B.M.S. COLLEGE OF ENGINEERING BENGALURU
Autonomous Institute, Affiliated to VTU

OOMD Mini Project Report

NETWORK INTRUSION DETECTION SYSTEM

Submitted in partial fulfillment for the award of degree of

Bachelor of Engineering
in
Computer Science and Engineering

Submitted by:

PADMASHREE JAIN D (1BM23CS223)

PARNIKA DEEPAK BHAT (1BM23CS226)

POOJA C HADLI (1BM23CS232)

NANDANA KISHORE C NAIR (1BM23CS364)

Department of Computer Science and Engineering
B.M.S. College of Engineering
Bull Temple Road, Basavanagudi, Bangalore 560 019
2025-26

B.M.S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, PADMASHREE JAIN D (1BM23CS223), PARNIKA DEEPAK BHAT (1BM23CS226), POOJA C HADLI (1BM23CS232) and NANDANA KISHORE C NAIR (1BM23CS364) students of 5th Semester, B.E, Department of Computer Science and Engineering, BMS College of Engineering, Bangalore, hereby declare that, this OOMD Mini Project entitled "NETWORK INTRUSION DETECTION SYSTEM" has been carried out in Department of CSE, B.M.S. College of Engineering, Bangalore during the academic semester August 2025- December 2025. I also declare that to the best of our knowledge and belief, the OOMD mini Project report is not from part of any other report by any other students.

Signature of the Candidate

PADMASHREE JAIN D (1BM23CS223)

PARNIKA DEEPAK BHAT (1BM23CS226)

POOJA C HADLI (1BM23CS232)

NANDANA KISHORE C NAIR (1BM23CS364)

B.M.S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the OOMD Mini Project titled “**NETWORK INTRUSION DETECTION SYSTEM”** has been carried out by PADMASHREE JAIN D (1BM23CS223), PARNIKA DEEPAK BHAT (1BM23CS226), POOJA C HADLI (1BM23CS232) and NANDANA KISHORE C NAIR (1BM23CS364) during the academic year 2025-2026.

Signature of the Faculty in Charge (Your Guide Name)

Table of Contents

Chapter 1: Problem Statement

The fast-growing complexity of cyber-attacks makes it increasingly difficult for current Network Intrusion Detection Systems (NIDS) to provide reliable and adaptive protection. Traditional signature-based IDS can only detect known attacks, fail to identify zero-day threats, and require frequent manual updates, making them inefficient for modern, dynamic networks. Although deep learning–based IDS models offer improvements, many still rely heavily on dataset-specific tuning and manual feature engineering. Their limited adaptability often results in high false positive rates, reducing practicality in real-world deployments.

This creates a clear need for an intelligent and automated intrusion detection solution capable of learning from evolving network behavior without constant human intervention. The core research problem is to design a NIDS that can effectively capture both spatial patterns within traffic data and temporal relationships across time, enabling accurate detection of both known and emerging threats. The system must also maintain low false alarm rates to ensure operational usability and reduce the burden on security teams. Developing such an adaptive and scalable IDS would significantly improve network security by providing robust protection in real-world, continuously changing environments.

Chapter 2: Software Requirement Specification

Software Requirements Specification (SRS) for Network Intrusion Detection System (NIDS)

1. Introduction

1.1 Intent of this Document

The purpose of this document is to define the functional and non-functional requirements for the development of a Network Intrusion Detection System (NIDS) that utilizes machine learning and deep learning techniques to detect anomalous network activities in real time.

It will serve as a guide for developers, testers, and stakeholders to ensure consistent understanding of the project's objectives, scope, and deliverables.

1.2 Scope of this Document

The NIDS is aimed at the monitoring of live network traffic, extraction of flow-based features, pattern analysis, and classification of network events as normal or malicious.

It integrates with tools like CICFlowMeter or Zeek for traffic capture and feature extraction, using a CapsNet + BiLSTM deep learning model for detection.

