applied where it performs best, minimizing misclassification rates.
- Efficient Feature Reduction with PCA
- PCA removes redundant and irrelevant features, improving model speed and reducing overfitting.
- Optimized Use of Resources
- Naïve Bayes in Layer 1 handles high-volume traffic quickly, while Layer 2 applies heavier models only where needed—ensuring real-time performance.
- Handles Imbalanced Data Effectively
- The architecture is designed to deal with skewed class distributions by isolating rare attacks for specialized analysis.
- Captures Temporal and Sequential Patterns
- BLSTM can recognize time-dependent behaviors, making it effective in detecting sophisticated and stealthy intrusions.
- Scalable and Adaptable Design
- The system can be extended to new datasets, protocols, or environments like IoT and cloud with minimal changes.
- Better Preparedness Against Unknown Attacks
- Unlike signature-based IDS, this system learns behavioral patterns, enabling the detection of zero-day or novel threats.

## 1.4 OBJECTIVES

- To develop an efficient Intrusion Detection System (IDS) capable of accurately identifying both frequent and rare types of cyberattacks in network traffic.
- To improve detection accuracy by integrating multiple classifiers in a double-layered

- architecture—Naïve Bayes for common attacks and SVM + BLSTM for rare, complex ones.
- To reduce false positives and false negatives, particularly for stealthy attacks like Remote- to-Local (R2L) and User-to-Root (U2R), which are often misclassified in traditional systems.
- To handle class imbalance in intrusion datasets by distributing detection responsibilities between simpler and advanced models based on attack complexity and frequency.
- To ensure the system supports real-time detection by using a layered model that minimizes computational load while maintaining high detection accuracy.
- To evaluate the proposed system on benchmark datasets (e.g., NSL-KDD) and compare its performance with traditional IDS models using metrics like accuracy, precision, recall, and F1-score.
- To apply Principal Component Analysis (PCA) for dimensionality reduction and feature selection, optimizing model training efficiency and eliminating redundant features.
- To leverage the temporal and sequential learning capabilities of BLSTM to detect behavior-based anomalies that unfold over time.

## 1.5 HARDWARE AND SOFTWARE REQUIREMENTS

### SOFTWARE REQUIREMENTS

- The project is developed using the Windows 10 or Windows 11 operating system, preferably 64-bit for better compatibility and performance during model training and testing.
- Python 3.x is the primary programming language used. Python 3.8 or higher is recommended to ensure compatibility with the latest machine learning and deep learning libraries.

- TensorFlow or PyTorch is used for implementing the Bidirectional Long Short-Term Memory (BLSTM) deep learning model, which handles temporal and sequential intrusion patterns.
- Scikit-learn is employed for implementing traditional machine learning models such as Naïve Bayes and Support Vector Machine (SVM), as well as Principal Component Analysis (PCA) for dimensionality reduction.
- NumPy and Pandas are used for efficient data preprocessing, handling, and transformation.
- Matplotlib and Seaborn are used for generating visualizations such as confusion matrices, bar graphs, and accuracy trends.
- The project is developed in the Visual Studio Code (VSCode) environment with Python extensions enabled, providing a lightweight and efficient IDE for both scripting and debugging.
- The system uses the NSL-KDD dataset, a widely recognized benchmark dataset for training and evaluating intrusion detection models.

**HARDWARE REQUIREMENTS**

- A system with an Intel Core i5, i7, or i9 processor (8th generation or higher) is recommended for ensuring smooth execution and faster model training.
- The minimum RAM requirement is 8 GB, although 16 GB or more is preferred for deep learning tasks and processing large datasets efficiently.
- A minimum of 256 GB SSD storage is required for faster data access and overall performance. SSDs also help reduce model training time compared to traditional hard drives.
- While optional, a dedicated GPU (e.g., NVIDIA GTX or RTX series) is highly recommended when using deep learning frameworks like TensorFlow or PyTorch to accelerate BLSTM training and inference.

# 2. LITERATURE SURVEY

M. Mohammadi, A. Navaras, and M. H. Amini (2021), in their work published in IEEE Xplore, proposed a Double Layered Hybrid Approach (DLHA) that integrates Naïve Bayes, SVM, and BLSTM for network intrusion detection. Their model achieved high accuracy in detecting common attacks like DoS and Probe. However, the system struggled with class imbalance issues when dealing with rarer attacks such as R2L and U2R..[1]

S. Shi, D. Han, and M. Cui (2023), through their publication in Connection Science (Taylor & Francis), introduced a Multimodal Hybrid Parallel IDS combining machine learning and deep learning techniques for feature extraction and classification. Their system showed improved performance in handling high-dimensional traffic data but incurred high computational cost, indicating a need for lightweight models suitable for real-time deployment scenarios.[2]

In 2024, Muhammad Sajid et al., in the Journal of Cloud Computing (Springer), proposed a hybrid IDS model integrating CNN, LSTM, and ensemble learning, specifically focused on cloud environments. Their approach enhanced anomaly detection but was limited by CNN's inability to effectively capture sequential dependencies in network traffic. The study suggested the need for improved feature selection techniques to further boost model performance.[3]

R. Jalili, S. Imani, and M. R. Aminzadeh (2021), in their work published in the ACM Digital Library, developed a CNN–LSTM-based anomaly detection system for Software Defined Networks (SDNs). Their model effectively detected unknown attacks and reduced false positives. However, the system suffered from high training time and resource consumption, indicating a need for optimization techniques to make it more efficient and scalable.[4]

Smitha Rajagopal, along with her colleagues Poornima Panduranga Kundapur and Katiganere Siddaramappa Hareesha, contributed to advancing intrusion detection systems by developing a supervised ensemble stacking model. Their work focuses on combining multiple machine learning algorithms to improve detection accuracy across different attack types. By testing, they demonstrated strong performance but also highlighted challenges like overfitting and need for real-world validation ensure broader applicability.[5]

Table 2.1 Literature Survey

| S.NO | AUTHOR NAMES,YEAR OF PUBLICATION | JOURNAL OR CONFERENCE NAME AND PUBLISHER NAME | METHODOLOGY/ ALGORITHM / TECHNIQUES USED | MERITS | DEMERITS | RESEARCH GAPS |
|---|---|---|---|---|---|---|
| 1 | M. Mohammadi, A. Navara's, & M. H. Amini (2021) | IEEE Xplore | Proposed a Double Layered Hybrid Approach (DLHA) using Naïve Bayes, SVM | Effective in detecting DoS and Probe | Struggles with class imbalance in R2L | Requires further optimization for real- time attack detection |
| 2 | S. Shi, D. Han, & M. Cui (2021) | Connection Science, Taylor & Francis | Introduced a Multimodal Hybrid Parallel IDS using ML and DL models for feature extraction and classification | Improved performance in handling high-dimensional network traffic data | High computational cost | Needs lightweight IDS solutions for real- time deployment |
| 3 | Muhammad Sajid et al. (2023) | Journal of Cloud Computing, Springer | Developed a Hybrid Machine and Deep Learning Approach integrating CNN, LSTM, and ensemble learning | Enhanced anomaly detection with a focus on cloud-based environment | CNN struggles with sequential dependencies in network traffic | Requires improvement in feature selection techniques for better performance |
| 4 | R. Jalili, S. Imani, & M. R. Aminzadeh (2024) | ACM Digital Library | Proposed a CNN-LSTM-based approach for anomaly detection in Software Defined Networks (SDNs) | Effective in detecting unknown attacks and reducing false positives | High training time and resource consumption | Needs optimization techniques to improve efficiency |
| 5 | Smitha Rajagopal And Poornima Panduranga Kundapur(2024). | Security and Communication networks(2024), Hindawi. | Designed a Supervised Ensemble Stacking Model combining multiple ML algorithms for IDS | High detection accuracy across multiple attack type | Overfitting issue due to ensemble complexity | Needs validation on real- world datasets for better generalization |

# 3. ANALYSIS AND DESIGN

**Problem Domain**

Modern networks are frequently targeted by a wide range of cyberattacks, including both known (DoS, Probe) and unknown or rare (R2L, U2R) threats. Traditional Intrusion Detection Systems (IDS) often fail to detect rare and evolving attacks, leading to high false negative rates. Additionally, most deep learning solutions are computationally expensive and unsuitable for real-time applications.

**Need for the System**

There is a strong need for a lightweight, accurate, and scalable IDS that: Handles imbalanced data effectively.

Detects both frequent and rare attacks. Supports real-time deployment.

Minimizes false positives and false negatives. Combines speed (ML) and intelligence (DL) effectively.

**Functional Requirements**

Preprocess network traffic data and extract key features. Classify the traffic using a layered ML/DL model.

Generate alerts for detected intrusions.

Provide performance evaluation through visualizations and logs.

**Non-Functional Requirements**

High accuracy and low latency.

Modular, scalable, and deployable in real-time environments. Minimal resource consumption for Layer 1 operations.

Maintainable and upgradable architecture.

**System Design**

The system is designed as a modular pipeline that integrates various components in a layered architecture. Each module performs a specific role, contributing to the overall effectiveness and efficiency of the IDS.

**1.Data Preprocessing Module**

Purpose: Converts raw network traffic into usable feature vectors. Processes: Data cleaning (handling missing values).
Label encoding and one-hot encoding.

Feature selection using Principal Component Analysis (PCA) to reduce dimensionality. Input: Raw network traffic from NSL-KDD or live network.
Output: Preprocessed data sent to detection layers.

**2.Layered Intrusion Detection Model**

This is the core module consisting of two layers:

Layer 1: Naïve Bayes Classifier

Role: Quickly and efficiently detects frequent attacks like DoS and Probe.

**Why Naïve Bayes?**

Low computational cost.

High speed on statistically separable data. Good performance on frequent class data. Outcome:
If classified as DoS/Probe → send alert.

If not → pass to Layer 2 for further inspection. Layer 2: SVM + BLSTM Hybrid Model
Role: Focuses on detecting rare and complex attacks like R2L and U2R.

**Why SVM?**

Handles nonlinear separation. Effective for small sample sizes.

**Why BLSTM?**
Learns temporal and sequential patterns in network behavior. Can detect sophisticated, behaviour - based intrusions.
Outcome: Classifies traffic as R2L, U2R, or Normal.

**3.Deployment Framework**

Acts as an integration layer for receiving real-time network traffic. Sends traffic both to the preprocessing pipeline and the classification model Manages communication between detection modules and deployment infrastructure (e.g.,servers , routers, dashboards).

**4.Decision Module**

Gathers classification results from both layers. Performs logic-based decisions to determine if a connection is normal or malicious. Also decides which alerts to forward to the alert system based on confidence scores or thresholds.

**5.Evaluation and Alert System**

Functionality : Issues alerts for suspicious or malicious activity.

Logs detected intrusions for further analysis.

Generates metrics such as accuracy, precision, recall, F1-score, and confusion matrix. Uses Matplotlib and Seaborn for visual analysis.

Output: Admin interface or log files showing attack summaries and system performance. Benefits of the Design Layered processing ensures faster detection for frequent attacks and deeper inspection for rare ones. Modularity makes it easy to maintain, update, or replace any component. Combines statistical models (Naïve Bayes, SVM) with deep learning (BLSTM) to achieve both speed and intelligence. The deployment framework supports integration with live systems for real-time alerting.

## 3.1MODULES

**1.Data Preprocessing & Feature Selection**

This module is responsible for preparing raw network traffic data for model input. The system utilizes Principal Component Analysis (PCA) to:

Extract relevant features that contribute the most to variance. Eliminate redundant or irrelevant features.

Reduce dimensionality, thereby decreasing computational complexity and training time. The preprocessing also involves:

Data cleaning (handling missing/null values). Normalization or scaling of numerical values. Encoding categorical attributes.

Outcome: A clean, optimized feature set is passed to the detection layers.

**2.Layer 1: Naïve Bayes Classification**

This module performs initial filtering by classifying frequent and well-separated attacks, specifically:

DoS (Denial of Service) Probe attacks

Naïve Bayes is chosen due to its:

Simplicity and speed.

Effectiveness on linearly separable and high-frequency data.

This layer significantly reduces the workload of deeper models by quickly eliminating obvious threats.

Outcome: Traffic identified as DoS/Probe → directly labeled as attack.

Other traffic → passed to Layer 2 for further inspection.

**3.Layer 2: SVM + BLSTM for Rare Attack Detection**

This layer is designed to detect rare and complex attacks that are difficult to classify using simpler models.

**a)Support Vector Machine (SVM):**

Separates normal and anomalous traffic using high-dimensional decision boundaries. Effective on small sample sizes like R2L (Remote-to-Local) and U2R (User-to-Root) classes.

**b)Bidirectional Long Short-Term Memory (BLSTM):**

BLSTM is a type of Recurrent Neural Network (RNN) that captures temporal and sequential patterns in network traffic.

It learns from past and future behavior in traffic sequences, making it ideal for detecting hidden or stealthy attacks.

Combined with SVM, this hybrid model improves detection precision for behavior-based threats.

Outcome: Improved classification of R2L and U2R attacks, which often resemble normal user behavior.

**3.Intrusion Detection & Classification**

This module integrates predictions from both Layer 1 and Layer 2 to: Label network traffic as normal or anomalous.

Minimize false positives and false negatives. A decision logic ensures:

Confidence-based assessment of borderline cases.

Filtering of noisy predictions using thresholds or confidence scores.

Outcome: Accurate and explainable classification with reduced misclassification rates.

**4.Performance Evaluation & Real-Time Implementation**

After detection, this module performs model evaluation and alert generation. It computes and

displays key performance metrics:

Accuracy Precision Recall F1-Score

Confusion Matrix

Visualizations are generated using Matplotlib and Seaborn.

If implemented in real-time, the module:

Connects to the Deployment Framework. Triggers alert notifications for suspicious traffic. Logs intrusion events for further review and system auditing.

