Deep Learning for Intrusion Detection in Reconfigurable Wireless Networks

A Main Project submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY
In
INFORMATION TECHNOLOGY

Submitted by

THOTA MERY SOWMYA	**21PA1A12B9**
SANGEPU TEJASWINI	**21PA1A12A8**
PEKKA KIRAN KUMAR	**21PA1A1288**
NERADABILLI NUKARAJU	**21PA1A1282**

Under the esteemed guidance of

Dr. Venkata Naga Rani Bandaru
Assistant Professor

DEPARTMENT OF INFORMATION TECHNOLOGY
VISHNU INSTITUTE OF TECHNOLOGY
(Autonomous)
(Approved by AICTE, Accredited by NBA & NAAC and permanently affiliated to JNTU Kakinada)
BHIMAVARAM – 534 202

2024 – 2025

Deep Learning for Intrusion Detection in Reconfigurable Wireless Networks

A Main Project submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY
In
INFORMATION TECHNOLOGY

Submitted by

THOTA MERY SOWMYA	21PA1A12B9
SANGEPU TEJASWINI	21PA1A12A8
PEKKA KIRAN KUMAR	21PA1A1288
NERADABILLI NUKARAJU	21PA1A1282

Under the esteemed guidance of

Dr. Venkata Naga Rani Bandaru
Assistant Professor

VISHNU
UNIVERSAL LEARNING

DEPARTMENT OF INFORMATION TECHNOLOGY
VISHNU INSTITUTE OF TECHNOLOGY
(Autonomous)
(Approved by AICTE, Accredited by NBA & NAAC and permanently affiliated to JNTU Kakinada)
BHIMAVARAM – 534 202

2024 – 2025

VISHNU INSTITUTE OF TECHNOLOGY

(Autonomous)

(Approved by AICTE, Accredited by NBA & NAAC and permanently affiliated to JNTU Kakinada)

BHIMAVARAM-534202

2024-2025

DEPARTMENT OF INFORMATION TECHNOLOGY

VISHNU
UNIVERSAL LEARNING

CERTIFICATE

This is to certify that the project entitled "DEEP LEARNING FOR INTRUSION DETECTION IN RECONFIGURABLE WIRELESS NETWORKS", is being submitted by *THOTA MERY SOWMYA, SANGEPU TEJASWINI, PEEKA KIRAN KUMAR and NERADABIL NUKARAJU*, bearing the REGD.NOS: 21PA1A12B9, 21PA1A12A8, 21PA1A1288 and 21PA1A1282 submitted in partial fulfilment for the award of the degree of **"**BACHELOR OF TECHNOLOGY" in "INFORMATION TECHNOLOGY" is a record of bonafide work carried out by them under my guidance and supervision during the academic year 2024-2025 and it has been found worthy of acceptance according to the requirements of university.

Internal Guide
Dr. Venkata Naga Rani Bandaru
Associate Professor

Head of the Department
Dr. D J Nagendra Kumar
Professor

External Examiner

ACKNOWLEDGEMENT

It is nature and inevitable that the thoughts and ideas of other people tend to drift in to the subconscious due to various human parameters, where one feels acknowledge the help and guidance derived from others. We acknowledge each of those who have contributed for the fulfilment of this project.

We take the opportunity to express our sincere gratitude to **Dr. M. Venu,** Principal, Vishnu Institute of Technology**,** Bhimavaram whose guidance from time to time helped us to complete this project successfully.

We are very much thankful to **Mrs. M. Srilakshmi**, Vice Principal and **Dr. D J Nagendra Kumar**, Head of the Department, INFORMATION TECHNOLOGY for his continuous and unrelenting support and guidance. We thank and acknowledge our gratitude to his valuable guidance and support expended to us right from the conception of the idea to the completion of this project.

We are very much thankful to **Dr. Venkata Naga Rani Bandaru**, Associate Professor, our internal guide whose guidance from time to time helped us to complete this project successfully.

Project Associates

Ms THOTA MERY SOWMYA	**21PA1A12B9**
Ms SANGEPU TEJASWINI	**21PA1A12A8**
Mr PEEKA KIRAN KUMAR	**21PA1A1288**
Mr NERADABIL NUKARAJU	**21PA1A1282**

ABSTRACT

With the rapid advancements in wireless communication technologies and the increasing deployment of Reconfigurable Wireless Networks (RWNs), new opportunities have arisen for designing flexible and scalable network architectures. RWNs, characterized by their reconfigurability at topological, frequency, and power levels, provide dynamic capabilities for adapting to a wide range of applications, including IoT, smart cities, and autonomous systems. However, these dynamic and heterogeneous networks also bring significant security challenges. The increased attack surface, combined with evolving cyber threats, necessitates robust intrusion detection mechanisms to ensure the integrity and reliability of RWNs. Traditional intrusion detection systems, which largely rely on rule-based methodologies, struggle to address the complexities of modern cyber-attacks, including zero-day threats, and fail to adapt to changes in attack strategies or network configurations. To address these limitations, this study proposes a deep learning-based intrusion detection system (DL-IDS) designed for both binary and multi-level attack classification. Leveraging the advanced capabilities of Convolutional Neural Networks (CNNs) and hybrid machine learning models, the system detects various cyber threats such as DoS attacks, spoofing, and unauthorized access. Optimized for resource-constrained environments, the DL-IDS ensures real-time detection and response, making it highly effective in critical and computationally intensive scenarios. This innovative framework represents a significant step forward in safeguarding next-generation wireless communication systems, offering a dynamic and scalable solution to the ever-evolving security challenges in RWNs

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1.INTRODUCTION

With the rapid advancements in wireless communication technologies and the increasing deployment of Reconfigurable Wireless Networks (RWNs), new opportunities have arisen for designing flexible and scalable network architectures. RWNs, characterized by their reconfigurability at topological, frequency, and power levels, provide dynamic capabilities for adapting to a wide range of applications, including IoT, smart cities, and autonomous systems. However, these dynamic and heterogeneous networks also bring significant security challenges. The increased attack surface, combined with evolving cyber threats, necessitates robust intrusion detection mechanisms to ensure the integrity and reliability of RWNs. Traditional intrusion detection systems, which largely rely on rule-based methodologies, struggle to address the complexities of modern cyber-attacks, including zero-day threats, and fail to adapt to changes in attack strategies or network configurations. To address these limitations, this study proposes a deep learning-based intrusion detection system (DL-IDS) designed for both binary and multi-level attack classification. Leveraging the advanced capabilities of Convolutional Neural Networks (CNNs) and hybrid machine learning models, the system detects various cyber threats such as DoS attacks, spoofing, and unauthorized access.

Optimized for resource-constrained environments, the DL-IDS ensures real-time detection and response, making it highly effective in critical and computationally intensive scenarios. This innovative framework represents a significant step forward in safeguarding next-generation wireless communication systems, offering a dynamic and scalable solution to the ever-evolving security challenges in RWNs. This project focuses on developing a deep learning-based intrusion detection system (DL-IDS) tailored to address the security challenges of Reconfigurable Wireless Networks (RWNs).

The system leverages advanced neural network architectures, such as Convolutional Neural Networks (CNNs), to identify and classify cyber threats effectively. The DL-IDS framework is designed to perform both binary classifications, determining whether an attack is present, and multi-level classification, identifying specific types of attacks like DoS, spoofing, or unauthorized access. This dual-level detection ensures comprehensive threat analysis, significantly enhancing the security of dynamic and heterogeneous networks. The framework is further optimized for resource-constrained environments, ensuring operational efficiency even in computationally intensive scenarios.

The project encompasses the design, implementation, and evaluation of the DL-IDS framework. It begins with data preprocessing and feature extraction to prepare network traffic data for effective analysis. The deep learning models are then trained on these datasets to identify complex patterns associated with different attack types. Real-time performance and adaptability are key aspects of the implementation, ensuring the system's effectiveness in practical deployments. The evaluation phase includes rigorous

testing to assess the accuracy, speed, and scalability of the system under various network conditions. Finally, the project outlines potential applications of the DL-IDS in safeguarding next-generation wireless systems, such as IoT networks, smart cities, and 5G infrastructures, establishing its significance as a robust solution for modern cybersecurity challenges.

The dynamic and heterogeneous nature of Reconfigurable Wireless Networks (RWNs) has opened up unparalleled opportunities for flexibility and scalability in various domains, including IoT, smart cities, and autonomous systems. However, this very dynamism has significantly expanded the attack surface, exposing these networks to sophisticated cyber threats and zero-day vulnerabilities.

Traditional rule-based intrusion detection systems fail to adapt to the ever-evolving nature of cyber-attacks and the fluid configurations of RWNs, making them inadequate for ensuring network integrity and security. This growing gap between advanced cyber threats and the limitations of existing security mechanisms underscores the need for a robust, adaptable, and efficient intrusion detection system. Deep learning, with its ability to uncover complex patterns in large datasets without explicit programming, offers a transformative solution. By leveraging deep learning architectures such as Convolutional Neural Networks (CNNs), it is possible to build an intrusion detection system capable of real-time threat identification and response, even in resource-constrained environments. This motivation drives the development of a novel DL-IDS framework to address the critical security challenges posed by next-generation wireless communication systems.

The proposed deep learning-based intrusion detection system (DL-IDS) aims to address the pressing security challenges in Reconfigurable Wireless Networks (RWNs) by leveraging advanced machine learning and neural network architectures. Its scope encompasses a wide range of applications and functionalities, including the detection of cyber threats such as DoS attacks, spoofing, and unauthorized access.

By performing both binary and multi-level classifications, the DL-IDS ensures a comprehensive approach to identifying the presence and type of attacks, enhancing the overall security framework for dynamic network environments. The DL-IDS framework is designed to operate effectively in resource-constrained environments, making it suitable for integration into IoT networks, smart cities, and other critical applications where computational efficiency is paramount. Furthermore, its real-time detection and response capabilities position it as a valuable tool for safeguarding next-generation wireless communication systems, including 5G and beyond. The system's adaptability to evolving attack patterns and network configurations ensures long-term relevance and scalability, providing a robust foundation for advancing the security of dynamic and heterogeneous wireless networks.

2 System Analysis

2.1 Hard and software requirements

1. OS: Windows 10/11, Linux, or macOS.
2. Languages: Python (TensorFlow/Keras) or Java.
3. IDEs: PyCharm, VS Code, or Jupyter Notebook.
4. Libraries: TensorFlow, Scikit-learn, Pandas, NumPy
5. Processor: Multi-core Intel i5/i7 or AMD equivalent.
6. RAM: 8 GB for development, 16 GB+ for production.
7. Storage: SSD with 256 GB for dev, 1 TB+ for production.
8. GPU: NVIDIA RTX for dev, Tesla for production.

2.2 Existing system and its disadvantages

Traditional Intrusion Detection Systems (IDS) in wireless networks primarily rely on rule-based methodologies to identify security breaches. These systems are designed to detect known attack patterns by matching them against pre-defined rules or signatures. While this approach is effective in detecting previously identified threats, it lacks the ability to address the dynamic and evolving nature of modern cyber-attacks, such as zero-day vulnerabilities and sophisticated intrusion techniques.