The system will : Detect and classify attack types such as DoS, DDoS, Botnet, and Exploit. Animate users in real time.Support both offline (dataset-based) and live network monitoring.

1.3 Summary

The proposed NIDS system will be implemented in an institutional, enterprise, or research network.
It will consist of:

1. A module for collecting data: PCAP capture and flow generation.
2. Feature pre-processing module (scaling, encoding and normalization)
3. A machine learning engine: CapsNet + BiLSTM
4. A real-time alerting and visualization interface.

2. General Description

The Network Intrusion Detection System will continuously monitor traffic from a specified network interface or dataset.

This model will process packet data, extract flow features, and then classify activities by using pre-trained models.

2.1 System Objectives

1. Precisely detect intrusions with ML-based detection.
2. Provide real-time alerts on network anomalies.
3. Support training on multiple datasets such as CIC-IDS2017, UNSW-NB15.
4. Allow modular updates for new attack signatures or model improvements.

2.2 User Classes and Characteristics

User Type	Description	Technical Expertise
Administrator	Configures NIDS, manages models, views full logs	High
Security Analyst	Monitors alerts, analyzes anomalies	Medium
General User	View alerts on dashboard, basic control	Low

2.3 Operating Environment

1. Platform: macOS / Linux / Windows
2. Languages: Python 3, Java (for CICFlowMeter)
3. Frameworks: PyTorch, FastAPI
4. External Tools: Zeek / CICFlowMeter / tcpreplay
5. Database: SQLite / JSON logs extendable to PostgreSQL

3. Functional Requirements

3.1 Data Collection and Feature Extraction

1. Capture network traffic in real-time using CICFlowMeter or Zeek.
2. Support PCAP replay using tcpreplay for offline testing.
3. Generate labeled CSV flow files with extracted features.

3.2 Data Preprocessing

1. Load CSV files and standardize feature columns.
2. Handle missing values, categorical encoding, and scaling.
3. Save preprocessed data and artifacts: scaler, label encoder.

3.3 Model Training

1. Train a deep learning model, CapsNet + BiLSTM, based on preprocessed datasets.
2. Save the trained models as serialized .pt files.
3. Generate performance metrics: precision, recall, F1-score, confusion matrix.

3.4 Real-Time Intrusion Detection

1. Continuously monitor live network flows.
2. Predict the traffic label (Normal/Attack) using the trained model.
3. Display live detection output with timestamp, source/destination IP, and attack label.
4. Log all detections into a CSV or database.

3.5 Alerting and Visualization

1. Send attack notifications by terminal or GUI.
2. Provide dashboard view - FastAPI web server.
3. Generate daily/weekly summary reports.

4. Interface Requirements

4.1 User Interface

1. Simple web dashboard - FastAPI + HTML frontend.
2. Live terminal display of detections
3. Graphical metrics dashboard: accuracy, F1-score, confusion matrix.

4.2 Integration Interfaces

1. Integration with Zeek logs or CICFlowMeter CSV outputs.
2. Optional integration with email/SMS notification systems.
3. REST API endpoint /predict to classify flow features in real-time.

5. Performance Requirements

Metric	Target
Response Time	Detection Latency ≤ 2 seconds per flow
Accuracy	≥ 97% detection accuracy on benchmark datasets
Throughput	Handle up to 10,000 flows/minute
Scalability	Support the monitoring of several interfaces simultaneously.
Model Load Time	≤ 3 seconds

6. Design Constraints

6.1 Hardware Limitations

1. Standard laptop or server with ≥ 8GB RAM, 4-core CPU, optional GPU.
2. Compatible with standard NIC interfaces.

6.2 Software Dependencies

1. Python 3.10+, PyTorch, pandas, scikit-learn.
2. CICFlowMeter.jar (Java 11+).
3. Zeek and tcpreplay installed (via Homebrew or apt)