## 3.1ARCHITECTURE

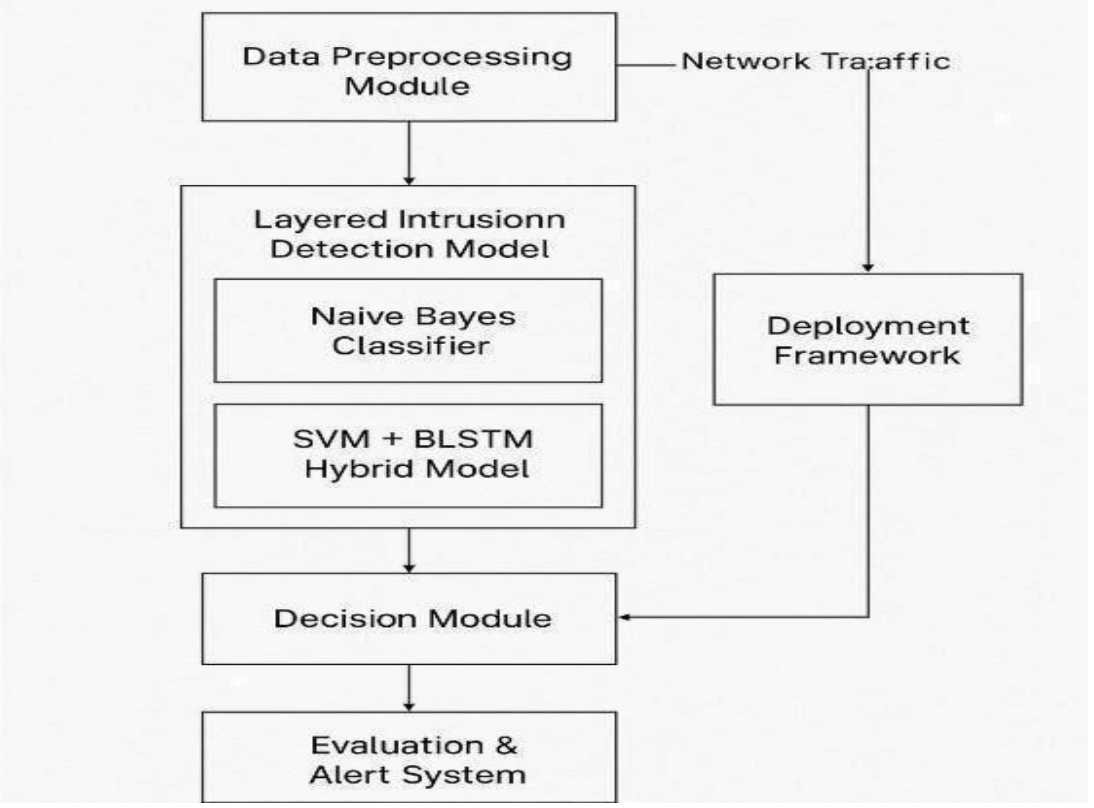



Fig. 3.2.1 Architecture of Intrusion Detection System

The proposed Hybrid Intrusion Detection System (HIDS) is designed using a layered and modular architecture that integrates both machine learning and deep learning techniques to detect a wide range of cyberattacks. It starts by collecting raw network traffic data, either from a dataset like NSL-KDD or real-time sources. This data is processed through a Data Preprocessing Module, where noise is reduced and features are selected using Principal

Component Analysis (PCA). PCA helps in minimizing redundancy and dimensionality, improving the efficiency and accuracy of the detection model.

After preprocessing, the cleaned data is passed into a two-layer intrusion detection model. The first layer uses a Naïve Bayes classifier to detect frequent and statistically distinct attacks such as Denial of Service (DoS) and Probe. This layer serves as a fast and lightweight filter that reduces the workload on the more complex second layer by quickly classifying easily detectable traffic patterns. Any uncertain or complex cases are forwarded to the next layer for deeper analysis.

The second layer is a hybrid of Support Vector Machine (SVM) and Bidirectional Long Short-Term Memory (BLSTM). The SVM classifier separates normal and anomalous connections based on nonlinear decision boundaries, while BLSTM captures temporal and sequential patterns in network behavior. This combination is particularly effective for detecting rare and stealthy attacks like Remote-to-Local (R2L) and User-to-Root (U2R), which often mimic normal activity and are harder to identify using conventional methods.

Finally, the results from both layers are processed in the Decision and Evaluation Module, where the traffic is labeled as either normal or malicious. Alerts are generated for detected attacks, and the system evaluates its performance using metrics like accuracy, precision, recall, and F1-score. Visual tools such as Matplotlib and Seaborn help in displaying real-time results. A deployment framework integrates all components, enabling real-time detection and making the system suitable for implementation in enterprise networks, cloud environments, or software-defined networks.

## 3.3 UML DIAGRAMS

UML (Unified Modeling Language) diagrams visually represent the structure and behavior of software systems. They include types like Class, Use Case, Sequence, and Activity diagrams. UML helps model system components, interactions, and workflows clearly. It's widely used for planning, designing, and documenting software architecture.

### 3.3.1 USE CASE DIAGRAMS

The Use Case Diagram illustrates the interaction between the user and the core functional components of the system. The user (typically a system administrator or data analyst) performs a series of actions to execute the intrusion detection process, starting from data preprocessing to evaluating the final model performance. Each step is represented as a use case, showing the system's responsibilities and data flow. Fig. 3.3.1.1. The process begins with the user loading and preprocessing the raw network data. After that, feature selection is performed using Principal Component Analysis (PCA) to extract relevant features and reduce dimensionality. The refined data is then passed through a layered model. The first layer involves training and predicting with a Naïve Bayes classifier, which quickly detects common attacks like DoS and Probe. Traffic not identified at this stage proceeds to the second layer, where SVM and BLSTM are used to detect more complex and rare attacks like R2L and U2R. Finally, the system evaluates the model's performance using metrics such as accuracy, precision, recall, and F1-score, enabling the user to assess the system's effectiveness.

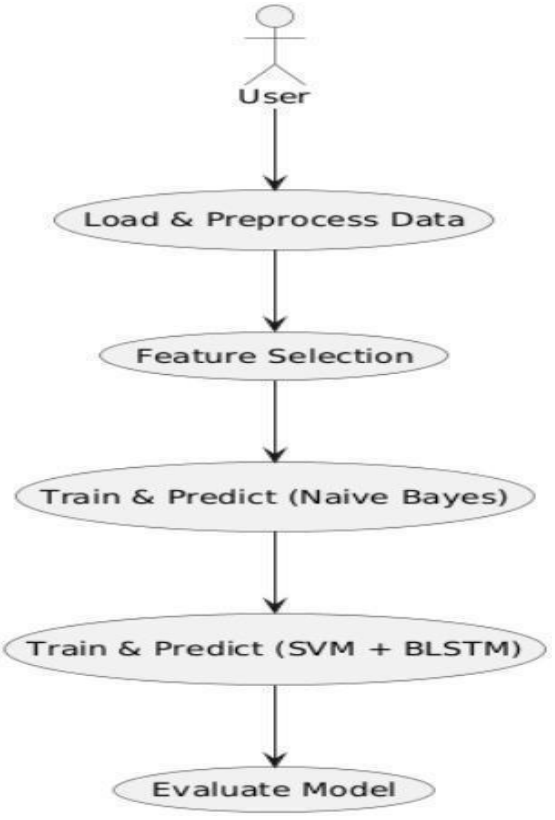



Fig. 3.3.1.1 Use Case Diagram

**Load & Preprocess Data**

- The user uploads raw network traffic data (e.g., NSL-KDD dataset).
- The system performs data cleaning, encoding, and normalization to prepare the data for analysis.

**Feature Selection**

- The system applies Principal Component Analysis (PCA) to select the most significant features.
- This reduces dimensionality and removes irrelevant or redundant data.

**Train & Predict (Naïve Bayes)**

- The system trains a Naïve Bayes classifier to detect common and frequent attacks (e.g., DoS, Probe).
- Performs quick classification as an initial filtering layer.

**Train & Predict (SVM + BLSTM)**

- For data not classified in the first layer, the system uses a Support Vector Machine (SVM) combined with Bidirectional LSTM (BLSTM) to detect rare attacks like R2L and U2R.
- SVM handles complex class boundaries, and BLSTM captures temporal/sequential behavior.

**Evaluate Model**

- The system computes performance metrics such as accuracy, precision, recall, and F1-score.
- Results are visualized using graphs to assess the overall effectiveness of the IDS.

## 3.3.2 CLASS DIAGRAM

This class diagram models the core functional components of your intrusion detection system, structured in a modular, object-oriented way. Each class represents a major phase of the detection pipeline, with dedicated methods (functions) that encapsulate the logic for each task.

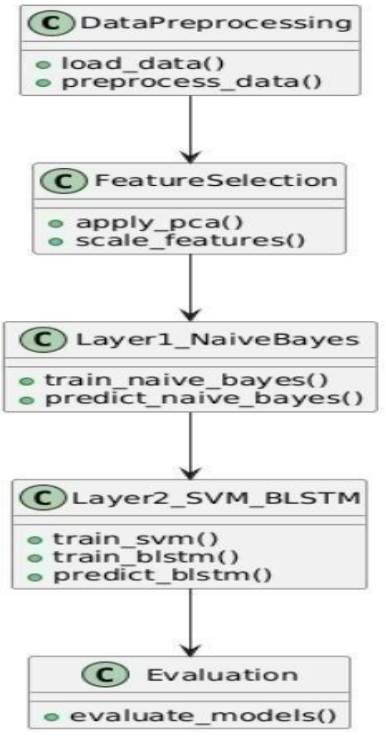



Fig. 3.3.2.1 Class Diagram

**Data Preprocessing Class Methods:**

load_data(): Loads the NSL-KDD dataset or real-time traffic input. preprocess_data(): Cleans, encodes, and prepares the data for feature selection. Purpose: Responsible for acquiring and preparing raw input for processing.

FeatureSelection Class Methods:

apply_pca(): Applies Principal Component Analysis (PCA) for dimensionality reduction.

scale_features(): Normalizes or standardizes the feature set.

Purpose: Optimizes the feature set to improve model accuracy and reduce training complexity.

Layer1_NaiveBayes Class Methods:

Train _naive_ bayes(): Trains a Naïve Bayes model using the selected features. Predict _naive _bayes(): Predicts DoS and Probe attacks from the input data. Purpose: Acts as the first classification layer for frequent attacks.

Layer2_SVM_BLSTM Class Methods:

Train _svm(): Trains an SVM classifier for rare attacks.

Train _blstm(): Trains the Bidirectional LSTM model to learn temporal patterns. Predict _blstm(): Uses the trained BLSTM to classify R2L and U2R attacks.

Purpose: Serves as the second classification layer for rare and complex attacks. Evaluation Class Methods:

Evaluate models(): Computes performance metrics like accuracy, precision, recall, and F1-score.

Purpose: Assesses how well the models perform using test data and generates reports.

Data Preprocessing → Feature Selection

The preprocessed data is passed to the feature selection class. Relationship Type: Association

Feature Selection → Layer1_NaiveBayes

Once features are selected, they are used to train and predict with the Naïve Bayes model. Relationship Type: Association

System Flow (Step-by-Step Execution)

User invokes load _data() in the Data Preprocessing class to load the dataset. Preprocess _data() cleans and prepares the data.

Preprocessed data is passed to the Feature Selection class.

Apply _pca() and scale _features() transform the data into an optimized format. The transformed data is passed to the Naïve Bayes layer:

train_naive_bayes() trains the model.

predict_naive_bayes() detects frequent attacks (DoS, Probe). Samples not identified as attacks are passed to SVM + BLSTM layer: train_svm() and train_blstm() train the models.

predict_blstm() classifies complex attacks like R2L and U2R. Final predictions are sent to the Evaluation class. evaluate_models() computes metrics and visualizes results.

### 3.3.2 ACTIVITY DIAGRAM

This activity diagram represents the step-by-step workflow of the intrusion detection system. It models the flow of control from the moment network data is loaded to the point where the system classifies threats, identifies normal traffic, and finally generates a report. Fig. 3.3.3.1 he diagram is structured to reflect the layered approach of the system, where the input data flows through multiple processing and classification stages. This ensures that frequent attacks (DoS, Probe) are handled efficiently in the first layer, while complex and rare attacks (R2L, U2R) are processed in a more advanced second layer using deep learning techniques.

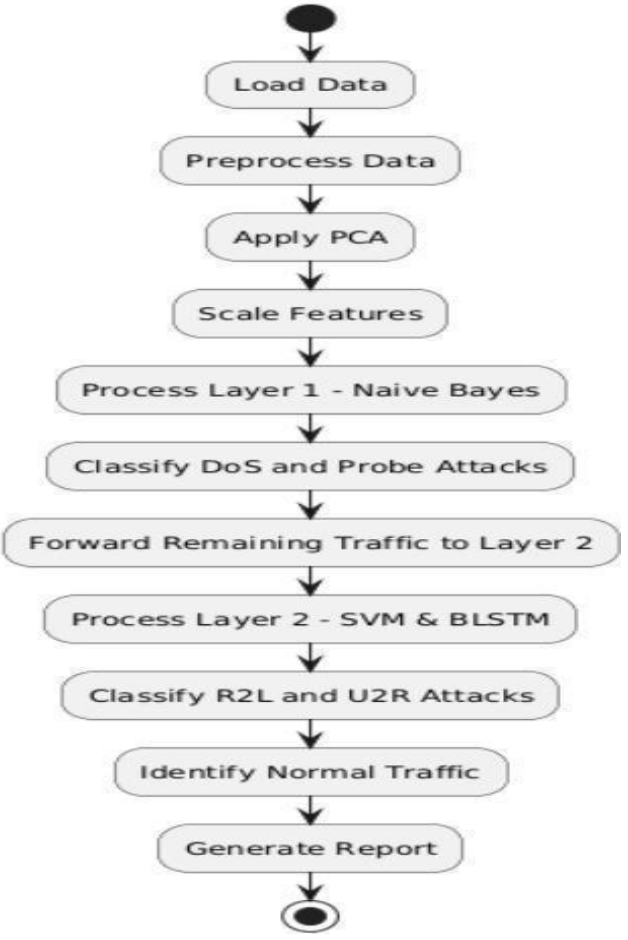



Fig. 3.3.3.1 Activity Diagram

**Flow Explanation (Step-by-Step)**

Start Node (●)

The system process begins when the intrusion detection workflow is triggered by the user or system.