Additionally, traditional IDS are often limited in scalability and adaptability, making them less suitable for dynamic environments like Reconfigurable Wireless Networks (RWNs). These systems typically struggle with real-time detection and response due to their dependency on static rule sets and their inability to accommodate changes in network topology, traffic patterns, or attack methods. Consequently, the effectiveness of traditional IDS is significantly compromised in modern, complex network environments.

● **Lack of Adaptability**: Traditional IDS rely on static rule-based methodologies, making them ineffective in identifying evolving and sophisticated attack patterns, such as zero-day attacks and advanced persistent threats.

● **Inability to Handle Dynamic Networks**: These systems struggle with real-time scalability and are not designed to adapt to changes in network topology, traffic patterns, or configurations, which are common in Reconfigurable Wireless Networks (RWNs).

● **Dependence on Pre-defined Rules**: The reliance on predefined rule sets limits the system's capability to detect novel or unknown intrusions that fall outside the scope of existing knowledge.

● **Limited Scalability**: Traditional IDS are often not optimized for large-scale or resource-constrained environments, which hinders their effectiveness in handling high volumes of network traffic in dynamic systems.

2.3 Proposed system and its advantages

The proposed system introduces a deep learning-based Intrusion Detection System (DL-IDS) tailored for Reconfigurable Wireless Networks (RWNs). This system leverages advanced neural network architectures, including Convolutional Neural Networks (CNNs) and machine learning models, to perform both binary and multi-level classification of network traffic. The binary classification identifies whether an intrusion has occurred, while the multi-level classification determines the specific type of attack, such as DoS attacks, spoofing, or unauthorized access.

This system is designed to adapt to dynamic network environments by learning complex patterns hidden within large datasets, making it capable of addressing sophisticated cyber threats, including zero-day attacks. The DL-IDS is optimized for real-time performance in resource-constrained environments and is robust enough to handle critical computationally intensive scenarios.

This adaptability and efficiency ensure the system's suitability for next-generation wireless communication networks, offering enhanced protection against evolving security challenges.

1. Enhanced Detection Accuracy: The use of deep learning models ensures superior accuracy in identifying both known and unknown intrusion patterns, reducing false positives and false negatives.

2. Real-time Scalability: The system is optimized to handle real-time network traffic, ensuring timely detection and response to potential threats.

3. Multi-level Classification: Unlike traditional systems, the proposed DL-IDS can classify various types of intrusions, offering more granular insights into the nature of attacks.

4. Adaptability to Dynamic Networks: The system is designed to function effectively in dynamic and heterogeneous environments, making it well-suited for RWNs.

5. Resource Efficiency: The proposed solution is optimized for resource-constrained scenarios, ensuring low computational overhead without compromising performance.

6. Robustness Against Evolving Threats: The deep learning-based approach equips the system to detect zero-day vulnerabilities and adapt to emerging cyber threats.

Feature	Traditional IDS	Proposed DL-IDS
Detection Method	Rule-based / Signature-based	Deep Learning (CNNs, LSTM, Autoencoder)
Accuracy	Limited accuracy; prone to false positives/negatives	Higher accuracy; better detection of complex patterns
Adaptability	Relies on static rules, making it ineffective against evolving threats	Dynamic learning capabilities; detects zero-day attacks
Threat Coverage	Detects only known attacks	Identifies both known and unknown intrusions
Scalability	Poor scalability; struggles in large-scale networks	Optimized for real-time, large-scale traffic
Response Time	Slower response due to static rule evaluation	Real-time response with faster mitigation actions
Multi-class Classification	Only detects generic malicious activity	Identifies specific attack types (DoS, R2L, U2R, etc.)
Efficiency in Dynamic Networks	Inefficient in Reconfigurable Wireless Networks (RWNs)	Designed for RWNs, adapting to changing topologies
Resource Efficiency	Resource-intensive; struggles with dynamic traffic	Optimized for resource-constrained environments
Robustness Against New Threats	Ineffective against zero-day attacks	Resilient to emerging cyber threats
False Positive/Negative Rate	Higher false positives and negatives	Lower false positive and negative rates
Overall Performance	Less effective in modern network environments	More robust, adaptive, and accurate

2.4 Feasibility Study

2.4.1 Functional Requirements

1. Intrusion Detection:

- Ability to detect intrusions in the network, including both known and unknown attack patterns.
- Perform binary classification to identify the presence of an attack.
- Perform multi-level classification to categorize the type of attack (e.g., DoS, spoofing, unauthorized access).

2. Data Preprocessing:

- Extract and preprocess network traffic data to prepare it for analysis.
- Handle missing or noisy data effectively to ensure accurate detection.

3. Model Training and Testing:

- Train the deep learning model using labeled datasets.
- Evaluate the model's performance through metrics such as accuracy, precision, recall, and F1-score.

2.4.2 Non-Functional Requirements

1. Performance:

- Ensure high throughput to process large volumes of network data without significant delays.
- Achieve low latency in real-time threat detection and response.

2. Scalability:

- The system should handle increasing network traffic and adapt to larger, more complex networks.

3.Reliability:

- Ensure consistent operation under varying conditions, including peak loads.
- Maintain high detection accuracy over time with minimal degradation.

4.Security:

- Protect sensitive data used in training and detection from unauthorized access or breaches.

5.Usability:

- Provide a user-friendly interface for administrators to monitor detections and configure settings.

6.Maintainability:

- Design the system to allow easy updates and modifications to accommodate new requirements or attack patterns.

2.4.3 Performance Requirements

1. High Detection Accuracy:

- The system should achieve a high accuracy rate in detecting both known and unknown intrusions.

2. Low False Positives and False Negatives:

- The system must minimize the occurrence of false alarms and undetected attacks to ensure reliability.

3. Real-time Processing:

- The system should process and classify network traffic data in real-time, with minimal delay, to ensure immediate response to threats.

4. Scalability:

- The system must handle large-scale networks and increased traffic volume without performance degradation.

5. **Latency:**

    - Ensure a detection and response latency of less than a specified threshold (e.g., milliseconds) to address threats promptly.

2.4.4 Safety Requirements

1. **Data Integrity:**

    - Ensure that the data being processed and stored remains unaltered and secure from unauthorized modifications.

2. **System Reliability:**

    - The system should function reliably under all conditions, including peak loads, ensuring uninterrupted intrusion detection.

3. **Error Handling:**

    - Implement robust error-handling mechanisms to prevent system crashes or data loss during failures.

4. **Fail-safe Mechanisms:**

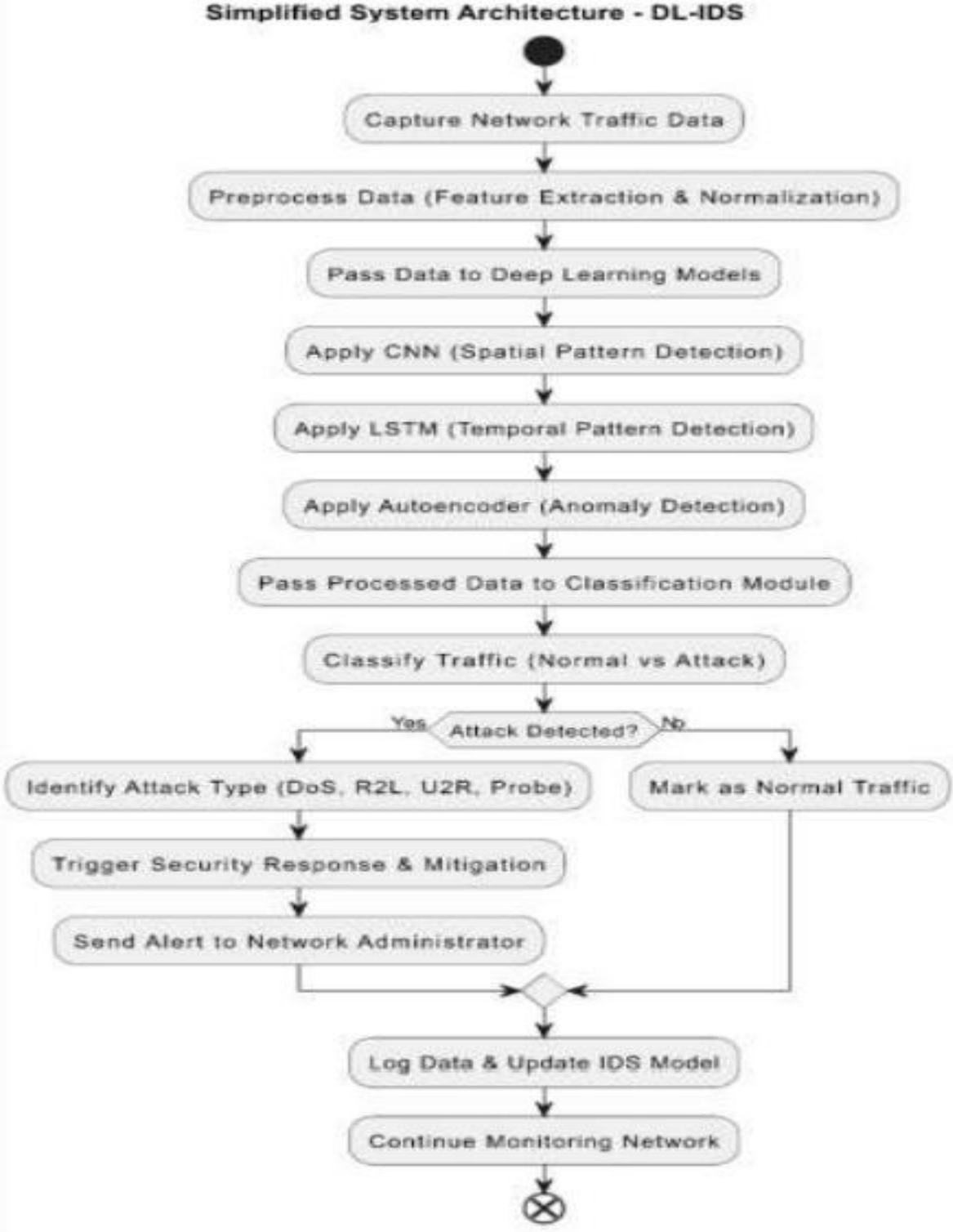
    - In the event of a system failure, implement measures to minimize the impact and ensure that critical operations can continue without interruption.

5. **Access Control:**

    - Restrict access to sensitive data and system configurations to authorized personnel only

3 System Design

3.1 Data flow diagram



3.1.1 Capture Network Traffic Data

- The system begins by collecting real-time network traffic data. This involves capturing packets, including their headers and payloads.
- Tools like Wireshark, tcpdump, or custom packet sniffers can be used for data collection.