7. Non-Functional Requirements

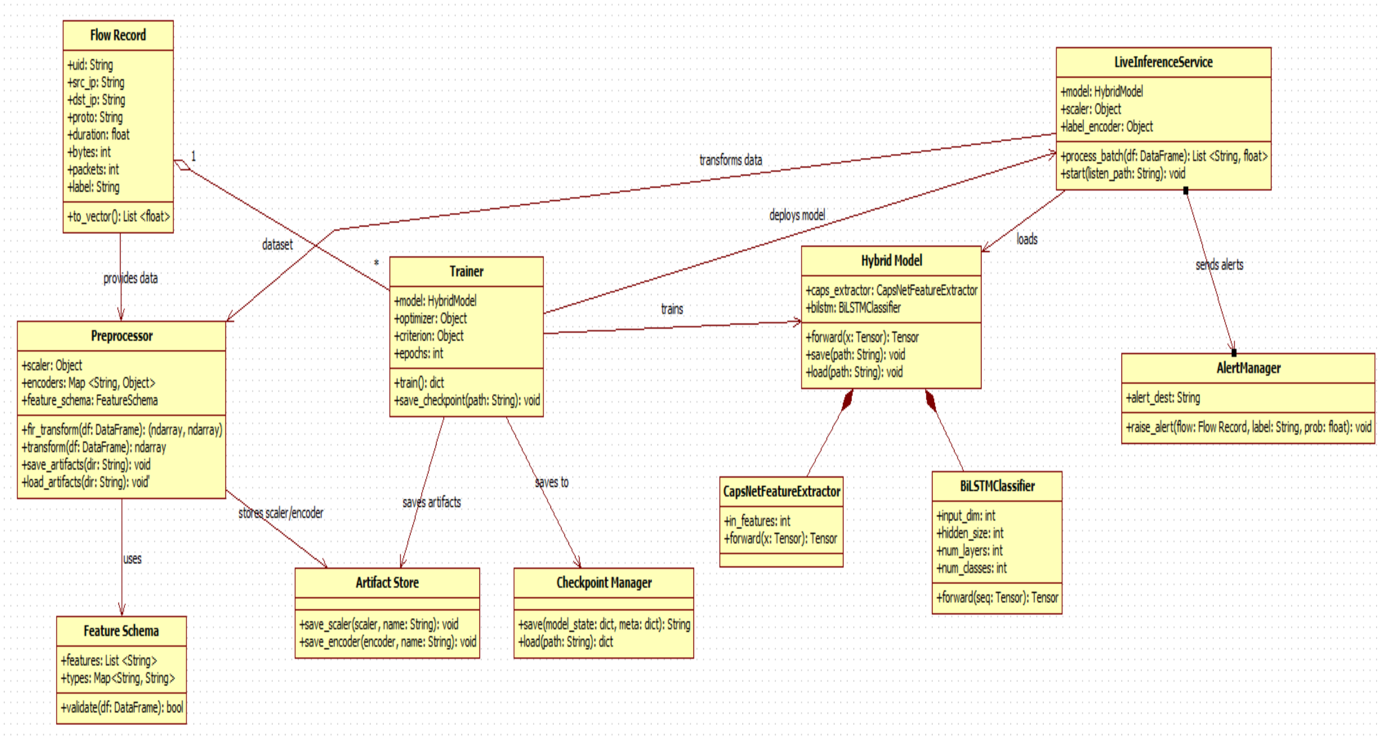
Attribute	Description
Security	Authentication required for admin access; encrypted log storage
Reliability	Auto-restart if the monitoring process fails
Scalability	Modular architecture for adding new detection models
Portability	Runs on Linux/macOS/Windows
Usability	Simple CLI + optional GUI dashboard
Maintainability	Modular Python packages, clear folder structure
Reusability	Shared preprocessing and model utilities
Compatibility	Works with common datasets and network capture tools