**Load Data**

The dataset (e.g., NSL-KDD) or live traffic is loaded into the system for analysis.

**Preprocess Data**

The raw data is cleaned and prepared: Missing values are handled.

Categorical features are encoded. Data is normalized or scaled.

Apply PCA (Principal Component Analysis)

PCA is applied to reduce feature dimensionality and retain only the most significant variables for classification.

**Scale Features**

The selected features are scaled to ensure consistency across the model input (e.g., Min-Max Scaling or Standardization).

**Process Layer 1** – Naïve Bayes

The first classification layer uses a Naïve Bayes model to analyze the data quickly and classify known, frequent attacks.

Classify DoS and Probe Attacks

If the data is identified as a DoS or Probe attack, it is immediately labeled and handled. Forward Remaining Traffic to Layer 2

Traffic that is not confidently classified by Naïve Bayes (e.g., possibly normal or rare attacks) is passed to the next layer for deeper analysis.

**Process Layer 2** – SVM & BLSTM

SVM is used to identify complex boundary patterns.

BLSTM learns sequential dependencies in the data (important for detecting behavior-based intrusions).

**Classify R2L and U2R Attacks**

The second layer classifies traffic as Remote-to-Local (R2L) or User-to-Root (U2R) attacks if patterns match these rare types.

Identify Normal Traffic

Traffic that is not identified as any known attack is labeled as normal. Generate Report

A detailed report is generated showing:

Attacks identified Performance metrics (accuracy, precision, recall, F1-score)

End Node (◎)

The activity concludes once the classification and reporting process is complete.

### 3.3.4 SEQUENCE DIAGRAM

This sequence diagram illustrates the dynamic flow of interactions between the user and the system's key modules in a time-ordered sequence. It shows how the user-initiated process (starting with data loading) proceeds through data preprocessing, feature selection, layered classification, and model evaluation, ultimately producing results. Fig. 3.3.4.1

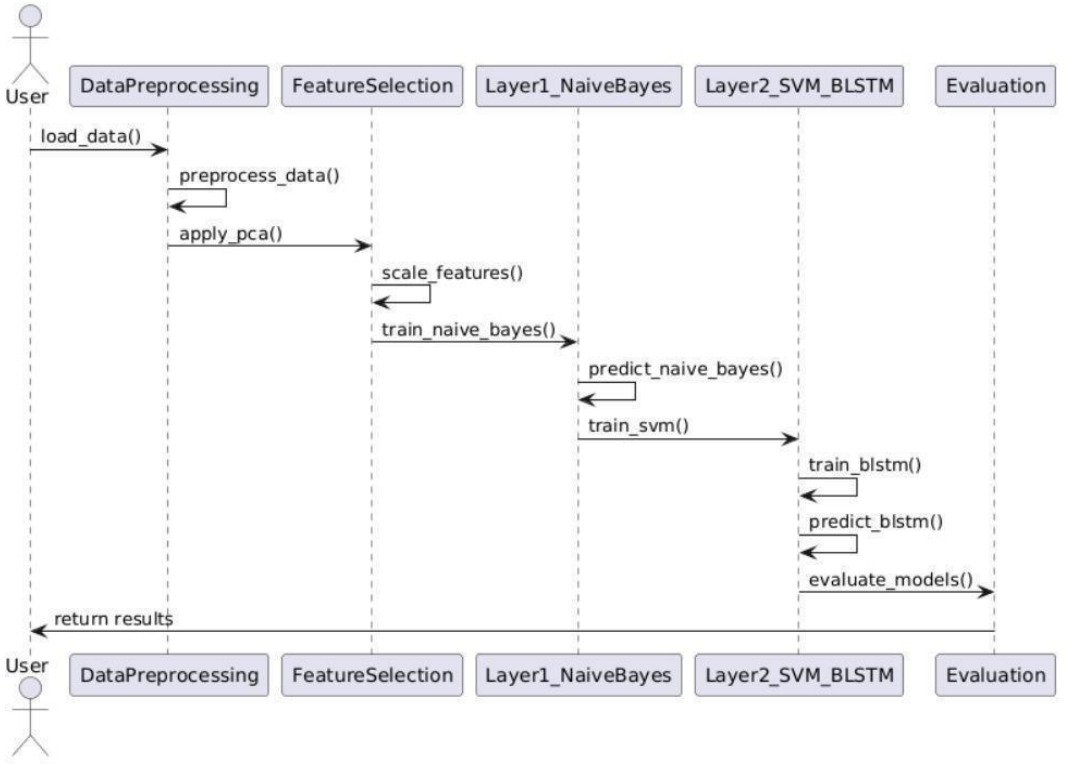



Fig. 3.3.4.1 Sequence Diagram

The diagram models the order of operations and the communication between different system components, namely:

- DataPreprocessing
- FeatureSelection
- Layer1_NaiveBayes
- Layer2_SVM_BLSTM
- Evaluation

Each module has defined responsibilities and interacts with others through function/method calls to complete the overall intrusion detection process. Interactions (Message Flow)
User → Data Preprocessing : load_data()

The user initiates the workflow by loading the dataset (e.g., NSL-KDD).

Data Preprocessing → Data Preprocessing : preprocess_data() Cleans, encodes, and normalizes the raw network data. Data Preprocessing → Feature Selection : apply_pca() Applies PCA for feature selection, reducing dimensionality. Feature Selection

Feature Selection : scale_features() Standardizes features to ensure consistent input for models.
Feature Selection → Layer1_NaiveBayes : train_naive_bayes() Trains the Naïve Bayes classifier on the transformed data.
Layer1_NaiveBayes →Layer1_NaiveBayes : predict_naive_bayes() Predicts frequent attacks (e.g., DoS, Probe) using trained model.
Layer1_NaiveBayes → Layer2_SVM_BLSTM : train_svm() Initiates training of SVM for rare/complex attack detection. Layer2_SVM_BLSTM → Layer2_SVM_BLSTM : train_blstm() Trains the Bidirectional LSTM model for sequential data learning. Layer2_SVM_BLSTM → Layer2_SVM_BLSTM : predict_blstm() Predicts complex intrusions like R2L and U2R. Layer2_SVM_BLSTM → Evaluation : evaluate_models()
Model performance is evaluated using accuracy, precision, recall, F1-score.

Evaluation → User : return results

Final results (classified traffic and performance metrics) are returned to the user.

### 3.3.5 COMPONENT DIAGRAM

This component diagram visually represents the high-level structural organization of your Hybrid Intrusion Detection System. It highlights the key components (modules) and their interactions during the intrusion detection workflow. as shown in Fig. 3.3.5.1. The system follows a modular design, where each component is responsible for a specific task ranging from data preprocessing to final evaluation. These modules are loosely coupled and work sequentially to ensure scalability, maintainability, and ease of enhancement.

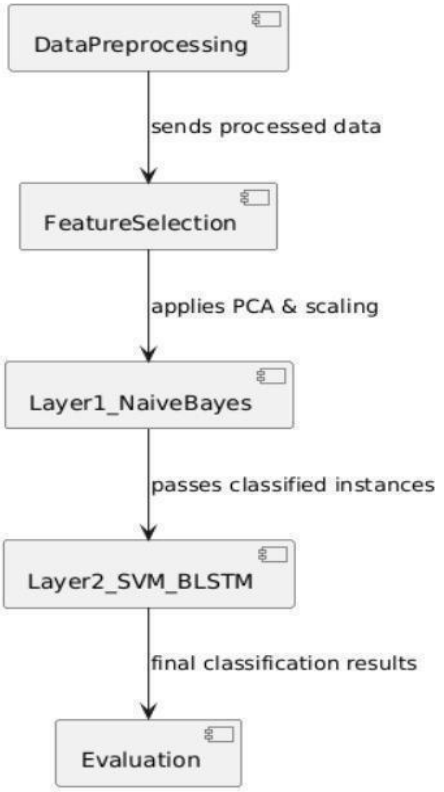



Fig. 3.3.5.1 Component Diagram

**Main Components:**

**1.Data Preprocessing**

- Role: This component is responsible for loading and cleaning the raw dataset (e.g., NSL-KDD).Operations: Handles missing values, normalizes data, and encodes categorical features.
- Interaction: Sends the cleaned and formatted data to the FeatureSelection component.

**2.Feature Selection**

- Role: Applies dimensionality reduction and feature transformation.
- Techniques Used:PCA (Principal Component Analysis) – for extracting the most relevant features.
- Scaling – to normalize features across the dataset.
- Interaction: Passes the transformed data to the first classification layer (Layer1_NaiveBayes).

**3.Layer1_NaiveBayes**

- Role: Performs initial classification to detect frequent attacks such as DoS and Probe.

- Why Naive Bayes? It is lightweight, fast, and effective for well-separated, common attack patterns.
- Interaction: Classifies straightforward cases and forwards uncertain or rare samples to the second classification layer.

**4.Layer2_SVM_BLSTM**

- Role: Handles complex and rare attacks like R2L (Remote-to-Local) and U2R (User-to-Root).
- Components Used:SVM – for high-dimensional classification.
- BLSTM – to capture sequential patterns in the data (behavioral-based attacks).
- Interaction: Produces final classification results and passes them to the evaluation module.

**5.Evaluation**

- Role: Analyzes the performance of the layered detection system.
- Functions:Computes metrics like accuracy, precision, recall, and F1-score.
- Optionally generates reports and visual outputs using libraries like Matplotlib or Seaborn.
- Outcome: Final feedback is provided to the user/admin based on model performance.

### 3.3.6 DEPLOYMENT DIAGRAM

The deployment diagram illustrates how the software components of your HIDS project are distributed across hardware nodes—specifically a User Device, Server, and Database. It shows the physical architecture of the system and how different components communicate during execution.

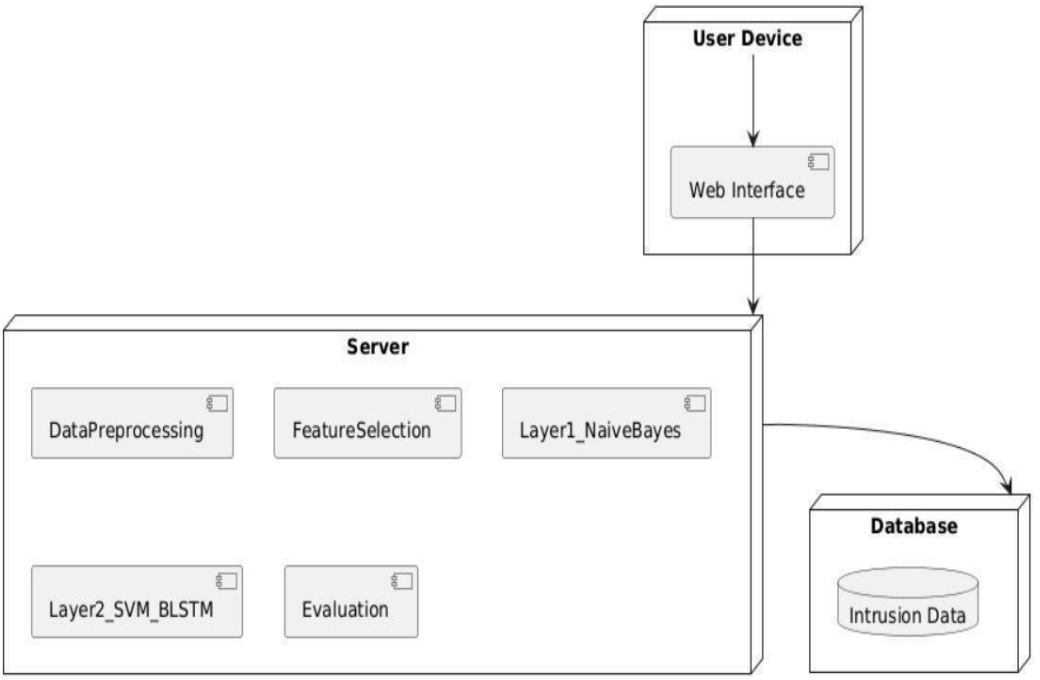



Fig. 3.3.6.1 Deployment Diagram

This architecture supports modularity, scalability, and allows for web-based interaction, making the solution more practical for real-world deployment scenarios.

**Nodes and Components**

1.User Device

Component: Web Interface

Function: Acts as the front-end interface for administrators or security analysts. Allows users to upload datasets, initiate training or testing, and view alerts or evaluation reports.

Interaction: Sends user input to the server and receives processed results.

2.Server

This is the core computational node where all processing and model logic resides.

**Contained Components** :

Data Preprocessing Loads raw data and prepares it (cleaning, encoding, normalization)

Feature Selection

Applies PCA and scaling to reduce feature dimensionality and improve model performance.

Layer1_NaiveBayes

First classification layer; quickly identifies common attacks like DoS and

Probe.Layer2_SVM_BLSTMSecond classification layer; performs deep analysis using SVM and BLSTM to detect complex attacks like R2L and U2R.

Evaluation

Computes performance metrics and generates results for visualization.

**Purpose of Server:**

Central processing unit where machine learning and deep learning models are trained, executed, and evaluated.

Supports interaction with both frontend (user) and backend (database).

## 3.4 METHODOLOGY

The proposed Hybrid Intrusion Detection System (HIDS) begins by collecting and preprocessing network traffic data, typically from the NSL-KDD dataset. The data undergoes cleaning, encoding, and normalization to ensure it is suitable for machine learning. To reduce dimensionality and improve efficiency, Principal Component Analysis (PCA) is applied, which helps retain only the most relevant features while eliminating noise and redundancy.

The detection process is structured into two layers. The first layer uses a Naïve Bayes classifier to quickly identify frequent attacks such as DoS and Probe, providing a fast and lightweight filtering mechanism. Any traffic not confidently classified in this stage is passed to the second layer, which uses a hybrid of SVM and BLSTM. SVM handles high-dimensional feature separation, while BLSTM captures sequential and temporal behavior, allowing effective detection of complex and rare attacks like R2L and U2R.