3.1.2 Preprocess Data (Feature Extraction & Normalization)

- **Feature Extraction:**
    - Raw packet data is transformed into structured features like **protocol type, source and destination IP addresses, port numbers, packet size**, etc.
    - More complex features, such as **time-based or connection-based statistics**, may also be extracted.
- **Normalization:**
    - The features are scaled to a consistent range (e.g., 0-1) using techniques like **Min-Max Scaling** or **Z-score normalization**.
    - This ensures that all features have equal influence on the deep learning models.

3.1.3 Pass Data to Deep Learning Models

- The preprocessed data is fed into multiple **deep learning models** for detection and classification.
- The use of multiple models indicates an **ensemble or hybrid architecture**.

3.1.4. Apply CNN (Spatial Pattern Detection)

- **Convolutional Neural Networks (CNNs)** identify **spatial correlations** in the input data.
- This is useful for detecting specific patterns in the packet data, such as:
    - Signature-based attack patterns
    - Abnormal packet distributions
- CNNs extract low-level features like **protocol behavior** and **packet sequences**.

3.1.5. Apply LSTM (Temporal Pattern Detection)

- **Long Short-Term Memory (LSTM)** networks capture **temporal dependencies** in network traffic.
- This is effective for identifying **sequential attack behaviors**, such as:

- Slow or continuous probing
- Patterns in distributed denial-of-service (DDoS) attacks

- LSTM can detect **time-dependent anomalies** that are not visible in isolated packets.

3.1.6 Apply Autoencoder (Anomaly Detection)

- An **autoencoder** is used to detect **anomalies** by learning a compressed representation of normal traffic.
- During inference:
  - If the **reconstruction error** is high, the sample is flagged as **anomalous**.
  - This helps detect **zero-day attacks** or unknown threats.

3.1.7 Pass Processed Data to Classification Module

- The output from the deep learning models is passed to a **classification module**.
- The module distinguishes between:
  - **Normal traffic**
  - **Attack traffic**

3.1.8 Classify Traffic (Normal vs Attack)

- The system classifies each packet or connection as:
  - **Normal**: Legitimate network traffic.
  - **Attack**: Malicious activity.

3.1.9 Attack Detection Decision

- **If an attack is detected**:
  - Proceed to attack type identification.
- **If no attack is detected**:
  - The traffic is marked as **normal**.

3.1.10 Identify Attack Type

- If malicious traffic is detected, the system identifies the **specific type of attack**:
  - **DoS (Denial of Service)**: Flooding the network with traffic to overwhelm resources.
  - **R2L (Remote-to-Local)**: Unauthorized access from a remote system.

- **U2R (User-to-Root)**: Privilege escalation attempts.
- **Probe**: Scanning for vulnerabilities.

- Multi-class classification techniques are used here.

3.1.11 Trigger Security Response & Mitigation

- The system triggers an automated **mitigation response**, which may include:
  - Blocking malicious IPs.
  - Throttling suspicious traffic.
  - Applying access control rules.

3.1.12 Send Alert to Network Administrator

- The system sends an **alert notification** to the **network administrator**.
- The alert includes details such as:
  - Attack type.
  - Source and destination IP addresses.
  - Timestamp.
  - Mitigation action taken.

3.1.13 Log Data & Update IDS Model

- The system logs the traffic data and detection results.
- This data is used to **retrain the model** periodically, improving its accuracy and adaptability to evolving threats.

3.1.14 Continue Monitoring Network

- The system continues monitoring in a **continuous feedback loop**, ensuring ongoing protection.

3.2 Use case diagrams

Actors:

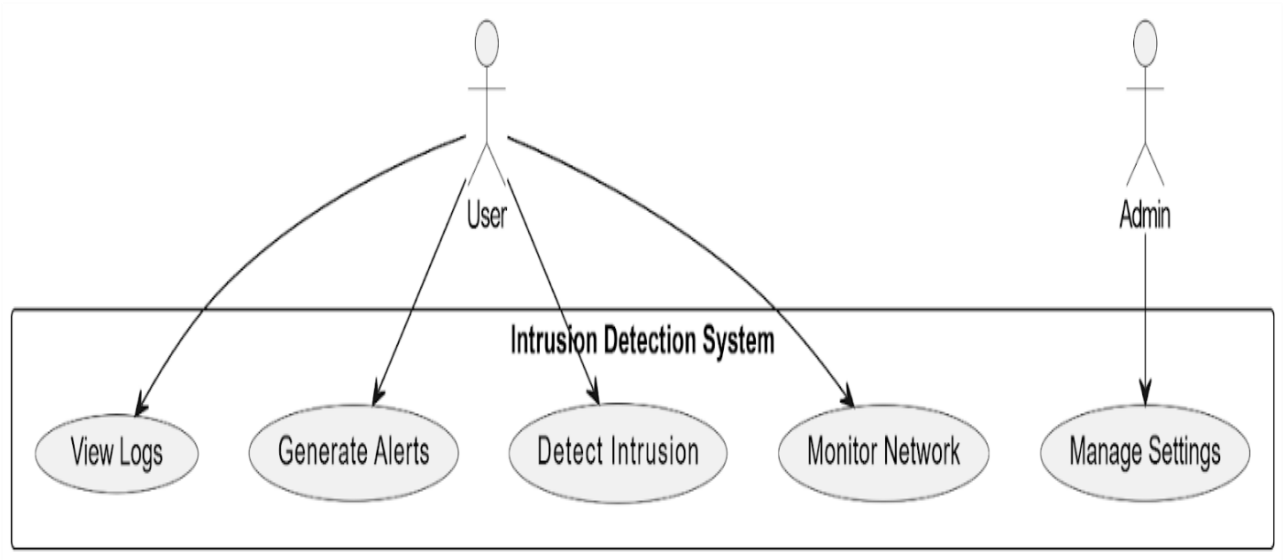
1.User

- Represents the regular user of the IDS, likely a security analyst or network operator.
- Can perform the following actions:

- View Logs: Access historical network activity logs for inspection.
- Generate Alerts: Create notifications based on detected threats.
- Detect Intrusion: Utilize the IDS to identify potential threats in the network.
- Monitor Network: View real-time network traffic and analyze ongoing activities.

2.Admin

- Represents the system administrator with elevated privileges.
- Can perform:
  - Manage Settings: Configure and customize the IDS settings, such as security rules, thresholds, or alert preferences**.**



3.3 Class diagram

1. IntrusionDetectionSystem

- The core class responsible for managing the IDS.
- **Methods:**
  - +detectIntrusion() → Detects potential threats in the network traffic.
  - +analyzeTraffic() → Analyzes incoming and outgoing network data.

- **Relationships:**
  - Uses the DeepLearningModel, FeatureExtractor, and AlertManager classes.

2. DeepLearningModel

- Handles the machine learning operations for intrusion detection.
- **Methods:**
  - +trainModel() → Trains the ML model using labeled datasets (e.g., KDD Cup 1999).
  - +predictIntrusion() → Predicts whether the traffic is normal or malicious.
- **Relationships:**
  - Uses the Dataset class for training and prediction operations.

3. FeatureExtractor

- Responsible for preprocessing and extracting features from the raw network data.
- **Methods:**
  - +extractFeatures() → Extracts relevant features for ML classification (e.g., protocol type, service, duration).
  - +normalizeData() → Normalizes the data to standardize it for consistent model performance.

4. AlertManager

- Manages alerts and logs incidents in the system.
- **Methods:**
  - +sendAlert() → Sends notifications in case of suspicious activity**.**
  - +logIncident() → Records incidents for future reference and auditing.
- **Relationships:**
  - Notifies the User class.

5. Dataset

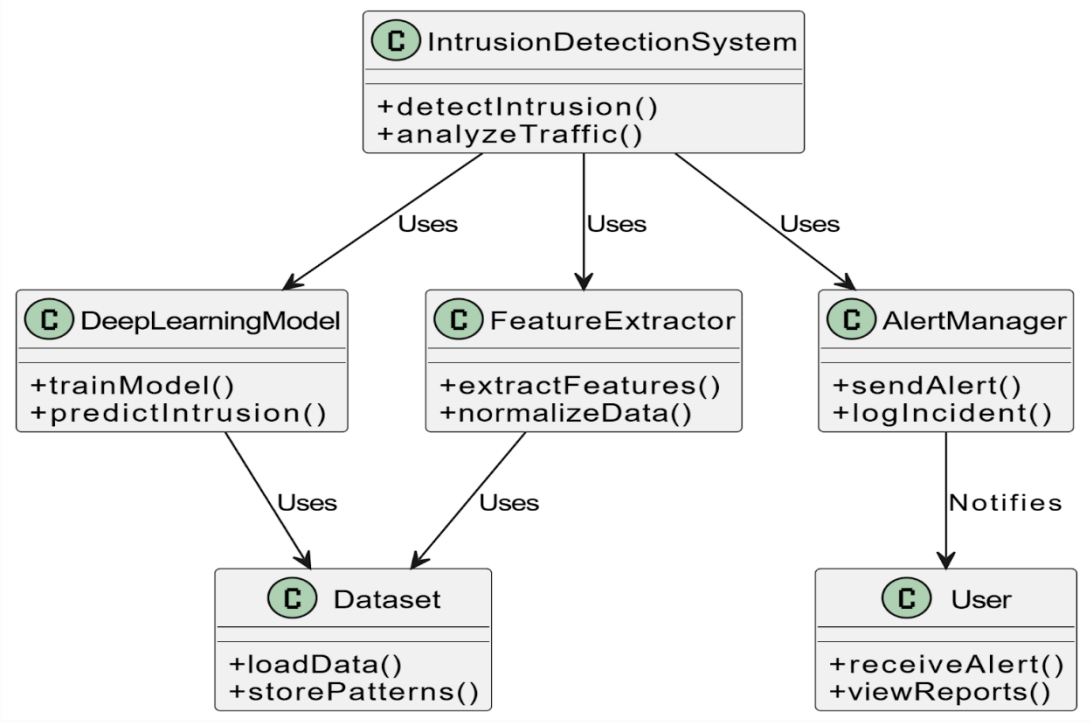
- Represents the data handling operations.
- **Methods:**
  - +loadData() → Loads the training and testing data from the KDD Cup 1999 dataset.
  - +storePatterns() → Stores identified patterns for future use or reference.

- **Relationships:**
  - Used by the DeepLearningModel during training and prediction phases.

6. User

- Represents the system's user who receives alerts and views reports.
- **Methods:**
  - +receiveAlert() → Receives alerts about suspicious activities.
  - +viewReports() → Views reports of past incidents and network activity.
- **Relationships:**
  - Notified by the AlertManager.

The DL-IDS class diagram presents a robust, modular, and scalable architecture that efficiently integrates deep learning models with real-time intrusion detection capabilities. The system is designed to handle large-scale, dynamic network environments, making it highly effective in combating modern cyber threats, including zero-day attacks and sophisticated intrusion techniques.



3.4 Activity Diagram

1. Capture Network Traffic

- The system monitors and captures network packets from incoming and outgoing network traffic.
- Tools like Wireshark, Tcpdump, or network taps can be used for this purpose.

2. Preprocess Data

- The captured raw data is preprocessed to remove noise and extract relevant information.
- This may involve packet filtering, removing redundant information, and normalizing data for analysis.