8. Preliminary Schedule and Budget

Phase	Duration	Deliverables
Requirement Analysis	2 weeks	SRS Document
Dataset Preparation	2 weeks	Processed CSVs
Model Development	4 weeks	Trained CapsNet + BiLSTM model
System Integration	3 weeks	Combined live inference pipeline
Testing & Evaluation	3 weeks	Reports and metrics
Deployment	2 weeks	Working prototype

Estimated Duration: 4 months
Estimated Budget: $7,000 (for hardware, dataset storage, compute, and tools)

Chapter 3: Class Modeling



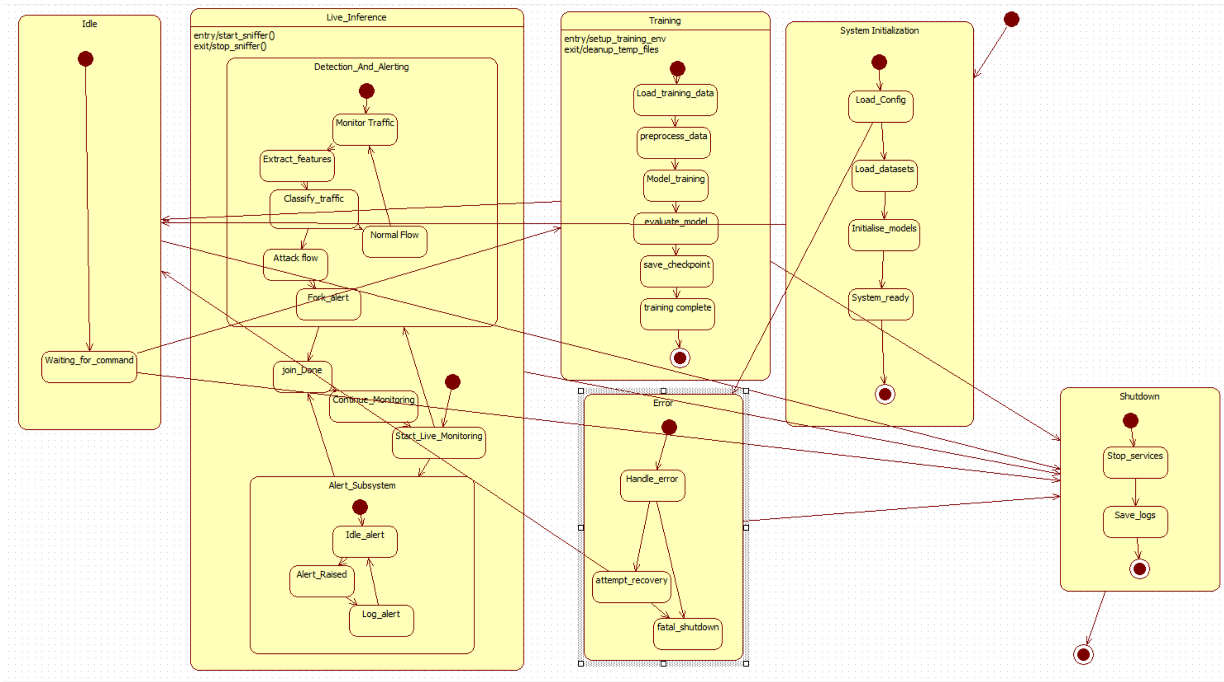
The raw data is processed by a Preprocessor, which uses a `reader` and `recorder` to ingest data and employs transformation methods (`transform`, `reset_splitText`) to clean and prepare the data, likely converting it into a numerical `ndarray` format suitable for machine learning.

The central intelligence is the Hybrid Model, a composite architecture. It consists of a CaptiveFeatureExtractor for initial feature extraction and a BLSTMClassifier (a Bidirectional LSTM neural network) for classification. This hybrid design is powerful for NIDS, as it can capture complex spatial and temporal patterns in network traffic, which are often indicative of attacks.

Model training is managed by the Trainer within a `hybridModel` module, which handles epochs and checkpoints. For real-time operation, the UserInferenceService loads the trained `hybridModel` and provides a `reprocess_backlight` method to analyze new data frames and return intrusion predictions.