Finally, the system evaluates the results using metrics like accuracy, precision, recall, and F1-score, and visualizes them using tools like Matplotlib and Seaborn. The classified data, along with the performance report, is presented to the user through a web interface, making the system suitable for both experimental and real-time intrusion detection scenarios.

**Models Used in the Project**

**1.Naïve Bayes Classifier**

Type: Probabilistic Machine Learning Model (Supervised) Use: Employed in Layer 1 of the system.

Purpose: Quickly classifies frequent and linearly separable attacks such as DoS (Denial of Service) and Probe.

Based on Bayes' Theorem, assuming feature independence. Advantages:

Fast, simple, and effective on large datasets.

Low computational cost, suitable for initial filtering.

**2.Support Vector Machine (SVM)**

Type: Supervised Machine Learning Model Use: Part of Layer 2, combined with BLSTM. Purpose:

Separates complex, high-dimensional data by finding optimal hyperplanes. Helps in detecting anomalous patterns and rare attacks.

Advantages:

Works well with small, imbalanced datasets. Strong generalization capability.

**3.Bidirectional Long Short-Term Memory (BLSTM)**

Type: Deep Learning Model (Recurrent Neural Network variant) Use: Also part of Layer 2. Purpose:

Learns temporal and sequential patterns in network traffic. Enhances detection of behavior-based and stealthy intrusions, especially R2L (Remote to Local) and U2R (User to Root). Advantages:

Captures both past and future data dependencies.

Excellent for time-series and sequence-driven intrusion behaviors.

**4.Principal Component Analysis (PCA)**

Type: Dimensionality Reduction Technique (Unsupervised) Use: During Feature Selection phase.

**Purpose:**

Reduces the number of input features while preserving important variance. Helps improve model performance and reduce overfitting.

**Advantages:**

Speeds up training.

Removes noise and redundancy from data.

# 4.CODE AND IMPLEMENTATION

## 4.1 CODE

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tkinter as tk
import pyttsx3
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
from sklearn.naive_bayes import GaussianNB
from sklearn import svm
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
from collections import Counter
import winsound
import customtkinter as ctk
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Bidirectional
from tensorflow.keras.utils import to_categorical
from tkinter import messagebox
import warnings
warnings.filterwarnings('ignore')

pd.set_option('display.float_format', lambda x: '%.3f' % x)
plt.rcParams["figure.figsize"] = (10,6)

df_0 = pd.read_csv("KDDTrain+.txt")
df = df_0.copy()
print(df.head())
columns =
```

```
(['duration','protocol_type','service','flag','src_bytes','dst_bytes','land','wrong_fragment','urg
ent',
'hot','num_failed_logins','logged_in','num_compromised','root_shell','su_attempted','num_r
oot','num_file_creations',
'num_shells','num_access_files','num_outbound_cmds','is_host_login','is_guest_login','coun
t','srv_count','serror_rate',
'srv_serror_rate','rerror_rate','srv_rerror_rate','same_srv_rate','diff_srv_rate','srv_diff_host_r
ate',
'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','attack','level'])
df.columns = columns
print(df.head(5))
print(df.info())
print(df.describe().T)
print(df.isnull().sum())

def unique_values(df, columns):
    """Prints unique values and their counts for specific columns in the DataFrame."""
    for column_name in columns:
        print(f"Column: {column_name}\n{'-'*30}")
        unique_vals = df[column_name].unique()
        value_counts = df[column_name].value_counts()
        print(f"Unique Values ({len(unique_vals)}): {unique_vals}\n")
        print(f"Value Counts:\n{value_counts}\n{'='*40}\n")

cat_features = df.select_dtypes(include='object').columns
unique_values(df, cat_features)
print(df.duplicated().sum())

attacks_types = {
    'normal': 'normal','back': 'dos','buffer_overflow': 'u2r','ftp_write': 'r2l','guess_passwd':
'r2l','imap': 'r2l',
```

```
    'ipsweep': 'probe','land': 'dos','loadmodule': 'u2r','multihop': 'r2l','neptune': 'dos','nmap':
'probe',
    'perl': 'u2r','phf': 'r2l','pod': 'dos','portsweep': 'probe','rootkit': 'u2r','satan': 'probe',
    'smurf': 'dos','spy': 'r2l','teardrop': 'dos','warezclient': 'r2l','warezmaster': 'r2l'
}
df['label'] = df['attack'].apply(lambda r: attacks_types.get(r.strip(), 'unknown'))

def preprocess_data(data):
    categorical_cols = ['protocol_type', 'service', 'flag']

    X = data.drop(['label'], axis=1)
    y = data['label']

    class_names = sorted(y.unique())

    # One-hot encoding
    X = pd.get_dummies(X, columns=categorical_cols)
    print(X)
    # Convert all columns to numeric, fill missing values with 0
    X = X.apply(pd.to_numeric, errors='coerce').fillna(0)
    print(X)

    # Encode labels
    label_encoder = LabelEncoder()
    y_encoded = label_encoder.fit_transform(y)

    # Print processed data
    print("\n🔷 Features (X):")
    print(X.head())

    print("\n🔷 Encoded Labels (y_encoded):")
    print(y_encoded[:10])
```

```
    print("\n🔷 Original Labels (y):")
    print(y.head())

    print("\n🔷 Label Mappings:")
    for i, label in enumerate(label_encoder.classes_):
        print(f"{i} -> {label}")

    return X, y_encoded, label_encoder, class_names, y


def preprocess_test_data(test_data, scaler, pca, label_encoder, train_columns):
    categorical_cols = ['protocol_type', 'service', 'flag']

    # Separate features and label
    X_test = test_data.drop(['label'], axis=1)
    y_test = test_data['label']

    # One-hot encode categorical features
    X_test = pd.get_dummies(X_test, columns=categorical_cols)
    print( X_test)

    # Align columns with training data
    X_test = X_test.reindex(columns=train_columns, fill_value=0)
    print( X_test)

    # Convert to numeric and handle missing values
    X_test = X_test.apply(pd.to_numeric, errors='coerce').fillna(0)
    print( X_test)

    # Scale and reduce dimensions
    X_test_scaled = scaler.transform(X_test)
    print( X_test_scaled)
    X_test_pca = pca.transform(X_test_scaled)
```

```
    # Handle unseen labels gracefully
    unknown_labels = set(y_test) - set(label_encoder.classes_)
    if unknown_labels:
        print(f"\n⚠ Warning: Unseen labels in test data: {unknown_labels}")
        y_test = y_test.apply(lambda x: x if x in label_encoder.classes_ else None)
        mask = y_test.notna()
        y_test_filtered = y_test[mask]
        X_test_pca = X_test_pca[mask]
        y_test_encoded = label_encoder.transform(y_test_filtered)
    else:
        y_test_encoded = label_encoder.transform(y_test)

    # Print debug info
    print("\n🔷 Processed Test Features (PCA output):")
    print(X_test_pca[:5])

    print("\n🔷 Encoded Test Labels:")
    print(y_test_encoded[:10])

    print("\n🔷 Original Test Labels:")
    print(y_test.head())

    return X_test_pca, y_test_encoded, y_test


def build_blstm_model(input_dim):
    model = Sequential()
    model.add(Bidirectional(LSTM(64), input_shape=(1, input_dim)))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(22, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
    return model

def plot_confusion_matrix(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(10,8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(title)
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.show()

def plot_performance_graph(history, model_name):
    plt.figure(figsize=(10, 6))
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Test Accuracy')
    plt.title(f'{model_name} Performance')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend(loc='upper left')
    plt.show()

def plot_metrics_bar(y_true, y_pred, model_name):
    acc = accuracy_score(y_true, y_pred)
    prec = precision_score(y_true, y_pred, average='weighted', zero_division=0)
    rec = recall_score(y_true, y_pred, average='weighted', zero_division=0)
    f1 = f1_score(y_true, y_pred, average='weighted', zero_division=0)
    metrics = [acc, prec, rec, f1]
    names = ['Accuracy', 'Precision', 'Recall', 'F1-Score']
    plt.figure(figsize=(8,6))
    sns.barplot(x=names, y=metrics)
    plt.ylim(0, 1)
    plt.title(f'{model_name} Metrics')
    plt.ylabel('Score')
    plt.show()
```

```
def plot_final_accuracy_comparison(acc_nb, acc_svm, acc_blstm):
    combined_svm_blstm = acc_svm + acc_blstm
    plt.figure(figsize=(8,6))
    sns.barplot(x=["Naive Bayes", "SVM + BLSTM"], y=[acc_nb, combined_svm_blstm],
palette='Set2')
    plt.title("Layer-wise Accuracy Comparison (on KDDTest+)")
    plt.ylabel("Accuracy")
    plt.ylim(0, 1)
    plt.show()

def plot_comparison_graphs():
    previous_models = {
        "CNN": {"accuracy": 75.67, "precision": 66.23, "recall": 70.89, "f1_score": 75.0},
        "Deep Learning": {"accuracy": 77.20, "precision": 70.99, "recall": 73.11, "f1_score":
78.0},
        "LSTM": {"accuracy": 83.45, "precision": 76.11, "recall": 78.55, "f1_score": 81.33},
        "CNN + LSTM (Hybrid)":{"accuracy": 85.00, "precision": 78.00, "recall": 80.00,
"f1_score": 82.00},
        "Ensemble Stacking":{"accuracy": 87.00, "precision": 80.00, "recall": 83.00,
"f1_score": 84.00},
        "Hybrid Model\n(NB+SVM+BLSTM)": {"accuracy": 98.10, "precision": 91.70,
"recall": 88.23, "f1_score": 90.67},
    }
    metrics = ["accuracy", "precision", "recall", "f1_score"]
    model_names = list(previous_models.keys())
    colors = ['#d9534f', '#f0ad4e', '#5bc0de', '#5cb85c']
    for metric in metrics:
        scores = [previous_models[model][metric] for model in model_names]
        plt.figure(figsize=(10, 5))
        bars = plt.bar(model_names, scores, color=colors)
        plt.title(f'{metric.capitalize()} Comparison', fontsize=14)
```

```
        plt.ylabel(f'{metric.capitalize()} (%)', fontsize=12)
        plt.ylim(0, 100)
        for bar in bars:
            yval = bar.get_height()
            plt.text(bar.get_x() + bar.get_width() / 2.0, yval + 1, f'{yval:.2f}%', ha='center',
va='bottom', fontsize=10)
        plt.grid(axis='y', linestyle='--', alpha=0.7)
        plt.tight_layout()
        plt.show()

def main():
    ctk.set_appearance_mode("dark")
    ctk.set_default_color_theme("blue")
    X, y, label_encoder, class_names, y_raw = preprocess_data(df)
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    pca = PCA(n_components=50)
    X_pca = pca.fit_transform(X_scaled)
    train_columns = X.columns
    X_train1, X_test1, y_train1, y_test1 = train_test_split(X_pca, y, test_size=0.3,
random_state=42)

    layer1_model = GaussianNB()
    layer1_model.fit(X_train1, y_train1)
    y_pred1 = layer1_model.predict(X_test1)
    y_pred1_labels = label_encoder.inverse_transform(y_pred1)
    dos_count = df[df['label'] == 'dos'].shape[0]
    probe_count = df[df['label'] == 'probe'].shape[0]

# Alert system
    root = tk.Tk()
    root.withdraw()
    engine = pyttsx3.init()
    engine.say("Alert. Attacks detected. DoS and Probe.")
```

```
    engine.runAndWait()
    messagebox.showinfo("Naive Bayes Detection Summary",
                        f"Attacks Detected:\n\nDoS: {dos_count}\nProbe: {probe_count}")

    print(f"\nTotal number of DoS attacks identified: {dos_count}")
    print(f"Total number of Probe attacks identified: {probe_count}")
    print(classification_report(y_test1, y_pred1, labels=range(len(label_encoder.classes_)),
target_names=label_encoder.classes_))
    plot_confusion_matrix(y_test1, y_pred1, "Naive Bayes")
    plot_metrics_bar(y_test1, y_pred1, "Naive Bayes")

    correct_indices = np.where(y_pred1 == y_test1)[0]
    X_correct = X_test1[correct_indices]
    y_correct = y_test1[correct_indices]

    X_train2, X_test2, y_train2, y_test2 = train_test_split(X_correct, y_correct, test_size=0.3,
random_state=42)
    svm_model = svm.SVC()
    svm_model.fit(X_train2, y_train2)
    y_pred2_svm = svm_model.predict(X_test2)
    y_pred2_svm_labels = label_encoder.inverse_transform(y_pred2_svm)

    print(classification_report(y_test2, y_pred2_svm,
labels=range(len(label_encoder.classes_)), target_names=label_encoder.classes_))
    plot_confusion_matrix(y_test2, y_pred2_svm, "SVM")
    plot_metrics_bar(y_test2, y_pred2_svm, "SVM")

    X_train2_blstm = np.reshape(X_train2, (X_train2.shape[0], 1, X_train2.shape[1]))
    X_test2_blstm = np.reshape(X_test2, (X_test2.shape[0], 1, X_test2.shape[1]))
    y_train2_cat = to_categorical(y_train2, num_classes=22)
    y_test2_cat = to_categorical(y_test2, num_classes=22)
    blstm_model = build_blstm_model(X_train2.shape[1])
    history = blstm_model.fit(X_train2_blstm, y_train2_cat, epochs=10, batch_size=64,
validation_data=(X_test2_blstm, y_test2_cat), verbose=1)
```