3. Extract Features

- Important features such as packet size, protocol type, connection duration, and payload characteristics are extracted.
- Feature selection techniques (e.g., PCA, LDA) can be applied to reduce dimensionality.

4. Classify using Deep Learning Model

- The processed features are fed into a deep learning model (e.g., CNN, LSTM, Transformer-based models) trained to detect anomalies or malicious activity.
- The model classifies network activity as either normal or suspicious/malicious.

5. Threat Detection Decision

- A decision point determines whether a threat is detected:
    - Yes (Threat Detected): The system generates an alert.
    - No (No Threat Detected): The system continues monitoring network traffic.

6. Generate Alert (If a Threat is Detected)

- If malicious activity is identified, an alert is generated for security teams or automated response systems.

7. Log Incident

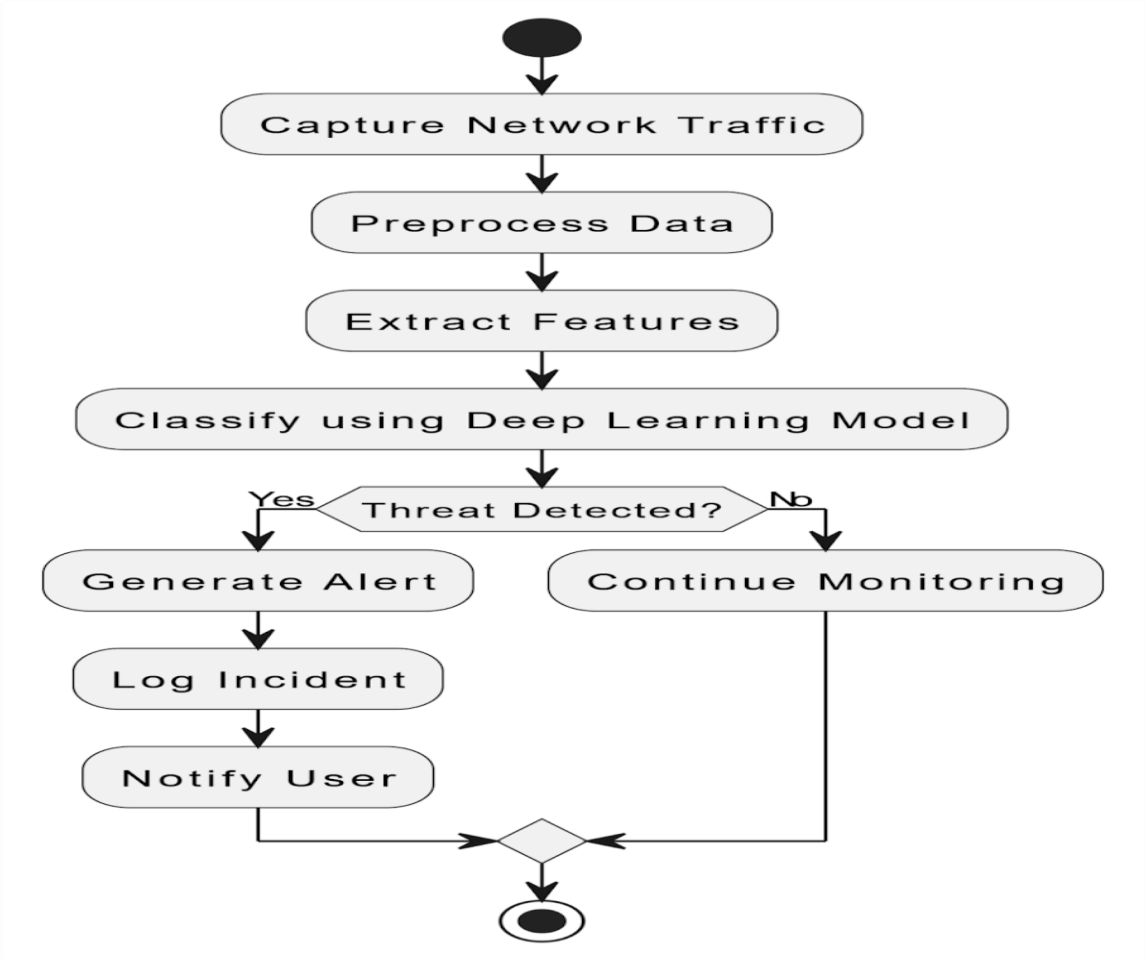
- The detected threat is logged in a database for further investigation.
- Logs may include timestamps, attack type, source and destination IPs, and affected services.

8. Notify User

- The security team or administrator is notified through email, SMS, or system logs.
- Automated countermeasures (such as blocking the IP) may also be triggered.

9. Continue Monitoring

- If no threat is detected, the system resumes monitoring network traffic for further anomalies.
- The process repeats in a continuous loop to ensure real-time security monitoring.



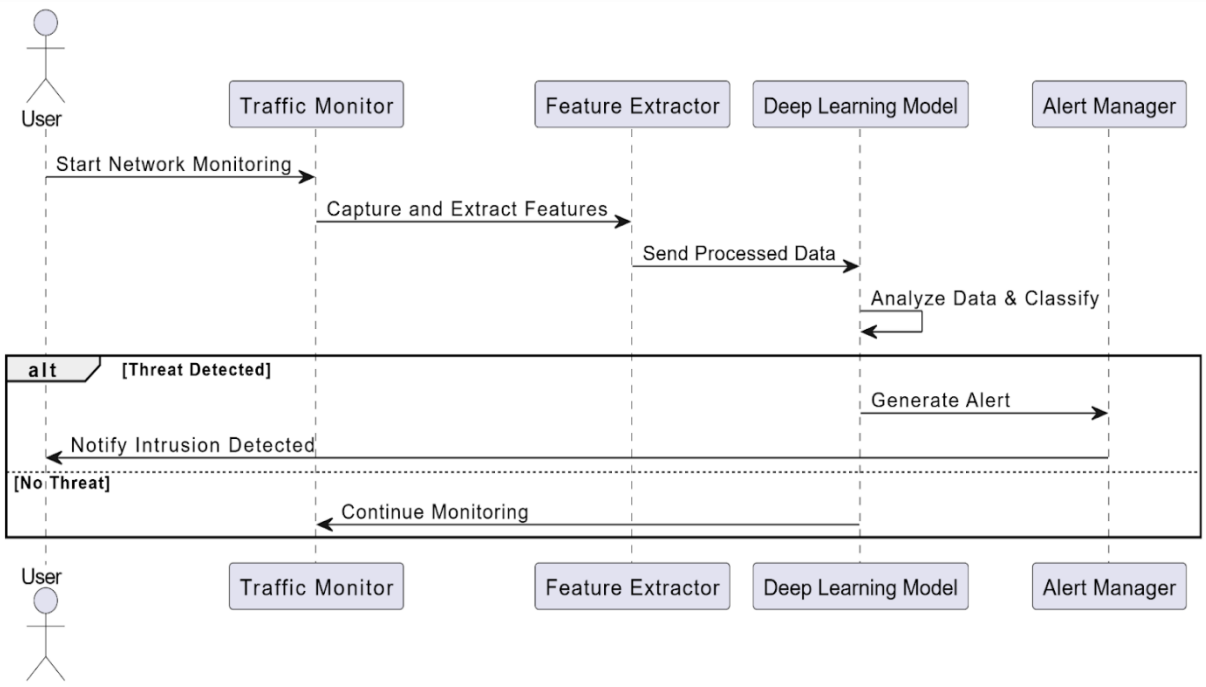
3.5 Sequence Diagram

Actors & Components Involved

1. **User:** The entity that starts and monitors network activity.
2. **Traffic Monitor:** Captures network traffic and forwards it for feature extraction**.**
3. **Feature Extractor:** Processes raw network data to extract relevant features for classification.
4. **Deep Learning Model:** Analyzes extracted features and classifies the traffic as normal or suspicious.
5. **Alert Manager:** Handles alert generation if a threat is detected.

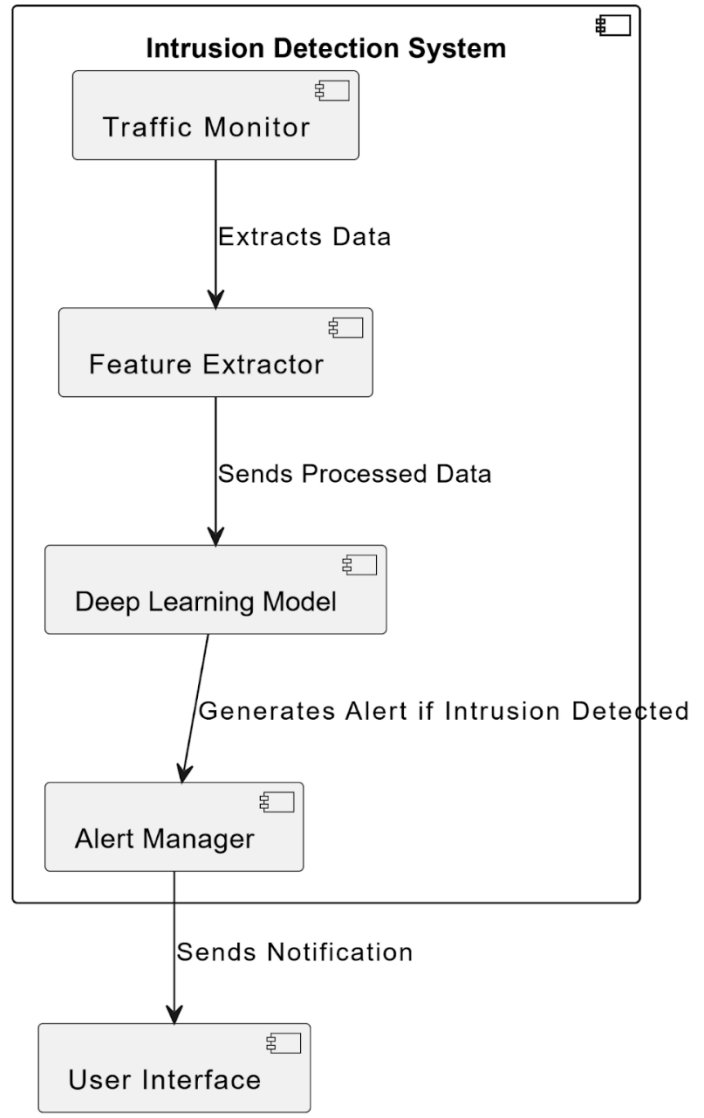
Key Features of the Diagram

- **Conditional Alternative (alt block)**: It ensures that different actions are taken based on whether a threat is detected.
- **Sequence Flow**: The diagram clearly shows how data moves between components.
- **User Interaction**: The user is notified only when a threat is detected.



3.6 Component Diagram

- Traffic Monitor extracts raw network data.
- Feature Extractor processes and converts data into structured features.
- Deep Learning Model analyzes the features and detects intrusions.
- Alert Manager processes alerts and determines their severity.
- User Interface displays the alerts for security monitoring.