Finally, an AgentManager acts as the orchestration layer. It takes the processed `Flow Record` and the inference results, and is responsible for taking action, such as triggering `update alerts` based on the detected threats. This creates a cohesive pipeline from raw network flow to actionable security intelligence.

Chapter 4: StateModeling



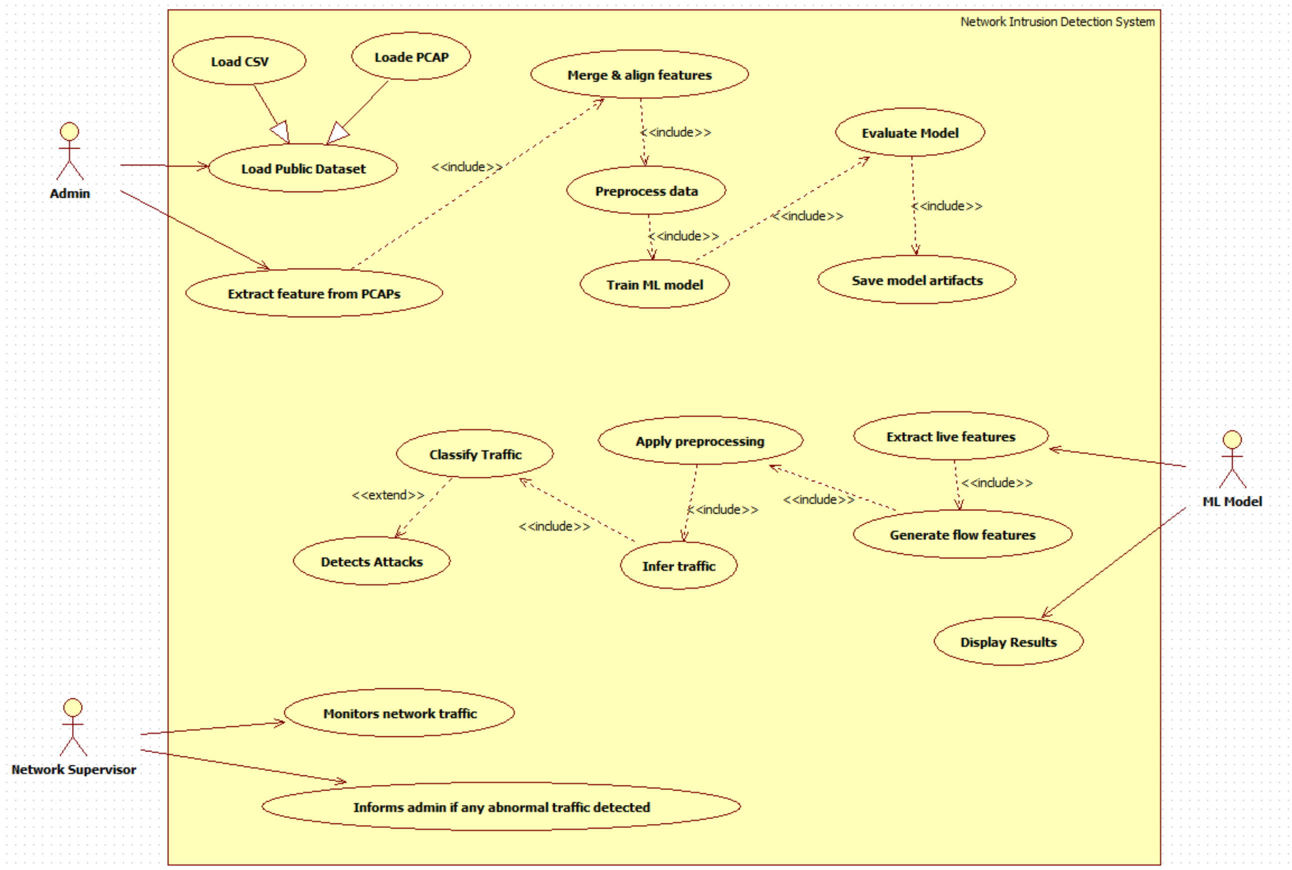
The system begins in an Idle state. Upon receiving a trigger, it enters the Live_Inference state, which encompasses the core real-time intrusion detection process. This process is a cycle of monitoring network traffic, connecting and extracting relevant features from the data, and classifying the traffic as either Normal Flow or Attack Flow. If an attack is detected, the system transitions to the Alert_Subsystem state, where an alert is generated and presumably logged or sent to an administrator. If the attack is severe, it may lead to a Join Down or Confringe_Mountains state, suggesting potential system isolation or active countermeasure deployment.