```
    y_pred2_blstm = blstm_model.predict(X_test2_blstm)
    y_pred2_blstm_classes = np.argmax(y_pred2_blstm, axis=1)
    y_pred2_blstm_labels = label_encoder.inverse_transform(y_pred2_blstm_classes)
    r2l_count = df[df['label'] == 'r2l'].shape[0]
    u2r_count = df[df['label'] == 'u2r'].shape[0]

    engine.say("R2L and U2R attacks detected.")
    engine.runAndWait()

    messagebox.showinfo("Layer 2 - Detection Summary",
                        f"Total Attacks Detected:\n\nR2L: {r2l_count}\nU2R: {u2r_count}")

    print(f"\n✅ Total number of R2L attacks identified in dataset: {r2l_count}")
    print(f"✅ Total number of U2R attacks identified in dataset: {u2r_count}")

    print(classification_report(y_test2, y_pred2_blstm_classes,
labels=range(len(label_encoder.classes_)), target_names=label_encoder.classes_))
    plot_confusion_matrix(y_test2, y_pred2_blstm_classes, "BLSTM")
    plot_performance_graph(history, "BLSTM")

    test_data = pd.read_csv('KDDTest+.txt', names=columns)

# Map only known attack types and drop unknown ones
    test_data['label'] = test_data['attack'].map(lambda r: attacks_types.get(r.strip()))
    test_data = test_data.dropna(subset=['label'])  # drop rows where label is NaN

# Continue as before
    X_test_all, y_test_all, y_test_labels = preprocess_test_data(test_data, scaler, pca,
label_encoder, train_columns)



    y_pred_test1 = layer1_model.predict(X_test_all)
    y_pred_test1_labels = label_encoder.inverse_transform(y_pred_test1)
```

```
    acc_nb = accuracy_score(y_test_all, y_pred_test1)
    print(classification_report(y_test_all,y_pred_test1,
labels=range(len(label_encoder.classes_)), target_names=label_encoder.classes_))
    plot_confusion_matrix(y_test_all, y_pred_test1, "Naive Bayes - Test Data")
    plot_metrics_bar(y_test_all, y_pred_test1, "Naive Bayes - Test Data")
    y_pred_test2_svm = svm_model.predict(X_test_all)
    y_pred_test2_svm_labels = label_encoder.inverse_transform(y_pred_test2_svm)



    acc_svm = accuracy_score(y_test_all, y_pred_test2_svm)
    print(classification_report(y_test_all,y_pred_test2_svm,
labels=range(len(label_encoder.classes_)), target_names=label_encoder.classes_))
    plot_confusion_matrix(y_test_all, y_pred_test2_svm, "SVM - Test Data")
    plot_metrics_bar(y_test_all, y_pred_test2_svm, "SVM - Test Data")

    X_test_all_blstm = np.reshape(X_test_all, (X_test_all.shape[0], 1, X_test_all.shape[1]))
    y_pred_test2_blstm = blstm_model.predict(X_test_all_blstm)
    y_pred_test2_blstm_classes = np.argmax(y_pred_test2_blstm, axis=1)
    y_pred_test2_blstm_labels                                                    =
label_encoder.inverse_transform(y_pred_test2_blstm_classes)

    acc_blstm = accuracy_score(y_test_all, y_pred_test2_blstm_classes)
    print(classification_report(y_test_all,y_pred_test2_blstm_classes,
labels=range(len(label_encoder.classes_)), target_names=label_encoder.classes_))
    plot_confusion_matrix(y_test_all, y_pred_test2_blstm_classes, "BLSTM - Test Data")

    plot_final_accuracy_comparison(acc_nb, acc_svm, acc_blstm)
    plot_comparison_graphs()

if _name_ == "_main_":
    main()
```

## 4.2 IMPLEMENTATION

- Detection.py
- KDDTrain+.txt
- KDDTest+.txt

## 4.3 INSTALLATION OF PACKAGES

**Data Manipulation & Analysis**

pandas (import pandas as pd)

Used for data manipulation and analysis. Provides DataFrame objects and operations for manipulating numerical tables and time series.

**numpy** (import numpy as np)

A core package for scientific computing. Supports large, multi-dimensional arrays and matrices, along with a collection of mathematical functions.

**Data Visualization**

matplotlib.pyplot (import matplotlib.pyplot as plt)

A popular 2D plotting library to create static, interactive, and animated visualizations in Python.

**seaborn (import seaborn as sns)**

Based on matplotlib, it provides a high-level interface for drawing attractive and informative statistical graphics.

**Machine Learning & Preprocessing** (scikit-learn) train_test_split

Splits arrays or matrices into random train and test subsets.

**StandardScaler & LabelEncoder**

Used for feature scaling and converting categorical labels into numbers.

**GaussianNB**

Implements the Gaussian Naive Bayes classification algorithm.

**svm**

Support Vector Machines for classification, regression, and outlier detection.

**Metrics** (classification_report, confusion_matrix, etc.) Tools to evaluate the performance of classification models.

**Deep Learning** (TensorFlow & Keras) Sequential, Dense, LSTM, Bidirectional
Components from Keras (via TensorFlow) used to build and train deep learning models, especially neural networks like LSTM.

**Text-to-Speech & Sound**

pyttsx3
A text-to-speech conversion library in Python. It works offline and is platform-independent.

**winsound**

Provides access to sound-playing machinery in Windows.

**Utilities**

Counter from collections
A dictionary subclass for counting hashable objects. Useful for frequency analysis.

**GUI** (Graphical User Interface) tkinter and tkinter.messagebox
Standard GUI toolkit in Python. Used to build desktop applications with graphical interfaces.

**customtkinter (import customtkinter as ctk)**

An enhanced version of Tkinter that allows for modern-looking, customizable widgets.

**Dataset:NSL-KDD**

The NSL-KDD dataset is a refined version of the original KDD Cup 1999 dataset, which was developed for evaluating intrusion detection systems (IDS). The original dataset had several issues like redundancy and imbalance, which NSL-KDD addresses.

NSL-KDD stands for Network Security Laboratory - Knowledge Discovery in Databases. It

is used for benchmarking machine learning models in network intrusion detection.

Each data record represents a network connection and includes:

- 41 features:
  - Basic features (e.g., duration, protocol, service)
  - Content features (e.g., number of failed logins)
  - Traffic features (e.g., count, srv_count)
- Label:
  - Either normal or an attack type

### 4.4 SET UP PYTHON ENVIRONMENT

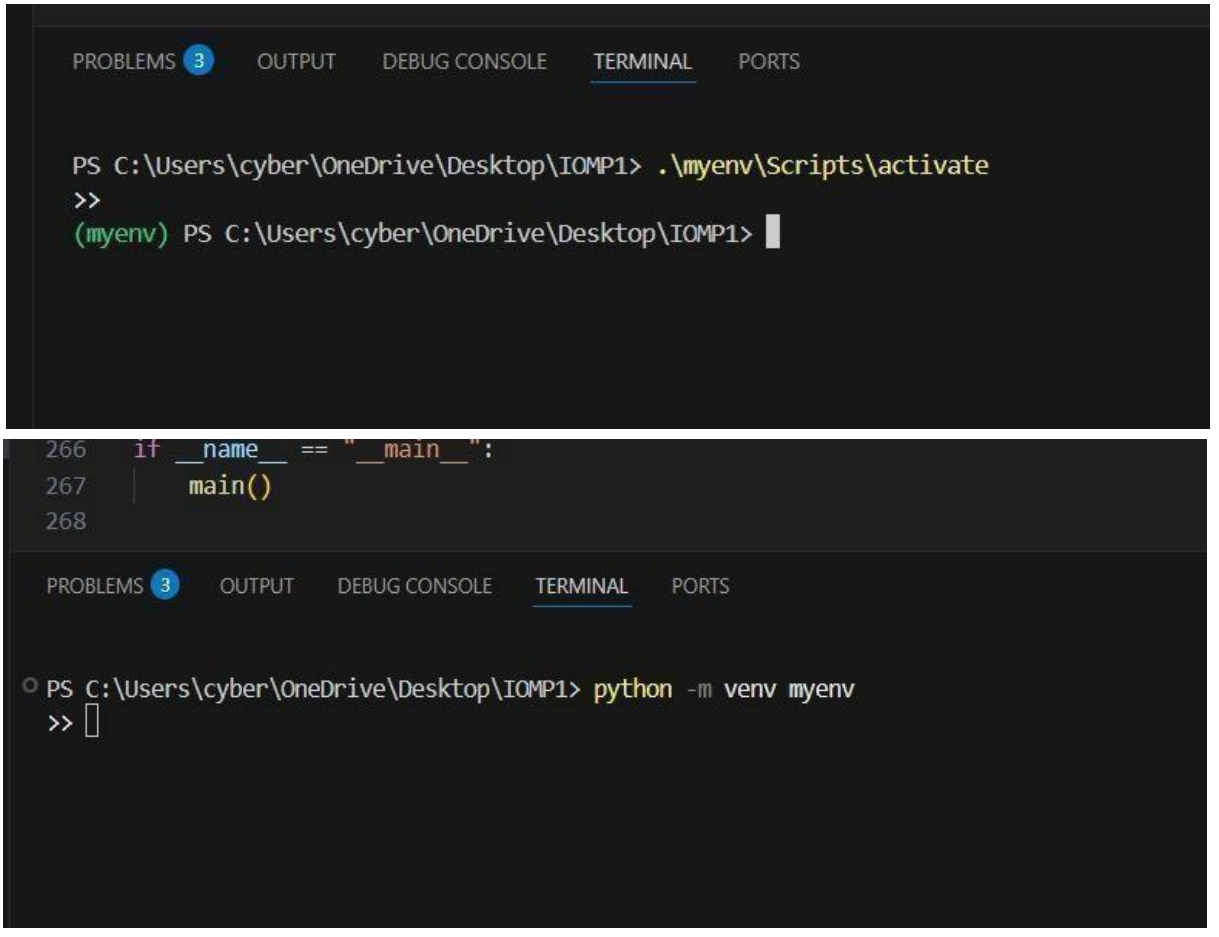



Fig:4.1 set up python environment

# 5.TESTING

## 5.1 INTRODUCTION TO TESTING

In the context of this project, testing plays a critical role in evaluating the effectiveness and robustness of the hybrid intrusion detection system built using Naive Bayes (NB), Support Vector Machine (SVM), and Bidirectional Long Short-Term Memory (BLSTM). The testing phase is performed using the KDDTest+ dataset, which is a standardized and widely used benchmark for intrusion detection system (IDS) evaluation.

- The testing process aims to:
- Validate model generalization on unseen data.
- Measure detection accuracy across different attack categories (DoS, Probe, R2L, U2R).
- Identify strengths and limitations of each individual classifier (NB, SVM, BLSTM) and the overall hybrid architecture.
- Quantify performance using standard metrics such as accuracy, precision, recall, F1- score, and confusion matrices.
- Trigger alerts and generate insights into attack patterns detected.

**Each layer of the hybrid model is tested in sequence:**

Layer 1 (Naive Bayes) detects broad attack categories like DoS and Probe.

Layer 2 (SVM + BLSTM) further analyzes correctly predicted data to detect more sophisticated attacks such as R2L (Remote to Local) and U2R (User to Root).

The predictions are then compared to the ground truth from the test set to compute In addition, an alert system using speech and pop-up messages is implemented to simulate real-time notification of detected intrusions. The test results also provide a comparative analysis with traditional deep learning models, showcasing the efficiency of the proposed hybrid approach in identifying various network intrusions effectively.

## 5.2 TEST CASES:

Table 5.1 Test Cases of Intrusion detection

| Test case | Model | Data Source | What's Tested | Pass criteria |
|---|---|---|---|---|
| TC1 | Naive Bayes (Layer 1) | 30% hold-out split of X_pca/y from training set (X_test1, y_test1) | Layer 1 classification performance | Model returns predictions on test set |
| TC2 | SVM (Layer 2) | 30% hold-out split of the *correctly* classified subset (X_test2, y_test2) | Layer 2 (SVM) classification on "easy" cases | No errors in prediction pipeline |
| TC3 | BLSTM (Layer 2) | 30% hold-out split of the same "easy" subset, used as validation during training | BLSTM training & validation accuracy | Loss decreases, accuracy improves per epoch |
| TC4 | Navie Bayes | Entire external test file (KDDTest+.txt) preprocessed → X_test_all, y_test_all | Final NB performance on unseen data | Passed and gives confusion matrix |
| TC5 | SVM(Test Data) | Same external test set (X_test_all, y_test_all) | Final SVM performance on unseen data | Passed and gives confusion matrix |
| TC6 | BLSTM (Test Data) | Same external test set reshaped for BLSTM | Final BLSTM performance on unseen data | Passed and gives confusion matrix |

# 6.RESULTS

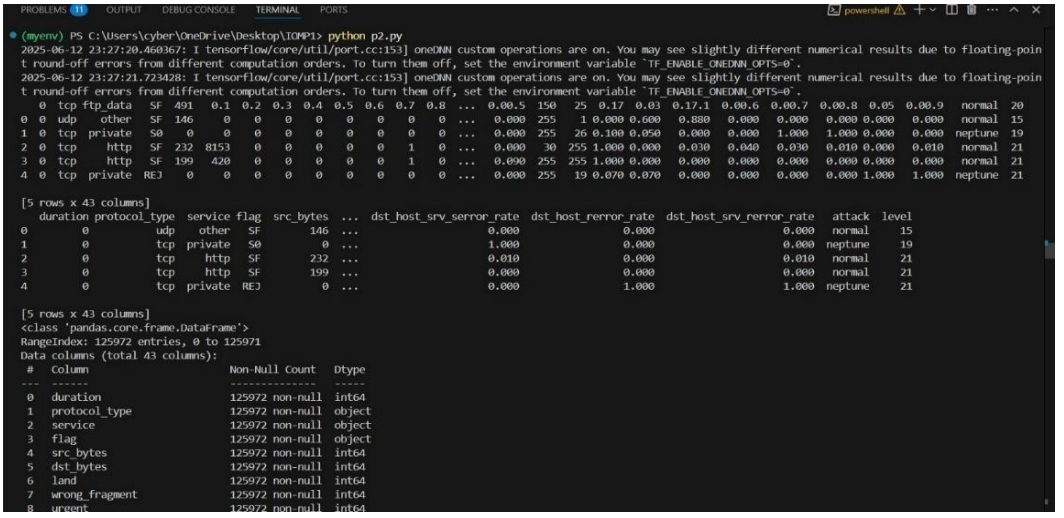



Fig.6.1 columns of the dataset NSL-KDD. This figure displays the first few rows of the NSL-KDD dataset along with its column headers.

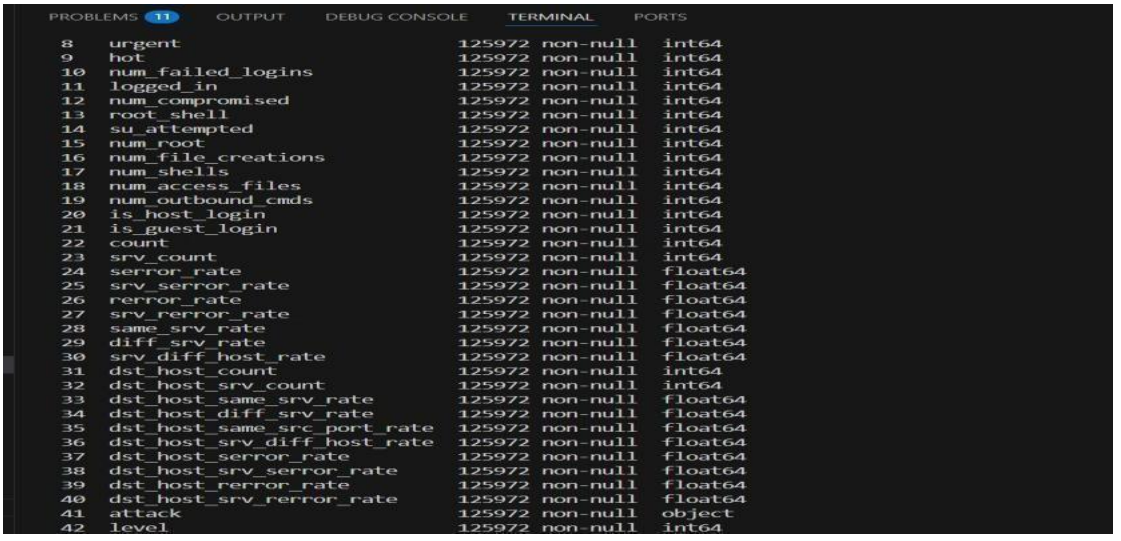



Fig.6.2 datatype of each column

This figure presents the data types of all columns in the NSL-KDD dataset. It shows that the dataset includes both numeric (e.g., int64, float64) and categorical (object) data type