3.7 ER Model Overview

The ER model represents the data structure and relationships within the DL-IDS. It defines the entities, attributes, and relationships between different components, focusing on how data flows through the system.

3.6.1. Entities and Attributes

1. IntrusionDetectionSystem

- Primary Entity representing the IDS core functionality.
- **Attributes:**
  - system_id (PK) → Unique ID for each IDS instance.
  - status → Current state (running, idle, completed).
  - detection_time → Timestamp of intrusion detection.
  - traffic_analyzed → Amount of network traffic analyzed.
- **Relationships:**
  - Uses → DeepLearningModel
  - Extracts features → FeatureExtractor
  - Generates alerts → Alert

2. DeepLearningModel

- Entity handling ML operations for intrusion detection.
- **Attributes:**
  - model_id (PK) → Unique ID for each ML model instance.
  - algorithm → Type of model (CNN, LSTM, etc.).
  - accuracy → Model accuracy score.
  - training_status → Indicates whether the model is trained or in progress.
- **Relationships:**
  - Trains on → Dataset
  - Used by → IntrusionDetectionSystem

3. FeatureExtractor

- Entity responsible for feature extraction and preprocessing.
- **Attributes:**
  - extractor_id (PK) → Unique ID for each feature extractor.
  - feature_type → Type of feature extracted (e.g., protocol type, service, duration).
  - normalization_status → Indicates if the data is normalized.
- **Relationships:**
  - Extracts features from → RawTraffic
  - Used by → IntrusionDetectionSystem

4. Alert

- Entity handling alerts and notifications.
- **Attributes:**
  - alert_id (PK) → Unique ID for each alert.
  - alert_type → Type of alert (e.g., DoS, probing).
  - severity_level → Severity rating (low, medium, high).
  - timestamp → Time when the alert was generated.
- **Relationships:**
  - Sent by → AlertManager
  - Received by → User

5. AlertManager

- Entity managing incident reporting and alert logging.
- **Attributes:**
  - manager_id (PK) → Unique ID for each manager.
  - log_file → Path to the incident log.
  - active_alerts → Number of active alerts.

- **Relationships:**
    - Generates → Alert
    - Sends notification to → User

6. Dataset

- Entity representing the training and testing data.
- **Attributes:**
    - dataset_id (PK) → Unique ID for the dataset.
    - data_source → Source of the dataset (e.g., KDD Cup 1999).
    - record_count → Number of records in the dataset.
    - label → Labels indicating normal or malicious traffic.
- **Relationships:**
    - Used by → DeepLearningModel

7. User

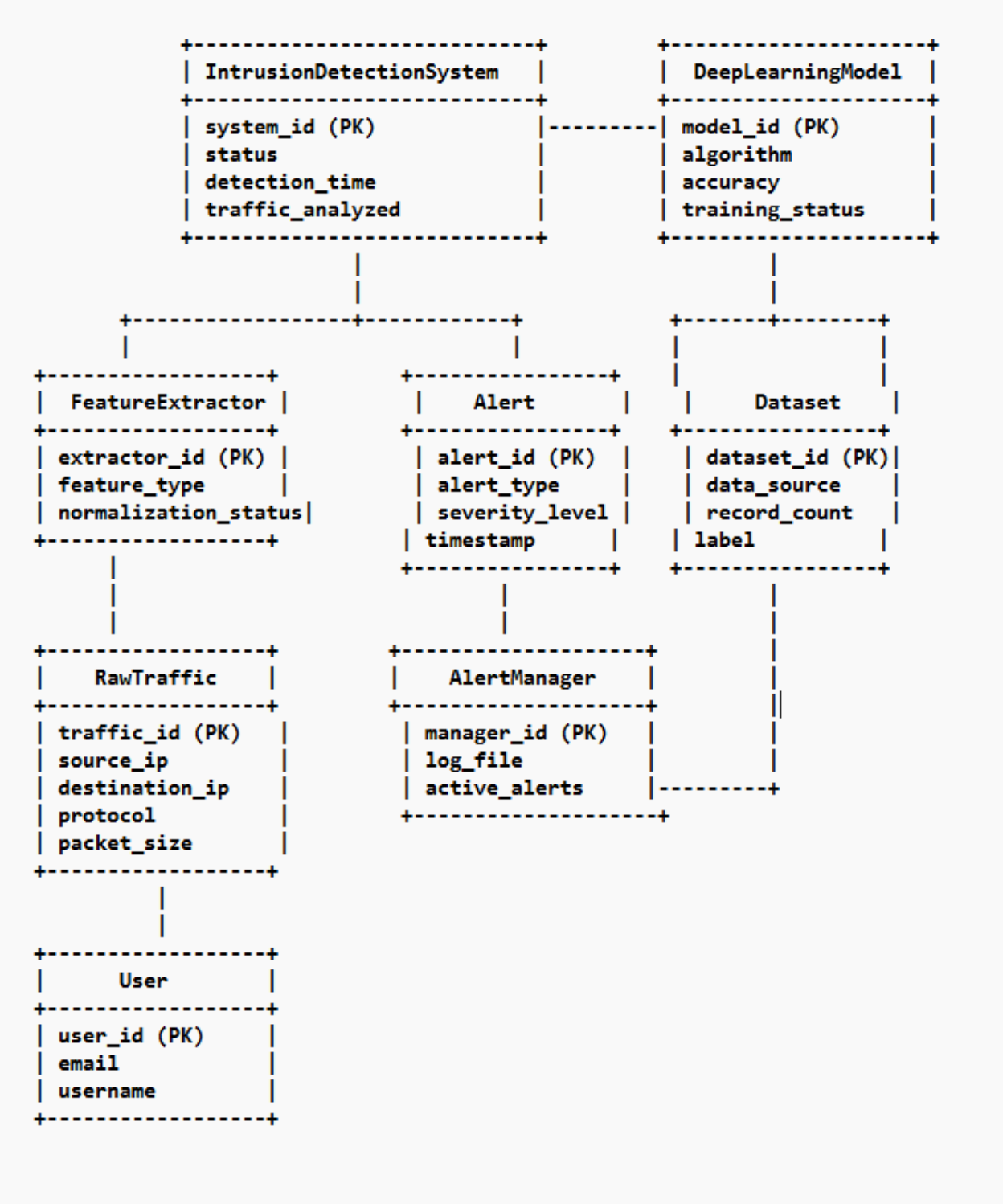
- Entity representing the system user.
- **Attributes:**
    - user_id (PK) → Unique ID for the user.
    - email → Email address for receiving alerts.
    - username → User's name or alias.
- **Relationships:**
    - Receives alerts from → Alert
    - Views reports from → IntrusionDetectionSystem

8. RawTraffic

- Entity representing the raw network traffic being analyzed.
- **Attributes:**
    - traffic_id (PK) → Unique ID for each traffic log.
    - source_ip → IP address of the sender.

- destination_ip → IP address of the receiver.
- protocol → Network protocol (TCP, UDP, etc.).
- packet_size → Size of the network packet.

- **Relationships:**
  - Analyzed by → IntrusionDetectionSystem
  - Processed by → FeatureExtractor


IntrusionDetectionSystem
system_id (PK)
status
detection_time
traffic_analyzed
DeepLearningModel
model_id (PK)
algorithm
accuracy
training_status
FeatureExtractor
extractor_id (PK)
feature_type
normalization_status
Alert
alert_id (PK)
alert_type
severity_level
timestamp
Dataset
dataset_id (PK)
data_source
record_count
label
RawTraffic
traffic_id (PK)
source_ip
destination_ip
protocol
packet_size
AlertManager
manager_id (PK)
log_file
active_alerts
User
user_id (PK)
email
username

3.6.2 Key Relationships and Cardinality

- IntrusionDetectionSystem → DeepLearningModel:
  - One-to-One: Each IDS uses a single deep learning model for intrusion detection.
- IntrusionDetectionSystem → FeatureExtractor:
  - One-to-One: Each IDS uses a single feature extractor to preprocess network traffic.
- DeepLearningModel → Dataset:
  - Many-to-One: Multiple models can train on the same dataset.
- AlertManager → Alert:
  - One-to-Many: Each alert manager generates multiple alerts.
- Alert → User:
  - Many-to-One: Multiple alerts can be sent to the same user.
- RawTraffic → IntrusionDetectionSystem:
  - Many-to-One: Multiple network traffic logs are analyzed by a single IDS instance

3.6.3. Conclusion

The ER model effectively captures the data structure and relationships of the DL-IDS, including the handling of network traffic, feature extraction, machine learning predictions, and alert management.

1. Modularity:

- The model promotes modularity and flexibility, making it easy to scale and expand by adding new features or components.

2.Efficient Data Flow:

- The relationships between entities facilitate smooth data flow across different IDS components, enabling efficient intrusion detection.

3.Scalability and Extensibility:

- The model is designed to support large-scale network traffic, making it adaptable to dynamic environments like Reconfigurable Wireless Networks (RWNs).

Real-time Alerting:

- The AlertManager and User relationships ensure immediate notifications of suspicious activities, enhancing the system's responsiveness to cyber threats.

4. Module descriptions

4.1 User Interface(UI) Module

● **Purpose:** Provides a user-friendly interface for interacting with the system. This is where users can input data, view results, and access reports.

● **Key Features:**

○ Manual data entry: Allows users to input specific network traffic details.

○ File upload: Supports bulk data submission for batch processing.

○ Real-time visualization: Displays graphs, confidence scores, and severity ratings for detected threats**.**

● **Technologies Used:** React.js for creating an interactive and responsive front-end.

4.2 Data Input and Preprocessing Module

● **Purpose:** Prepares the input data for the machine learning models. Ensures that data is clean, normalized, and formatted correctly for analysis.

● **Key Features:**

○ Data validation: Checks for missing or incorrect values in the input.

○ Normalization: Scales data to bring all features to a common range.

○ Feature extraction: Extracts meaningful features from raw network traffic logs.

● **Technologies Used**: Python libraries like pandas and numpy for data preprocessing.

4.3 Machine Learning Module

● **Purpose**: Handles the core threat detection and classification tasks. This module contains pre-trained machine learning models for identifying normal and malicious network traffic.

● **Key Features:**

○ Binary Classification: Determines whether traffic is normal or suspicious.

○ Multi-Class Classification: Categorizes suspicious traffic into specific attack types (e.g., DoS,probe,R2L, U2R).

○ Confidence scoring: Outputs the likelihood of each prediction for transparency.

● **Technologies Used**: Scikit-learn for model development and prediction.

4.4 Threat Analysis Module

● **Purpose**: Analyzes detected threats and provides actionable insights, including the type and severity of the threat.

● **Key Features:**

○ Severity rating: Assigns a severity level (e.g., low, medium, high) to detected threats.

○ Threat patterns: Identifies patterns in network traffic that may indicate ongoing attacks.

● **Technologies Used:** Python-based algorithms for severity analysis and scoring.

4.5 Dataset Module

● **Purpose:** Manages the datasets used for training and testing the machine learning models. This module is essential for ensuring accurate and reliable model predictions.