Simultaneously, the system has a separate Training pathway. From Idle, it can enter a training mode where it uses stored data to preprocess, train a new machine learning model, and save it once training is complete, making an available_model for the live detection process.

The model also accounts for system management states like System Installation for initial setup and an Error state. The Error state includes a sub-process for handling faults, attempting recovery, and logging the incident before (ideally) returning to a stable state. The Situation state handles administrative duties like showing system previews and saving logs, ensuring operational oversight and maintenance.

Chapter 5: Interaction Modeling

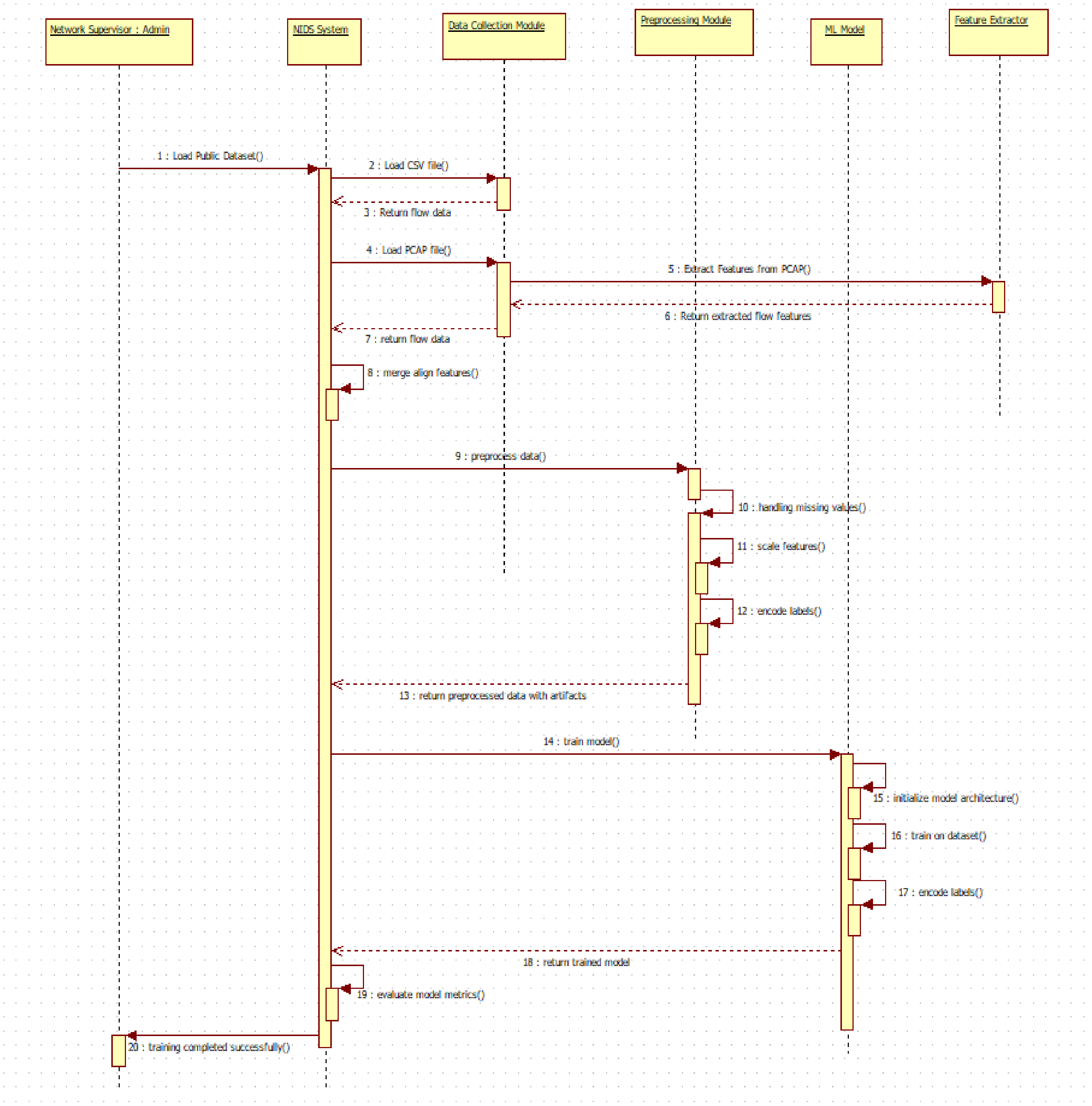
Use Case Diagram:



The process begins with the Model Training pipeline. A security engineer (or system) loads a public dataset or PCAP files containing both normal and malicious network traffic. The system then merges these data sources, aligns their features, and preprocesses the data (e.g., cleaning, normalization). This prepared data is used to train the Hybrid Model, which is subsequently Evaluated for accuracy. If performance is satisfactory, the model artifacts are saved for deployment, completing the training cycle.

The second is Real-time Traffic Inference & Monitoring. Here, the system continuously Captures live network traffic. For this traffic, it Generates flow features and Extracts live features compatible with the trained model. It then applies preprocessing and uses the saved model to infer traffic, classifying it as normal or an attack. If an attack is detected, the system executes the crucial steps to Delete Attacks and Informs the admin by generating an alert. Finally, the results are displayed on a dashboard for security oversight.

Sequence Diagram:



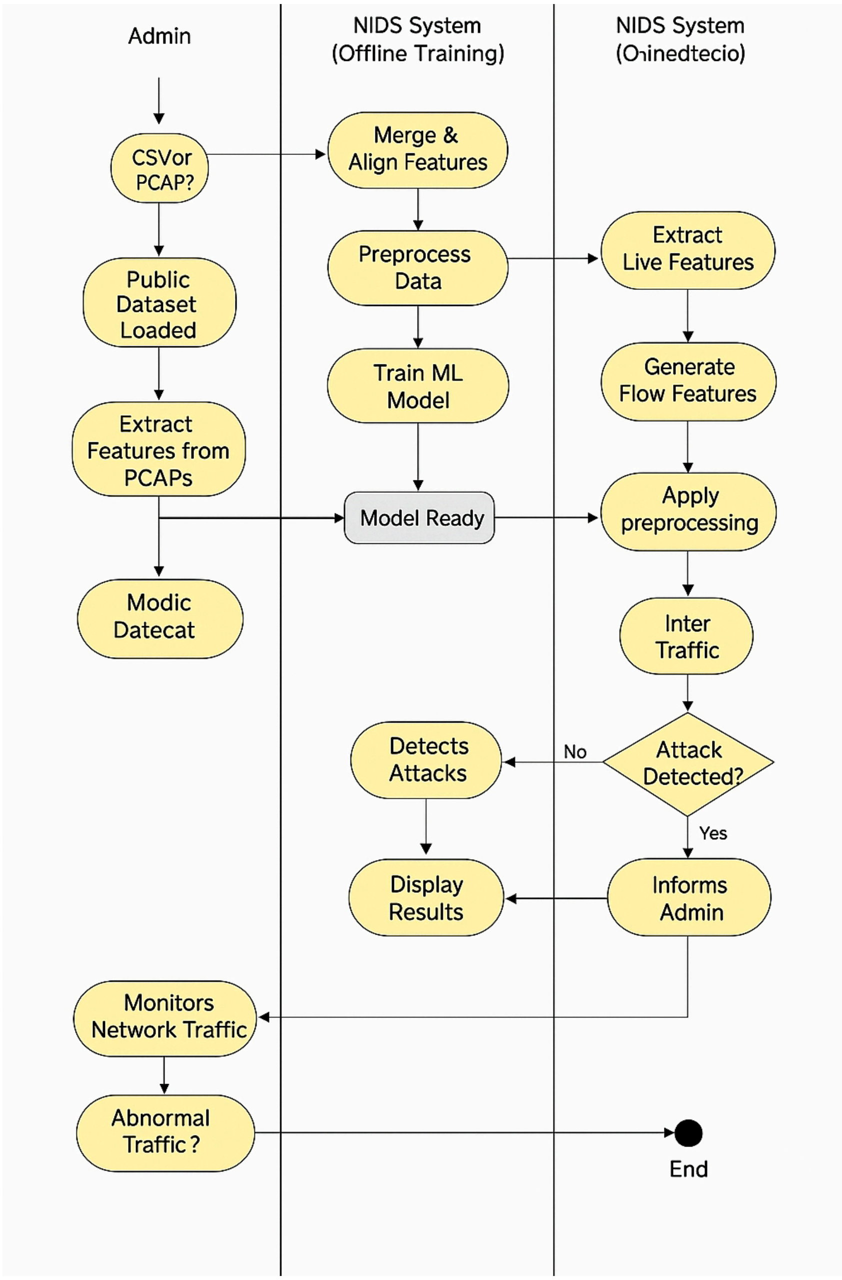
The process begins when the Admin issues a command to Load Public Dataset. The system responds by loading a CSV file and returning the initial flow data. In parallel, the Admin also commands the system to Load PCAP files. The system executes an Extract Features from PCAP routine, returning the extracted flow features.

The system then combines these two data streams by executing a merge align feature operation. With the unified dataset prepared, the system begins the crucial preprocessing stage. This involves a sequence of steps: preprocess data, handling missing value, scale features, and encode labels. Upon completion, it returns the preprocessed data along with the necessary preprocessing artifacts (like scalers and encoders) for use during inference.