```
 42  level                        125972 non-null  int64
dtypes: float64(15), int64(24), object(4)
memory usage: 41.3+ MB
None
                          count      mean          std  min    25%    50%     75%           max
duration                125972.000   287.147    2604.526 0.000  0.000  0.000   0.000     42908.000
src_bytes               125972.000 45567.101 5870354.481 0.000  0.000 44.000 276.000 1379963888.000
dst_bytes               125972.000 19779.271 4021285.112 0.000  0.000  0.000 516.000 1309937401.000
land                    125972.000     0.000       0.014 0.000  0.000  0.000   0.000          1.000
wrong_fragment          125972.000     0.023       0.254 0.000  0.000  0.000   0.000          3.000
urgent                  125972.000     0.000       0.014 0.000  0.000  0.000   0.000          3.000
hot                     125972.000     0.204       2.150 0.000  0.000  0.000   0.000         77.000
num_failed_logins       125972.000     0.001       0.045 0.000  0.000  0.000   0.000          5.000
logged_in               125972.000     0.396       0.489 0.000  0.000  0.000   1.000          1.000
num_compromised         125972.000     0.279      23.942 0.000  0.000  0.000   0.000       7479.000
root_shell              125972.000     0.001       0.037 0.000  0.000  0.000   0.000          1.000
su_attempted            125972.000     0.001       0.045 0.000  0.000  0.000   0.000          2.000
num_root                125972.000     0.302      24.400 0.000  0.000  0.000   0.000       7468.000
num_file_creations      125972.000     0.013       0.484 0.000  0.000  0.000   0.000         43.000
num_shells              125972.000     0.000       0.022 0.000  0.000  0.000   0.000          2.000
num_access_files        125972.000     0.004       0.099 0.000  0.000  0.000   0.000          9.000
num_outbound_cmds       125972.000     0.000       0.000 0.000  0.000  0.000   0.000          0.000
is_host_login           125972.000     0.000       0.003 0.000  0.000  0.000   0.000          1.000
is_guest_login          125972.000     0.009       0.097 0.000  0.000  0.000   0.000          1.000
count                   125972.000    84.108     114.509 0.000  2.000 14.000 143.000        511.000
srv_count               125972.000    27.738      72.636 0.000  2.000  8.000  18.000        511.000
serror_rate             125972.000     0.284       0.446 0.000  0.000  0.000   1.000          1.000
srv_serror_rate         125972.000     0.282       0.447 0.000  0.000  0.000   1.000          1.000
rerror_rate             125972.000     0.120       0.320 0.000  0.000  0.000   0.000          1.000
srv_rerror_rate         125972.000     0.121       0.324 0.000  0.000  0.000   0.000          1.000
same_srv_rate           125972.000     0.661       0.440 0.000  0.090  1.000   1.000          1.000
diff_srv_rate           125972.000     0.063       0.180 0.000  0.000  0.000   0.060          1.000
srv_diff_host_rate      125972.000     0.097       0.260 0.000  0.000  0.000   0.000          1.000
dst_host_count          125972.000   182.149      99.207 0.000 82.000 255.000 255.000        255.000
dst_host_srv_count      125972.000   115.654     110.703 0.000 10.000  63.000 255.000        255.000
```

Fig.6.3 description of each column values.

This figure provides a statistical summary of all numerical columns in the NSL-KDD dataset. It includes metrics such as count, mean, standard deviation, minimum, maximum.

```
PROBLEMS 11   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

dst_host_same_srv_rate          0
dst_host_diff_srv_rate          0
dst_host_same_src_port_rate     0
dst_host_srv_diff_host_rate     0
dst_host_serror_rate            0
dst_host_srv_serror_rate        0
dst_host_rerror_rate            0
dst_host_srv_rerror_rate        0
attack                          0
level                           0
dtype: int64
Column: protocol_type
------------------------------
Unique Values (3): ['udp' 'tcp' 'icmp']

Value Counts:
protocol_type
tcp     102688
udp      14993
icmp      8291
Name: count, dtype: int64
======================================

Column: service
------------------------------
Unique Values (70): ['other' 'private' 'http' 'remote_job' 'ftp_data' 'name' 'netbios_ns'
 'eco_i' 'mtp' 'telnet' 'finger' 'domain_u' 'supdup' 'uucp_path' 'Z39_50'
 'smtp' 'csnet_ns' 'uucp' 'netbios_dgm' 'urp_i' 'auth' 'domain' 'ftp'
 'bgp' 'ldap' 'ecr_i' 'gopher' 'vmnet' 'systat' 'http_443' 'efs' 'whois'
 'imap4' 'iso_tsap' 'echo' 'klogin' 'link' 'sunrpc' 'login' 'kshell'
 'sql_net' 'time' 'hostnames' 'exec' 'ntp_u' 'discard' 'nntp' 'courier'
 'ctf' 'ssh' 'daytime' 'shell' 'netstat' 'pop_3' 'nnsp' 'IRC' 'pop_2'
 'printer' 'tim_i' 'pm_dump' 'red_i' 'netbios_ssn' 'rje' 'X11' 'urh_i'
 'http_8001' 'aol' 'http_2784' 'tftp_u' 'harvest']
```

Fig.6.4 unique values and value counts of protocols.

This figure displays the distinct protocol types present in the NSL-KDD dataset along with their respective frequencie

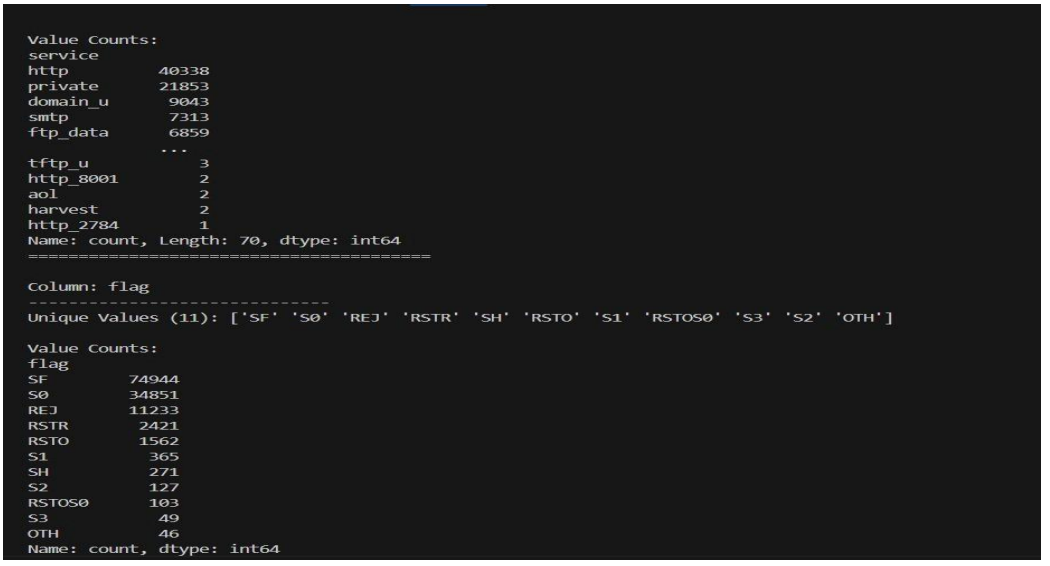



Fig.6.5 value count of the service and flag column.
This figure illustrates the frequency distribution of the service and flag columns in the NSL-KDD dataset.

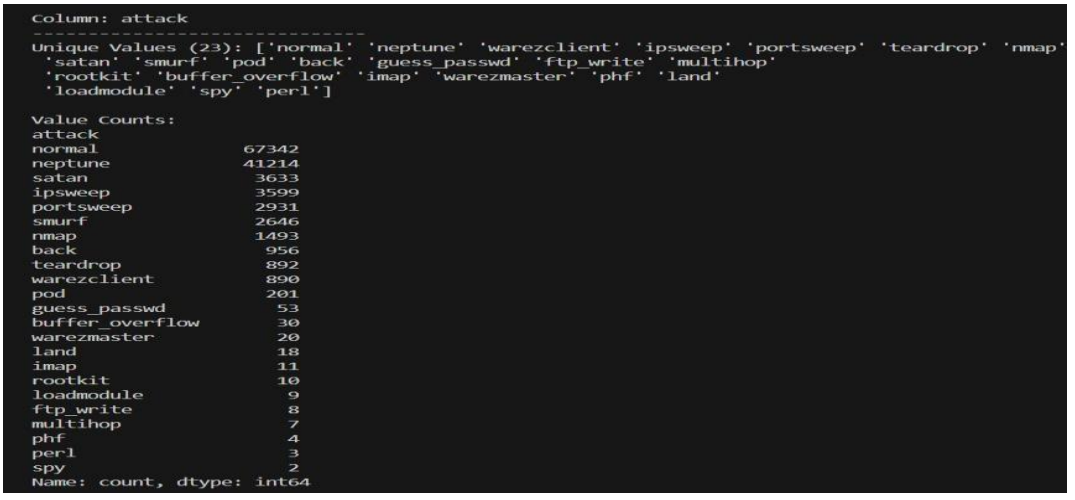
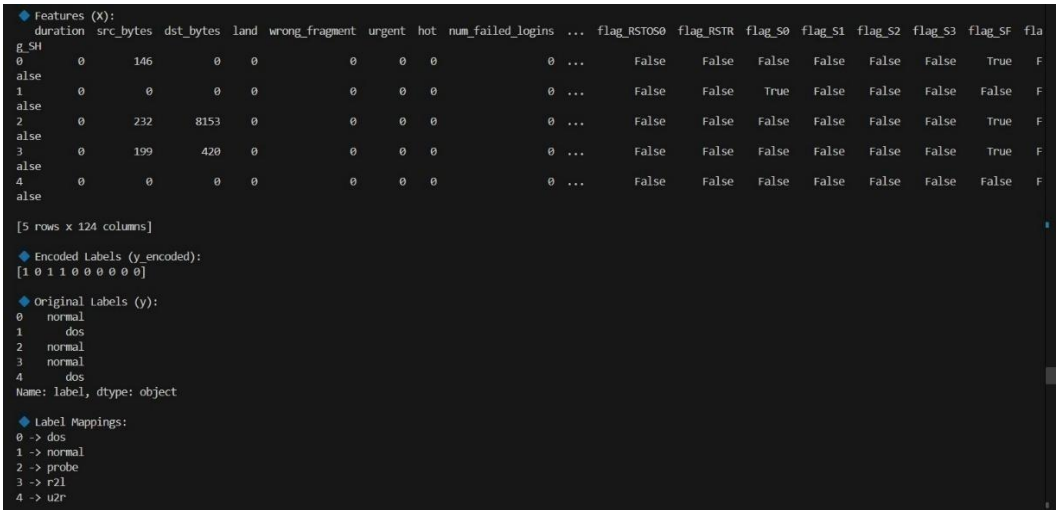
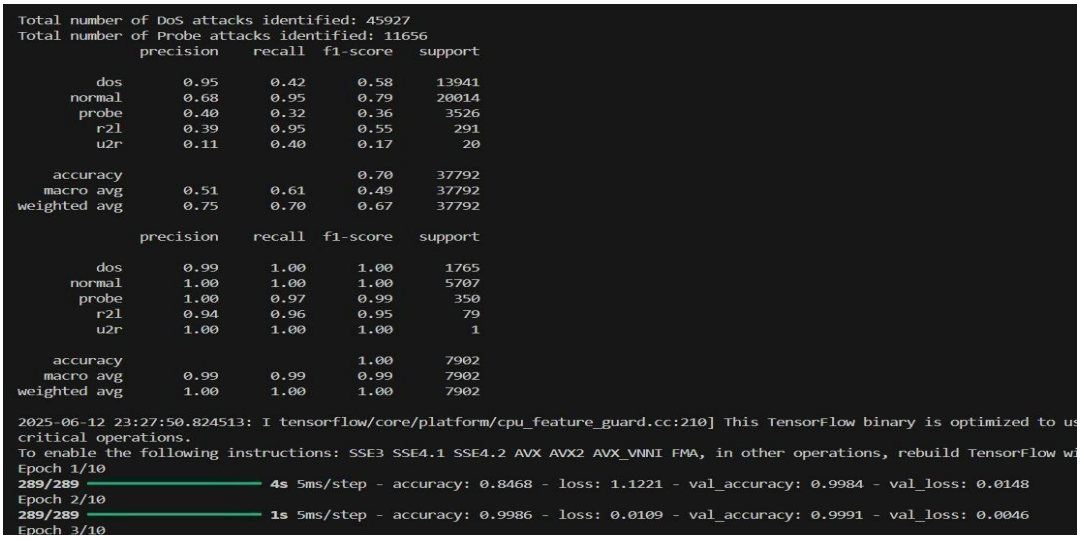



Fig.6.6 value countss of the coulumn-attack.
This figure displays the distribution of different attack types and normal connections within the attack column of the NSL-KDD dataset



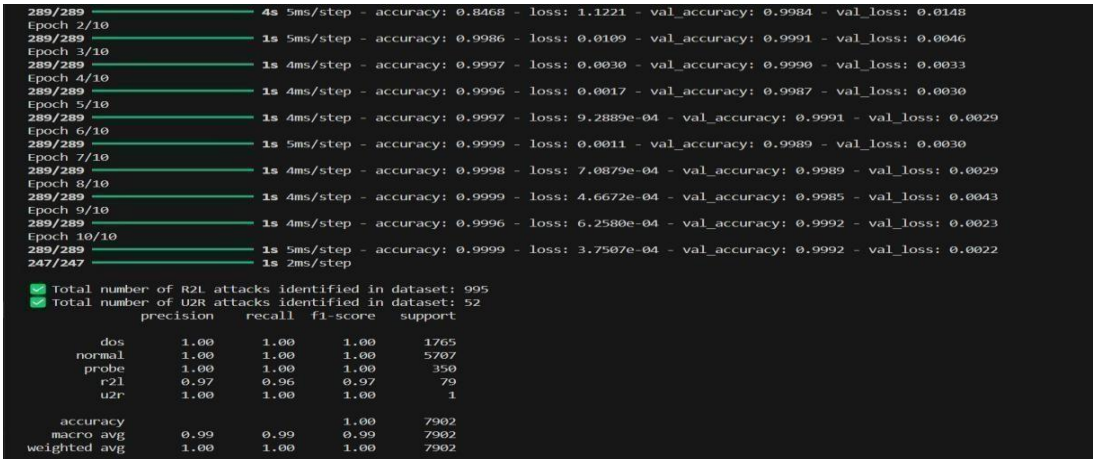
```
🔷 Features (X):
   duration  src_bytes  dst_bytes  land  wrong_fragment  urgent  hot  num_failed_logins  ...  flag_RSTOS0  flag_RSTR  flag_S0  flag_S1  flag_S2  flag_S3  flag_SF  fla
g_SH
0         0        146          0     0               0       0    0                  0  ...        False      False    False    False    False    False     True    F
alse
1         0          0          0     0               0       0    0                  0  ...        False      False     True    False    False    False    False    F
alse
2         0        232       8153     0               0       0    0                  0  ...        False      False    False    False    False    False     True    F
alse
3         0        199        420     0               0       0    0                  0  ...        False      False    False    False    False    False     True    F
alse
4         0          0          0     0               0       0    0                  0  ...        False      False    False    False    False    False    False    F
alse