● **Key Features:**

○ Training data: Uses the KDD Cup 1999 dataset to train the classification models.

○ Testing and evaluation: Tests the models on labeled data to measure performance metrics l like accuracy and F1 score.

● **Technologies Used**: Python for dataset management and Scikit-learn for splitting and evaluation.

4.6 Backend Communication Module

● **Purpose:** Acts as the communication bridge between the UI and the ML models. Ensures smooth data transfer and model integration.

● **Key Features:**

○ API endpoints: Handles requests from the frontend and passes data to the ML module.

○ Real-time processing: Supports real-time detection for immediate feedback.

● **Technologies Used:** Python with Flask frameworks.

4.7 Security Module

● **Purpose**: Ensures that the system is secure from external threats and misuse. Protects sensitive data such as network traffic logs.

● **Key Features:**

○ Authentication: Verifies user credentials to prevent unauthorized access.

- Data encryption: Secures sensitive data during transmission.
- Threat logging: Keeps a log of detected threats for future reference and analysis.

- **Technologies Used**: SSL/TLS protocols for encryption and Python for logging.

4.8 Real-Time Monitoring Module

- **Purpose**: Monitors live network traffic to detect and classify threats as they occur.
- **Key Features:**
  - Continuous data flow: Processes incoming network data without delays.
  - Immediate alerts: Notifies users of detected threats in real time.
- **Technologies Used:** Python for data streaming and backend processing.

5.IMPLEMENTATION

5.1 Technologies used

1. Frontend Development

● **React.js:** A powerful JavaScript library used to build the interactive and responsive user interface of NetSentinel. It helps in creating dynamic components like dashboards, input forms, and visualizations for real-time threat detection and classification. React ensures a seamless user experience with fast updates and rendering.

2. Backend Development

● **Python:** The backbone of the system, Python is used for server-side logic, handling data input/output, and integrating machine learning models. Its simplicity and extensive library support make it ideal for developing the backend of NetSentinel**.**

● **Flask/Django (Optional):** These Python frameworks might be used to manage API requests and data flow between the frontend and backend.

3. Machine Learning Framework

● **Scikit-learn:** This library powers the machine learning models in NetSentinel, enabling tasks such as:

○ **Binary Classification:** Differentiating between normal traffic and potential attacks.

○ **Multi-Class Classification:** Classifying specific attack types like DoS (Denial of Service), Probe, R2L (Remote to Local), and U2R (User to Root). Scikit-learn provides pre-built algorithms, tools for evaluation, and support for model optimization.

4. Dataset

● **KDD Cup 1999 Dataset**: A benchmark dataset designed for intrusion detection. It contains labeled network traffic data, including normal behavior and various attack types. The dataset is widely recognized for testing intrusion detection systems and helps in training accurate and reliable models.

5. Data Input Mechanisms

● **Manual Input**: Users can enter network traffic parameters directly into the system for immediate analysis. This feature supports scenarios where quick testing with specific data points is needed.

6. Real-time Data Processing

● **Real-time Detection**: The system is designed to analyze network traffic patterns in real time, identifying threats as they occur. This ensures administrators can respond to potential security breaches immediately.

7. Security Mechanisms

● **Binary Classification**: A first-layer security mechanism that identifies whether the network traffic is normal or suspicious.

● **Multi-Class Classification:** A second-layer analysis that categorizes detected threats into specific types, enabling targeted responses.

● **Advanced Analytics:** Provides confidence scores and severity ratings for threats, helping administrators prioritize and respond effectively.

Component	Technology/Description
Frontend Development	**React.js:** A JavaScript library used to build interactive and responsive user interfaces. It creates dynamic components such as dashboards, input forms, and visualizations for real-time threat detection and classification.
Backend Development	**Python:** Handles server-side logic, data processing, and integrates machine learning models. **Flask/Django (Optional):** Python frameworks used for managing API requests and data flow between the frontend and backend.
Machine Learning	**Scikit-learn:** Powers the ML models with the following tasks: - **Binary Classification:** Differentiates between normal and suspicious network traffic. - **Multi-Class Classification:** Categorizes attack types (DoS, Probe, R2L, U2R).
Dataset	**KDD Cup 1999:** A benchmark dataset for intrusion detection. Contains labeled network traffic data with both normal and attack behaviors.
Data Input Mechanisms	**Manual Input:** Allows users to enter network traffic parameters directly into the system for immediate analysis. This is useful for quick testing with specific data points.
Real-time Data Processing	**Real-time Detection:** Analyzes network traffic patterns in real time, identifying threats as they occur. This ensures instant threat detection and response.
Security Mechanisms	- **Binary Classification:** First-layer security that detects whether network traffic is normal or suspicious. - **Multi-Class Classification:** Second-layer analysis that categorizes detected threats into specific attack types. - **Advanced Analytics:** Provides confidence scores and severity ratings to help prioritize and respond to threats effectively.

5.2 Sample Code

```
import numpy as np
import pandas as pd
import pickle
import os
from pathlib import Path
class BinaryClassificationModel:
  def __init__(self):
    models_dir = Path(__file__).parent.parent / 'ml_models'
    # Load saved Random Forest model
    with open(models_dir / 'random_forest_binary.pkl', 'rb') as model_file:
      self.rf_binary = pickle.load(model_file)
    # Load saved StandardScaler
    with open(models_dir / 'scaler_binary.pkl', 'rb') as scaler_file:
      self.scaler = pickle.load(scaler_file)
    # Load saved train columns for feature alignment
    with open(models_dir / 'train_columns_binary.pkl', 'rb') as columns_file:
      self.train_columns = pickle.load(columns_file)
    # Define feature names (excluding label and difficulty columns)
    self.features = [
      "duration", "protocol_type", "service", "flag", "src_bytes",
      "dst_bytes", "land", "wrong_fragment", "urgent", "hot",
      "num_failed_logins", "logged_in", "num_compromised", "root_shell",
"su_attempted",
      "num_root", "num_file_creations", "num_shells", "num_access_files",
    "num_outbound_cmds",
      "is_host_login", "is_guest_login", "count", "srv_count", "serror_rate",
```

```
            "srv_serror_rate", "rerror_rate", "srv_rerror_rate", "same_srv_rate",
"diff_srv_rate",
            "srv_diff_host_rate", "dst_host_count", "dst_host_srv_count",
        "dst_host_same_srv_rate", "dst_host_diff_srv_rate",
           "dst_host_same_src_port_rate", "dst_host_srv_diff_host_rate",
        "dst_host_serror_rate", "dst_host_srv_serror_rate", "dst_host_rerror_rate",
            "dst_host_srv_rerror_rate"
        ]
    def preprocess(self, input_data):
        """Preprocess input data for binary classification"""
        # Convert inputs into a DataFrame
        test_df = pd.DataFrame([input_data], columns=self.features
        # Convert data types to match training data
        for col in test_df.columns:
            try:
                test_df[col] = test_df[col].astype(float)
            except:
                pass  # Ignore categorical column
        # One-hot encode categorical features
        test_df = pd.get_dummies(test_df, columns=['protocol_type', 'service', 'flag'],
        prefix="", prefix_sep=""
        # Ensure the test data has the same features as training data
        for col in self.train_columns:
            if col not in test_df:
                test_df[col] = 0  # Add missing features with default value
        # Ensure correct column order
        test_df = test_df[self.train_columns]
```

```
        # Normalize numerical features using the saved StandardScaler
        test_df.iloc[:, :41] = self.scaler.transform(test_df.iloc[:, :41])
        return test_df
    def predict(self, input_data):
        """Make a binary classification prediction"""
        try:
            # Preprocess the data
            test_df = self.preprocess(input_data)
            # Predict attack type using the trained model
            y_pred = self.rf_binary.predict(test_df)
            # Get prediction probabilities
            y_prob = self.rf_binary.predict_proba(test_df)
            confidence = round(max(y_prob[0]) * 100)
            # Format result
            if y_pred[0] == 1:
                result = {
                    "prediction": "attack",
                    "confidence": confidence
                }
            else:
                result = {
                    "prediction": "normal",
                    "confidence": confidence
                }
            return result, None
        except Exception as e:
            return None, str(e)
```

```
import numpy as np
import pandas as pd
import pickle
import os
from pathlib import Path
class MultiClassificationModel:
  def __init__(self):
    models_dir = Path(__file__).parent.parent / 'ml_models'
    # Load trained Random Forest model
    with open(models_dir / 'random_forest_model_multi.pkl', 'rb') as model_file:
      self.rf = pickle.load(model_file)
    # Load saved train columns for feature alignment
    with open(models_dir / 'train_columns_multi.pkl', 'rb') as file:
      self.train_columns = pickle.load(file)
    # Load saved Label Encoder
    with open(models_dir / 'label_encoder_multi.pkl', 'rb') as le_file:
      self.le = pickle.load(le_file)
    # Load saved StandardScaler
    with open(models_dir / 'scaler_multi.pkl', 'rb') as scaler_file:
      self.scaler = pickle.load(scaler_file)
    # Define feature names (excluding label column)
    self.features = [
      "duration", "protocol_type", "service", "flag", "src_bytes", "dst_bytes", "land",
"wrong_fragment", "urgent", "hot",
      "num_failed_logins", "logged_in", "num_compromised", "root_shell",
"su_attempted", "num_root", "num_file_creations", "num_shells",
      "num_access_files", "num_outbound_cmds", "is_host_login", "is_guest_login",
"count", "srv_count", "serror_rate", "srv_serror_rate",
```