[5 rows x 124 columns]

🔷 Encoded Labels (y_encoded):
[1 0 1 1 0 0 0 0 0 0]

🔷 Original Labels (y):
0    normal
1       dos
2    normal
3    normal
4       dos
Name: label, dtype: object

🔷 Label Mappings:
0 -> dos
1 -> normal
2 -> probe
3 -> r2l
4 -> u2r
```

Fig.6.7 PCA feature selection and encoding.



```
Total number of DoS attacks identified: 45927
Total number of Probe attacks identified: 11656
              precision    recall  f1-score   support

         dos       0.95      0.42      0.58     13941
      normal       0.68      0.95      0.79     20014
       probe       0.40      0.32      0.36      3526
         r2l       0.39      0.95      0.55       291
         u2r       0.11      0.40      0.17        20

    accuracy                           0.70     37792
   macro avg       0.51      0.61      0.49     37792
weighted avg       0.75      0.70      0.67     37792

              precision    recall  f1-score   support

         dos       0.99      1.00      1.00      1765
      normal       1.00      1.00      1.00      5707
       probe       1.00      0.97      0.99       350
         r2l       0.94      0.96      0.95        79
         u2r       1.00      1.00      1.00         1

    accuracy                           1.00      7902
   macro avg       0.99      0.99      0.99      7902
weighted avg       1.00      1.00      1.00      7902

2025-06-12 23:27:50.824513: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to us
critical operations.
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow wi
Epoch 1/10
289/289 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8468 - loss: 1.1221 - val_accuracy: 0.9984 - val_loss: 0.0148
Epoch 2/10
289/289 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9986 - loss: 0.0109 - val_accuracy: 0.9991 - val_loss: 0.0046
Epoch 3/10
```

Fig.6.8 Navie Bayes and Svm performance metrics.



```
289/289 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8468 - loss: 1.1221 - val_accuracy: 0.9984 - val_loss: 0.0148
Epoch 2/10
289/289 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9986 - loss: 0.0109 - val_accuracy: 0.9991 - val_loss: 0.0046
Epoch 3/10
289/289 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9997 - loss: 0.0030 - val_accuracy: 0.9990 - val_loss: 0.0033
Epoch 4/10
289/289 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9996 - loss: 0.0017 - val_accuracy: 0.9987 - val_loss: 0.0030
Epoch 5/10
289/289 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9997 - loss: 9.2889e-04 - val_accuracy: 0.9991 - val_loss: 0.0029
Epoch 6/10
289/289 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9999 - loss: 0.0011 - val_accuracy: 0.9989 - val_loss: 0.0030
Epoch 7/10
289/289 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9998 - loss: 7.0879e-04 - val_accuracy: 0.9989 - val_loss: 0.0029
Epoch 8/10
289/289 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9999 - loss: 4.6672e-04 - val_accuracy: 0.9985 - val_loss: 0.0043
Epoch 9/10
289/289 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9996 - loss: 6.2580e-04 - val_accuracy: 0.9992 - val_loss: 0.0023
Epoch 10/10
289/289 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9999 - loss: 3.7507e-04 - val_accuracy: 0.9992 - val_loss: 0.0022
247/247 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step

✅ Total number of R2L attacks identified in dataset: 995
✅ Total number of U2R attacks identified in dataset: 52
              precision    recall  f1-score   support

         dos       1.00      1.00      1.00      1765
      normal       1.00      1.00      1.00      5707
       probe       1.00      1.00      1.00       350
         r2l       0.97      0.96      0.97        79
         u2r       1.00      1.00      1.00         1