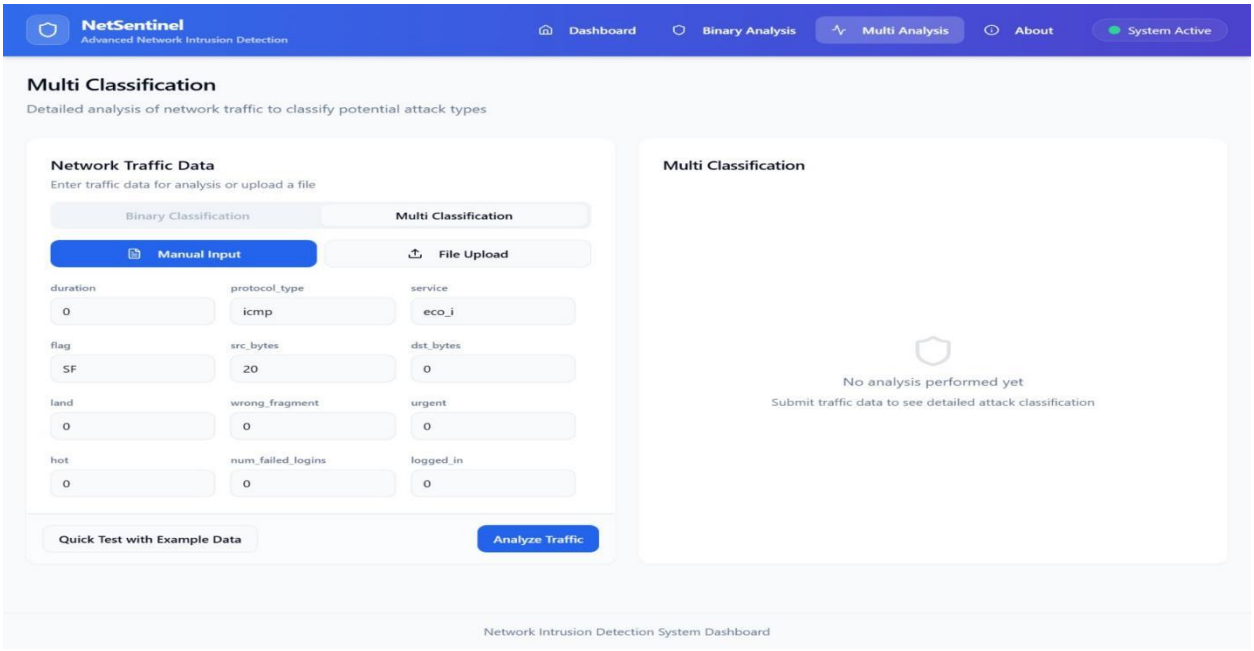
```
            "rerror_rate", "srv_rerror_rate", "same_srv_rate", "diff_srv_rate",
        "srv_diff_host_rate", "dst_host_count", "dst_host_srv_count",

            "dst_host_same_srv_rate", "dst_host_diff_srv_rate",
        "dst_host_same_src_port_rate", "dst_host_srv_diff_host_rate", "dst_host_serror_rate",

            "dst_host_srv_serror_rate", "dst_host_rerror_rate", "dst_host_srv_rerror_rate",
        "label", "difficulty"

        ]

        # Define attack categories mapping

        self.attack_categories = {

            'normal': {'category': 'Normal', 'severity': 'low', 'impact': 'No impact - normal
traffic'},

            'neptune': {'category': 'DoS', 'severity': 'high', 'impact': 'Denial of Service through
        SYN flooding'},

            'ipsweep': {'category': 'Probe', 'severity': 'medium', 'impact': 'Information
        gathering and vulnerability scanning'},

            'portsweep': {'category': 'Probe', 'severity': 'medium', 'impact': 'Port scanning to
        find active services'},

            'satan': {'category': 'Probe', 'severity': 'medium', 'impact': 'Looks for known
        vulnerabilities'},

            'smurf': {'category': 'DoS', 'severity': 'high', 'impact': 'ICMP flooding attack'},

            'teardrop': {'category': 'DoS', 'severity': 'high', 'impact': 'Sending fragmented
        packets that cannot be reassembled'},

            'buffer_overflow': {'category': 'U2R', 'severity': 'high', 'impact': 'Privilege escalation
        by overflowing memory buffers'},

            'loadmodule': {'category': 'U2R', 'severity': 'high', 'impact': 'Gaining root
privileges'},

            'perl': {'category': 'U2R', 'severity': 'high', 'impact': 'Executing commands through
Perl'},

            'rootkit': {'category': 'U2R', 'severity': 'high', 'impact': 'Installing a rootkit to gain
        continued access'},

            'ftp_write': {'category': 'R2L', 'severity': 'high', 'impact': 'Unauthorized write access
        to FTP server'},
```

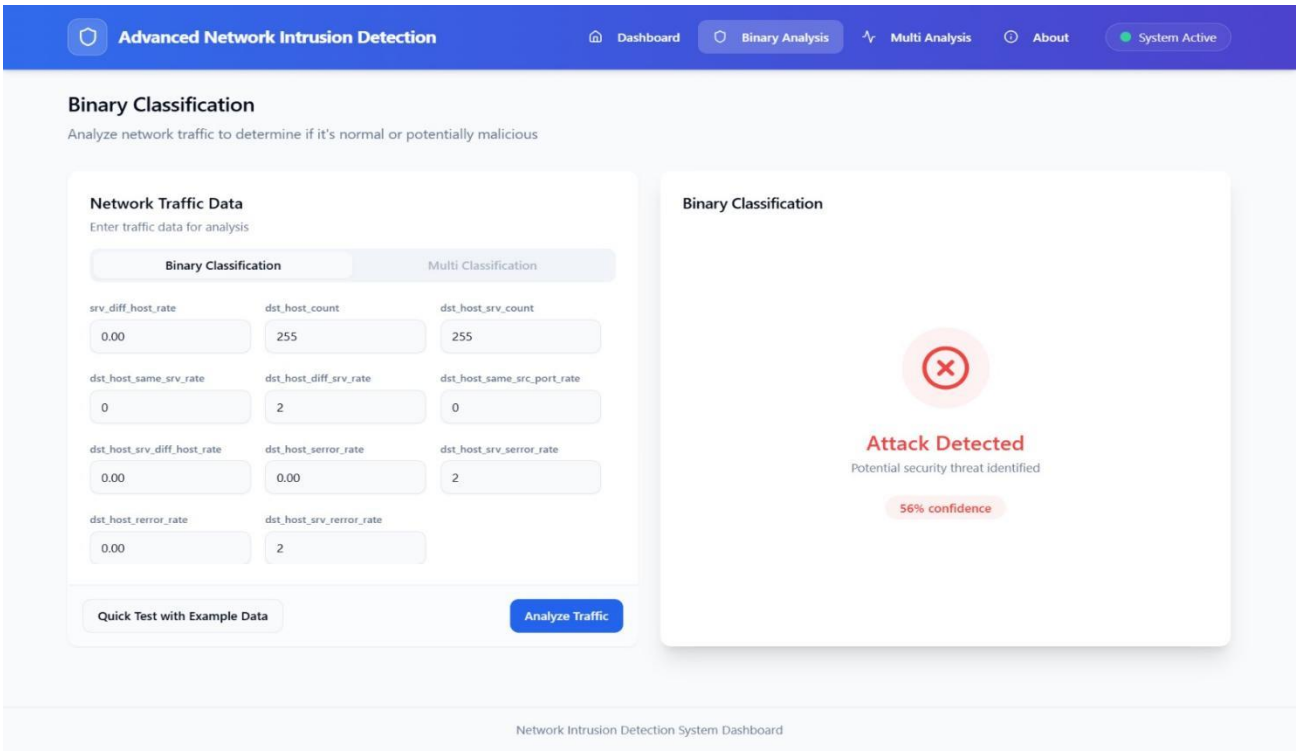
```
            'guess_passwd': {'category': 'R2L', 'severity': 'high', 'impact': 'Password guessing attack'},
            'imap': {'category': 'R2L', 'severity': 'high', 'impact': 'Exploiting vulnerabilities in IMAP protocol'},
            'multihop': {'category': 'R2L', 'severity': 'high', 'impact': 'Relay attack through multiple systems'},
            'phf': {'category': 'R2L', 'severity': 'high', 'impact': 'Executing commands through web server CGI'},
            'spy': {'category': 'R2L', 'severity': 'high', 'impact': 'Information theft through network sniffing'},
            'warezclient': {'category': 'R2L', 'severity': 'medium', 'impact': 'Using illegal software'},
            'warezmaster': {'category': 'R2L', 'severity': 'high', 'impact': 'Setting up illegal software repository'},
            'pod': {'category': 'DoS', 'severity': 'high', 'impact': 'Ping of death attack'},
            'back': {'category': 'DoS', 'severity': 'high', 'impact': 'Web server attack causing service disruption'},
            'land': {'category': 'DoS', 'severity': 'high', 'impact': 'Same source and destination address/port attack'},
            'nmap': {'category': 'Probe', 'severity': 'medium', 'impact': 'Network scanning using Nmap tool'}
        }

    def preprocess(self, input_data):
        """Preprocess input data for multi-class classification"""
        # Convert input into a DataFrame
        single_df = pd.DataFrame([input_data], columns=self.features[:-2])  # Exclude 'label' and 'difficulty'
        # One-hot encode categorical features
        single_df = pd.get_dummies(single_df, columns=['protocol_type', 'service', 'flag'], prefix="", prefix_sep="")
        # Ensure the single input has the same features as training data
```

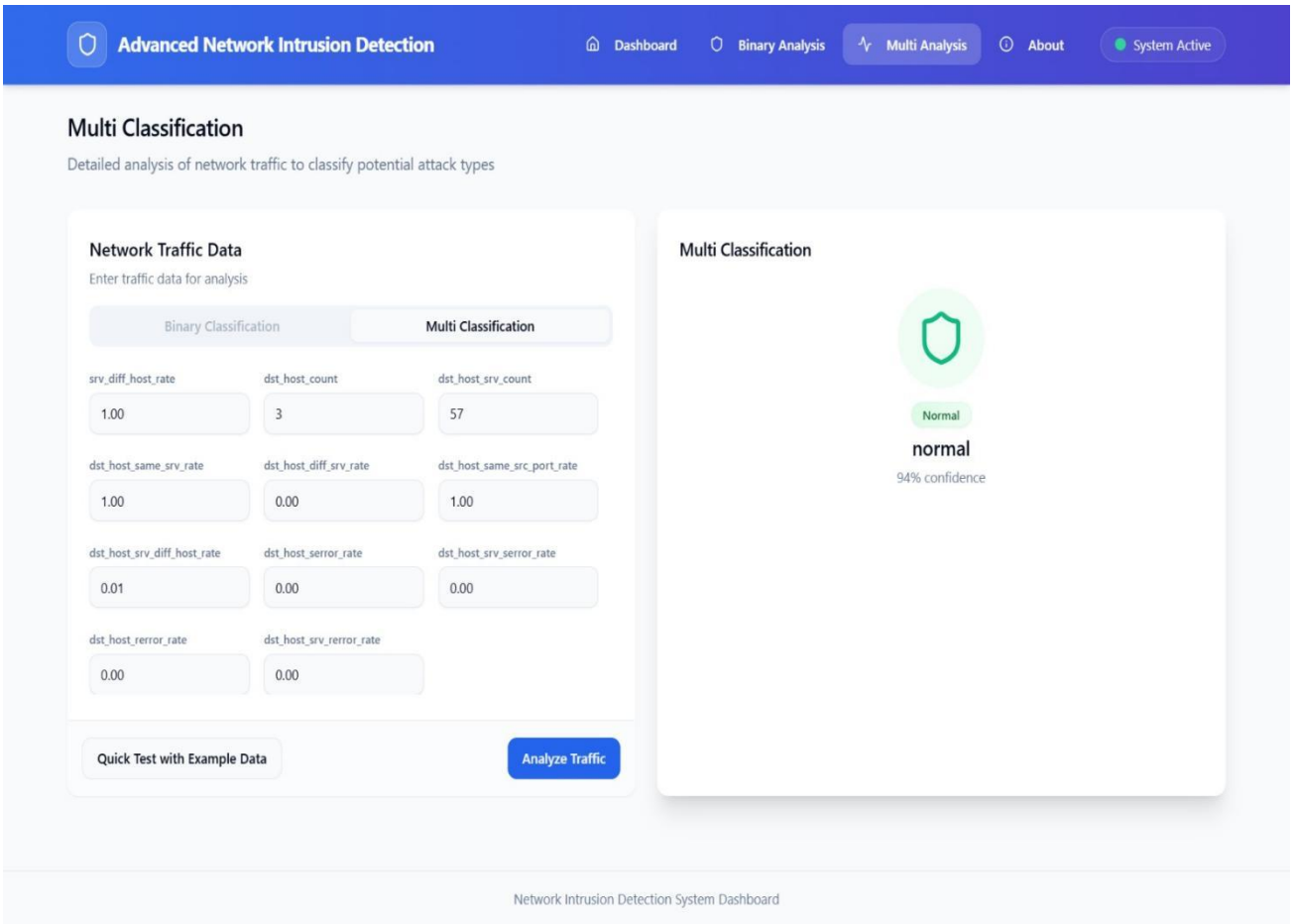
```
        for col in self.train_columns:
            if col not in single_df:
                single_df[col] = 0  # Add missing features with default value 0
        # Ensure correct column order
        single_df = single_df[self.train_columns]
        # Normalize numerical features using the saved StandardScaler
        single_df.iloc[:, :41] = self.scaler.transform(single_df.iloc[:, :41]).astype(float)

        return single_df
    def predict(self, input_data):
        """Make a multi-class classification prediction"""
        try:
            # Preprocess the data
            single_df = self.preprocess(input_data)
            # Predict attack type using the trained model
            y_pred = self.rf.predict(single_df)
            # Get prediction probabilities
            y_prob = self.rf.predict_proba(single_df)
            confidence = round(max(y_prob[0]) * 100)
            # Convert numeric prediction back to attack label
            y_pred_label = self.le.inverse_transform(y_pred)
            attack_type = y_pred_label[0].lower()
            # Get attack details
            details = self.attack_categories.get(attack_type, {
                'category': 'Unknown',
                'severity': 'medium',
                'impact': 'Unknown impact'
```

```
        })
        # Format result
        result = {
            "prediction": y_pred_label[0],
            "confidence": confidence,
            "details": details
        }
        return result, None
    except Exception as e:
        return None, str(e)
```