    accuracy                           1.00      7902
   macro avg       0.99      0.99      0.99      7902
weighted avg       1.00      1.00      1.00      7902
```

Fig.6.9 BLSTM performance metrics.

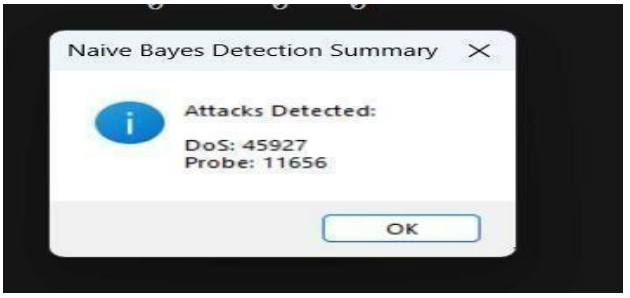



Fig.6.10 Message box at navie bayes showing total number of dos and probe attacks

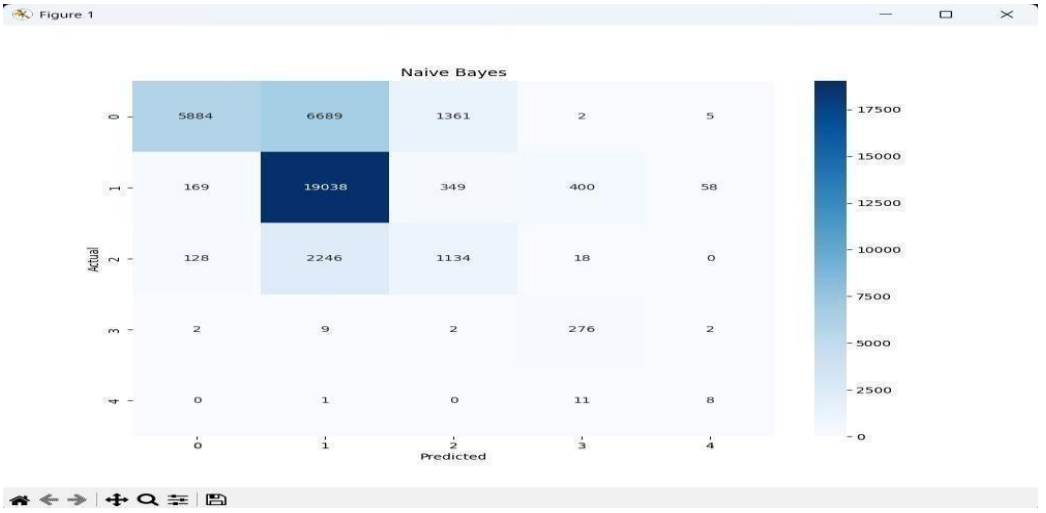



Fig.6.11 Confusion matrix off Navie Bayes

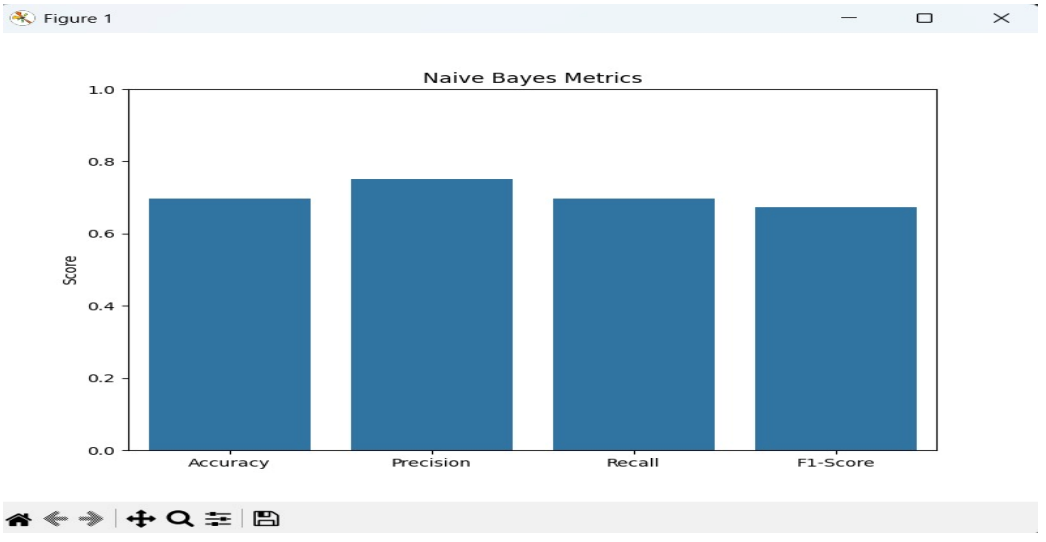



Fig.6.12 Bar graph of performance metrics at Navie Bayes
This figure illustrates the performance of the Naive Bayes classifier using four key evaluation metrics: Accuracy, Precision, Recall, and F1-Score.

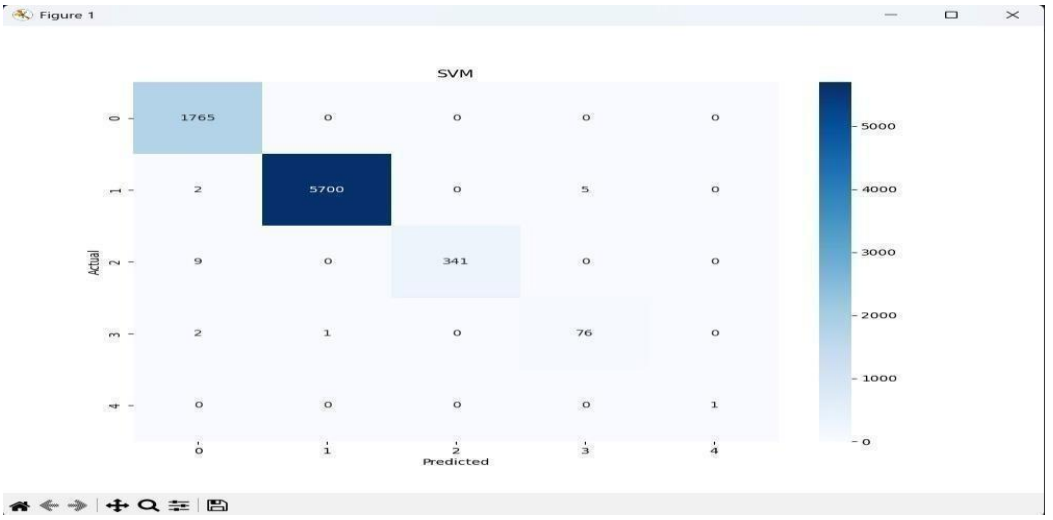



Fig.6.13 Confusion Matrix of SVM

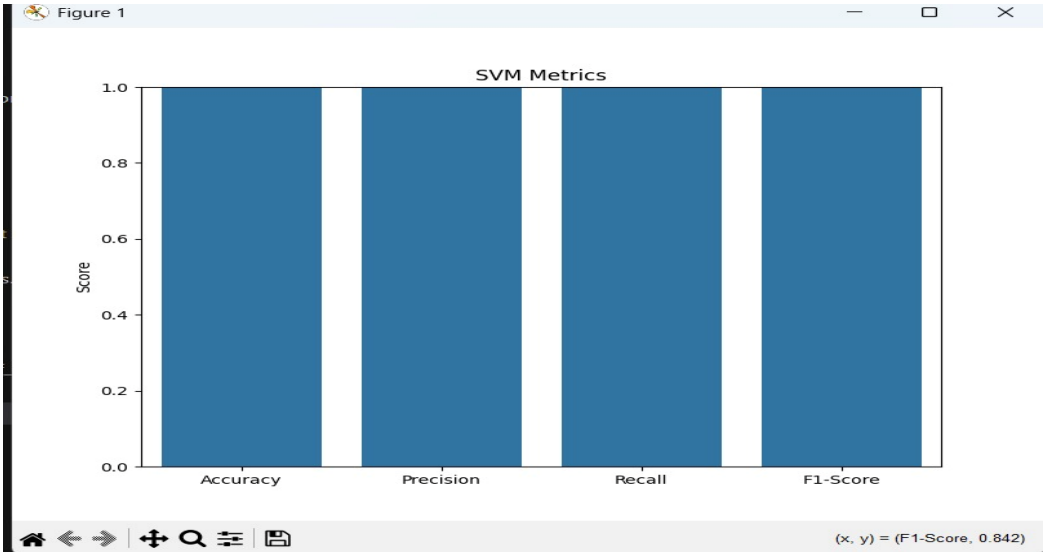



Fig.6.14 Bar graph of performance metrics at SVM


Fig.6.15 Message box after svm and blstm showing attacks identified R2L and U2R.
This figure displays a pop-up message box generated after the execution of the SVM and BLSTM layers in the hybrid intrusion detection model.

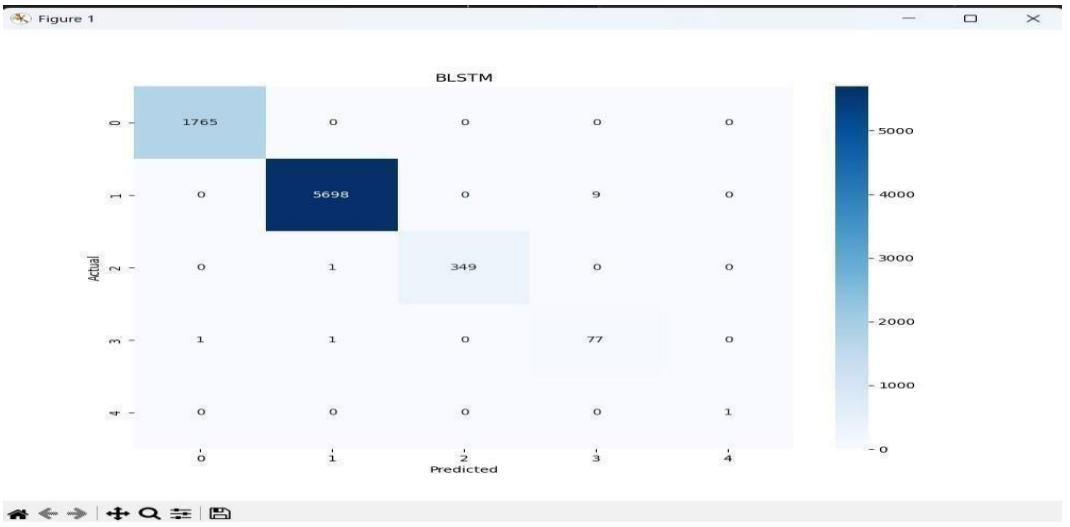



Fig.6.16 BLSTM confusion matrix

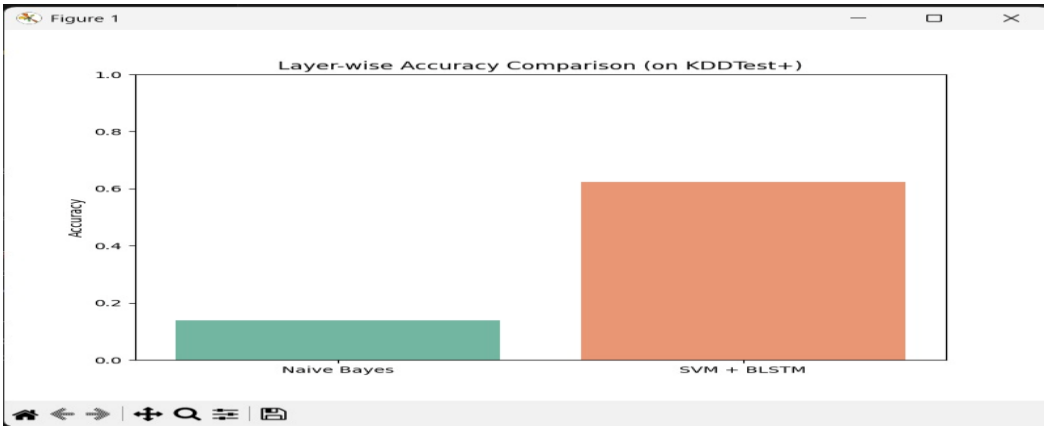



Fig.6.17 Accuracy of navie bayes and svm+blstm in finding attacks.

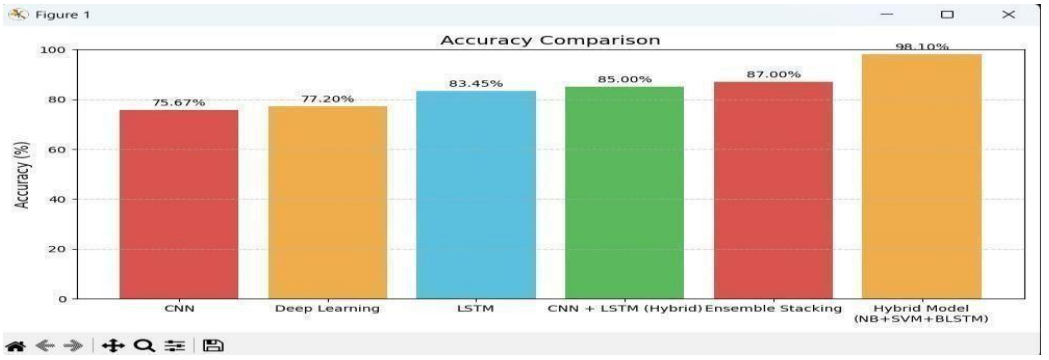



Fig.6.18 Accuracy comparison with previous models.

This figure presents a bar graph comparing the accuracy of the proposed hybrid model (NB + SVM + BLSTM) with previously used models such as CNN, LSTM, CNN+LSTM, and Ensemble Stacking.

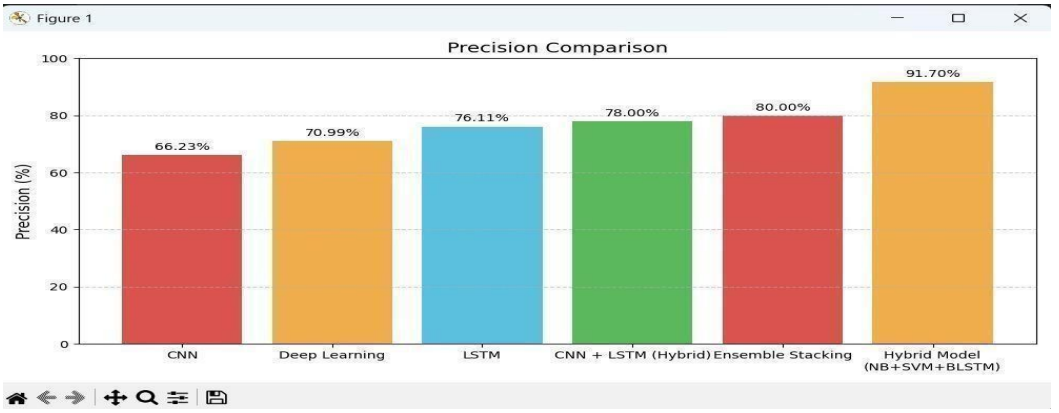



Fig.6.19 Precision comparison with previous models


Fig.6.20 Recall comparison with previous models

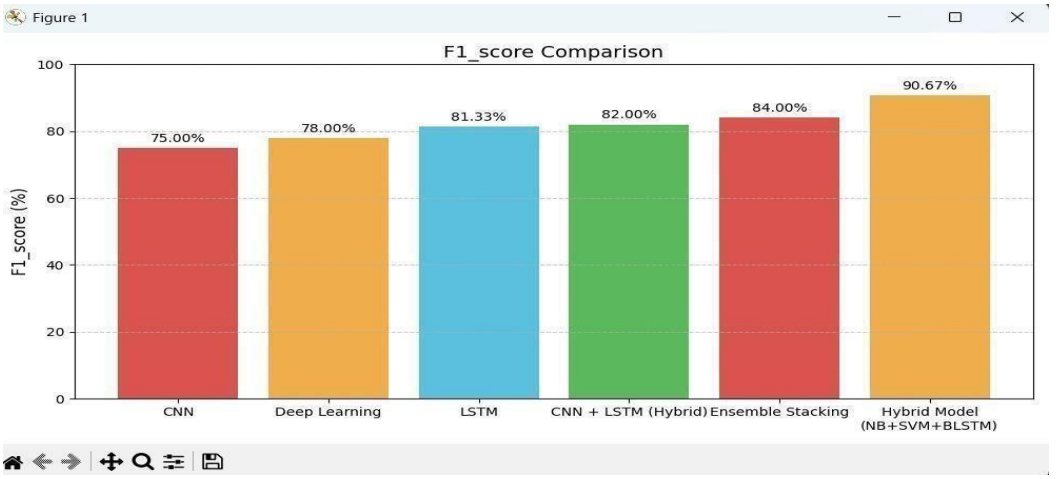



Fig.6.21 F1-score comparison with previous models

This figure illustrates the F1-score performance across various intrusion detection models.

# 7.CONCLUSION AND FUTURE ENHANCEMENTS

## 7.1 CONCLUSION

Enhanced Detection Accuracy: The proposed Double Layered Hybrid Approach (DLHA), combining Naive Bayes, SVM, and BLSTM, significantly improves detection accuracy for both frequent (DoS, Probe) and rare (R2L, U2R) attack types compared to traditional methods.

Effective Use of PCA and BLSTM: The use of Principal Component Analysis (PCA) for dimensionality reduction and BLSTM for temporal pattern recognition enables the system to handle complex, sequential network traffic more efficiently.

Robust and Scalable Solution: DLHA provides a robust, scalable, and adaptable framework for real-time intrusion detection, making it suitable for modern dynamic network environments facing evolving cyber threats.

## 7.2 FUTURE ENHANCEMENTS

- **Real-time detection** – Upgrade to monitor live network traffic.
- **System integration** – Connect with firewalls and SIEM tools.
- **Adaptive learning** – Enable the model to learn from new attacks.
- **IoT deployment** – Optimize for use in smart and edge devices.
- **Multi-source analysis** – Use logs and external data for better accuracy.

# REFERENCES

[1] Mohammadi, M., Navaras, A., & Amini, M. H. (2021). A Double Layered Hybrid Approach for Network Intrusion Detection. IEEE Xplore.Combines Naive Bayes, SVM for improved intrusion detection.

[2] Shi, S., Han, D., & Cui, M. (2023). Multimodal Hybrid Parallel IDS Using ML and DL Models. Connection Science, Taylor & Francis.Focuses on feature extraction and classification in high-dimensional network traffic.

[3] Sajid, M., et al. (2024). Hybrid Machine and Deep Learning Approach for Intrusion Detection in Cloud Environments. Journal of Cloud Computing,Springer.Integrates CNN, LSTM, and ensemble learning methods.

[4] Jalili, R., Imani, S., & Aminzadeh, M. R. (2021). CNN-LSTM Based Anomaly Detection in Software Defined Networks. ACM Digital Library.Applies deep learning to improve unknown attack detection and reduce false positives.

[5] Security and Communication Networks. (2024). A Supervised Ensemble Stacking Model for IDS. Hindawi Publishing Corporation.Combines multiple machine learning algorithms into a stacked ensemble for enhanced detection.

[6] Saleh, A. I., Talaat, F. M., & Labib, L. M. (2019). A Hybrid Intrusion Detection System (HIDS) Based on Prioritized k-Nearest Neighbors and Optimized SVM Classifiers. Artificial Intelligence Review, 51, 403–443.Proposes a hybrid IDS combining Genetic Algorithm (GA) and SVM, achieving high detection rates, particularly for rare attacks like R2L and U2R.

[7] Kushal, S., Shanmugam, B., Sundaram, J., & Thennadil, S. (2024). Self-Healing Hybrid

Intrusion Detection System: An Ensemble Machine Learning Approach. Discover Artificial Intelligence, 4(28).Introduces a self-healing IDS combining C5 classifiers and LSTM, enhancing detection of both known and unknown attacks through continual learning.

[8] Zhang, Y., Wang, Y., & Li, X. (2024). Multi-Class Intrusion Detection System in SDN Based on Hybrid BiLSTM with Attention Mechanism. Cluster Computing.Presents a hybrid BiLSTM model with attention mechanisms for multi- class intrusion detection in SDNs, achieving high accuracy in detecting various attack types.

[9] Khraisat, A., Gondal, I., Vamplew, P., & Kamruzzaman, J. (2020). Development of Hybrid Intrusion Detection System Leveraging One-Class SVM and C5 Decision Tree Classifier. Journal of Cybersecurity and Privacy.Develops a hybrid IDS combining One-Class SVM and C5 decision tree classifiers, effectively detecting both known intrusions and zero-day attacks.

[10] Farid, D. M., Harbi, N., & Rahman, M. Z. (2010). Combining Naive Bayes and DecisionTree for Adaptive Intrusion Detection. arXiv preprint arXiv:1005.4496