5.3 Screenshots of webpages




Advanced Network Intrusion Detection
Dashboard
Binary Analysis
Multi Analysis
About
System Active
Binary Classification
Analyze network traffic to determine if it's normal or potentially malicious
Network Traffic Data
Enter traffic data for analysis
Binary Classification
Multi Classification
srv_diff_host_rate
0.00
dst_host_count
255
dst_host_srv_count
255
dst_host_same_srv_rate
0
dst_host_diff_srv_rate
2
dst_host_same_src_port_rate
0
dst_host_srv_diff_host_rate
0.00
dst_host_serror_rate
0.00
dst_host_srv_serror_rate
2
dst_host_rerror_rate
0.00
dst_host_srv_rerror_rate
2
Quick Test with Example Data
Analyze Traffic
Binary Classification
Attack Detected
Potential security threat identified
56% confidence
Network Intrusion Detection System Dashboard


Advanced Network Intrusion Detection
Dashboard
Binary Analysis
Multi Analysis
About
System Active
Multi Classification
Detailed analysis of network traffic to classify potential attack types
Network Traffic Data
Enter traffic data for analysis
Binary Classification
Multi Classification
srv_diff_host_rate
1.00
dst_host_count
3
dst_host_srv_count
57
dst_host_same_srv_rate
1.00
dst_host_diff_srv_rate
0.00
dst_host_same_src_port_rate
1.00
dst_host_srv_diff_host_rate
0.01
dst_host_serror_rate
0.00
dst_host_srv_serror_rate
0.00
dst_host_rerror_rate
0.00
dst_host_srv_rerror_rate
0.00
Quick Test with Example Data
Analyze Traffic
Multi Classification
Normal
normal
94% confidence
Network Intrusion Detection System Dashboard

6.Testing

6.1 Testing strategies used

- **Functionality Testing:**
  - Validates that individual modules perform as expected.
  - Ensures processes like data preprocessing, model training, and threat visualization are accurate.
- **Accuracy Validation:**
  - Tests the system's ability to detect and classify network threats effectively.
  - Benchmarked using the KDD Cup 1999 dataset.
- **Integration Testing:**
  - Ensures seamless interaction between various modules (data preprocessing, machine learning, and user interface).
- **Security Testing:**
  - Verifies that sensitive data is handled securely and prevents unauthorized access.

6.2 Test case reports

- **Unit Testing:**
  - Focuses on verifying the functionality of individual components, such as ML algorithms and data processing.
- **Integration Testing:**
  - Tests the interaction between modules, ensuring end-to-end workflows like data upload to threat detection work correctly.
- **Accuracy Testing:**
  - Evaluates the detection of attack types like DoS, Probe, R2L, and U2R.
  - Ensures minimal false positives and negatives during threat classification.
- **UI Testing:**

- Validates that the user interface is intuitive, responsive, and displays visualizations like like graphs and reports accurately.

7.Conclusion

The project successfully addressed the growing need for effective intrusion detection in dynamic and complex network environments. By utilizing machine learning algorithms and comprehensive datasets such as KDD Cup 1999, the system was able to detect and classify a wide range of cyber threats, including DoS, Probe, R2L, and U2R attacks. The intuitive user interface provided a clear visualization of detected threats, ensuring accessibility and ease of use for network administrators. Rigorous testing confirmed the system's reliability, efficiency, and ability to operate in resource-constrained environments, fulfilling its objective of enhancing network security.

8.Bibliography

1. S. Widup, A. Pinto, D. Hylender, G. Bassett, and P. Langlois. Data Breach Investigations Report. [Online]. Available: https://www.wired.com/images_blogs/threatlevel/2011/04/Verizon -2011-DBIR_04-13-11.pdf

2. J. Andrew, R. J. Eunice, and J. Karthikeyan, "An anonymizationbased privacy-preserving data collection protocol for digital health data," Front Public Health, vol. 11, Mar. 2023.

3. O. Aslan and R. Samet, "A comprehensive review on malware detection approaches," IEEE Access, vol. 8, pp. 6249–6271, 2020.

4. Z. Yang et al., "A systematic literature review of methods and datasets for anomaly-based network intrusion detection," Comput. Secur., vol. 116, 102675,May 2022.

5. A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: Techniques, datasets and challenges," Cybersecurity, vol. 2, no. 1, 20, Dec. 2019.

6. M. Masdari and H. Khezri, "A survey and taxonomy of the fuzzy signature-based intrusion detection systems," Appl. Soft. Comput., vol. 92, 106301, Jul. 2020.

7. S. Dwivedi, M. Vardhan, S. Tripathi, and A. K. Shukla, "Implementation of adaptive scheme in
evolutionary technique foranomaly-based intrusion detection," Evol. Intell., vol. 13, no. 1, pp. 103–117, Mar. 2020.

8. Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," Transactions on Emerging Telecommunications Technologies, vol. 32, no. 1, Jan. 2021.

9. D. E. Kim and M. Gofman, "Comparison of shallow and deep neural networks for network intrusion detection," in Proc. 2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC), 2018, pp. 204–208.

10. Y. Zhong et al., "HELAD: A novel network anomaly detection model based on heterogeneous ensemble learning," Computer Networks, vol. 169, 107049, Mar. 2020.

11. Wang, Z., Jiang, D., Huo, L., Yang, W. (2021). An Efficient Network Intrusion Detection Approach Based on Deep Learning. doi: 10.1007/s11276-021-02698-9

12. Nguyen Dang, K. D., Fazio, P., Voznak, M. (2024). A Novel Deep Learning Framework for Intrusion Detection Systems in Wireless Networks. doi: 10.3390/fi16080264

13. Singh, A., Amutha, J., Nagar, J., Sharma, S. (2022). A Deep Learning Approach to Predict the Number of k- Barriers for Intrusion Detection Over a Circular Region Using Wireless Sensor doi:10.48550/arXiv.2208.11887 Networks.

14. Sultan, M. T., El Sayed, H., Khan, M. A. (2023). An Intrusion Detection Mechanism for MANETs Based on Deep Learning Artificial Neural Networks (ANNs). doi:10.48550/arXiv.2303.08248

15. Yang, K., Ren, J., Zhu, Y., Zhang, W. (2018). Active Learning for Wireless IoT Intrusion Detection. doi: 10.48550/arXiv.1808.01412

16. Gupta, A., Pandey, O. J., Shukla, M., Dadhich, A., Mathur, S., Ingle, A. (2021). Computational Intelligence- Based Intrusion Detection Systems for Wireless Communication. doi: 10.48550/arXiv.2105.03204

17. Zhang, J., He, T. (2022). AI-Based Solutions for Intrusion Detection in Next-Generation Wireless Networks. doi: 10.1109/ACCESS.2022.3140198

18. Kim, D. E., Gofman, M. (2018). Comparison of Shallow and Deep Neural Networks for Network Intrusion Detection. doi: 10.1109/CCWC.2018.8301638

19. Zhong, Y., Wang, Y., Wang, L., Li, J., Li, Z. (2020). HELAD: A Novel Network Anomaly Detection Model Based on Heterogeneous Ensemble Learning. doi: 10.1016/j.comnet.2020.107049

20. Ahmad, Z., Khan, A. S., Shiang, C. W., Abdullah, J., Ahmad, F. (2021). Network Intrusion Detection System: A Systematic Study of Machine Learning and Deep Learning Approaches. doi: 10.1002/ett.4147

21. S., Riyaz., G., Sannasi. (2020). A Deep Learning Approach for Effective Intrusion Detection in Wireless Networks Using CNN. doi: 10.1007/s00500-020-05017- 0

22. S., Wang., et al. (2019). Intrusion Detection for WiFi Network: A Deep Learning Approach. doi: 10.1007/978- 3-030-06158-6_10

23. Enhanced Intrusion Detection in Wireless Sensor Networks Using Deep Learning. (2024). doi: 10.1007/s11042-024-19305-6

24. Real-Time Intrusion Detection in Wireless Network: A Deep Learning Approach. (2020). doi: 10.1109/access.2020.9178792

25. Dinh Nguyen, Khoa., Fazio, P., Voznak, M. (2024). A Novel Deep Learning Framework for Intrusion Detection Systems in Wireless Networks. doi: 10.3390/1999-5903

26. Kimanzi, R., Kimanga, P., Cherori, D., Gikunda, P. K. (2024). Deep Learning Algorithms Used in Intrusion Detection Systems: A Review. doi: 10.4856/2402-17020

27. Feasibility of Deep Learning Sensor Network Intrusion Detection. (2019). doi: 10.1109/8653348

28. Patel, H., Mehta, P., Bhatt, P. (2023). Intrusion Detection Using Federated Deep Learning in IoT Networks. doi: 10.1007/s00521-023-08357-7

29. Liu, B., Xu, Y., Chen, M. (2021). Intrusion Detection in 5G Networks: A Deep Learning-Based Approach. doi: 10.1109/ACCESS.2021.3085481

30. Choi, Y., Han, J., Seo, S. (2022). A Lightweight Intrusion Detection Model Using Edge AI for Wireless Sensor Networks. doi: 10.1109/TNSM.2022.315