The core training phase is triggered by the train model command. The system first initializes the model architecture and then proceeds to train on the preprocessed dataset. The sequence includes another encoded label step, ensuring target variables are correctly formatted for the model. The system then returns the trained model object.

Finally, the workflow concludes with the system evaluating the model's performance metrics and confirming that the training completed successfully. This end-to-end sequence effectively captures the data preparation, model training, and validation stages required to deploy a machine learning model for network intrusion detection.

Activity diagram



The process begins with an Admin providing initial data, making a choice between loading a CSV or PCAP file, or using a Public Dataset. This triggers the Offline Training lane. The system first performs Feature Extraction from the provided data, then Merges & Aligns Features to create a unified dataset. This data is Preprocessed and used to Train the ML Model. Once the training is complete, the system reaches the Model Ready state.

Concurrently, the real-time Monitoring lane begins. The system continuously Monitors Network Traffic, Generates Flow Features from the live packets, and Extracts Live Features. These features are then fed into the same Preprocessing sub-process used in training, ensuring consistency. The prepared data is passed to the ready model to Infer Traffic.

A critical decision node, Attack Detected?, follows. If the traffic is classified as normal, the system simply Displays Results and the monitoring loop continues. However, if abnormal traffic is detected, the system Detects Attacks, Informs the Admin, and then the process reaches its End state for that specific detection cycle, while the overall monitoring activity continues. This diagram effectively illustrates how the offline-trained model is seamlessly integrated into a live, automated security monitoring pipeline.

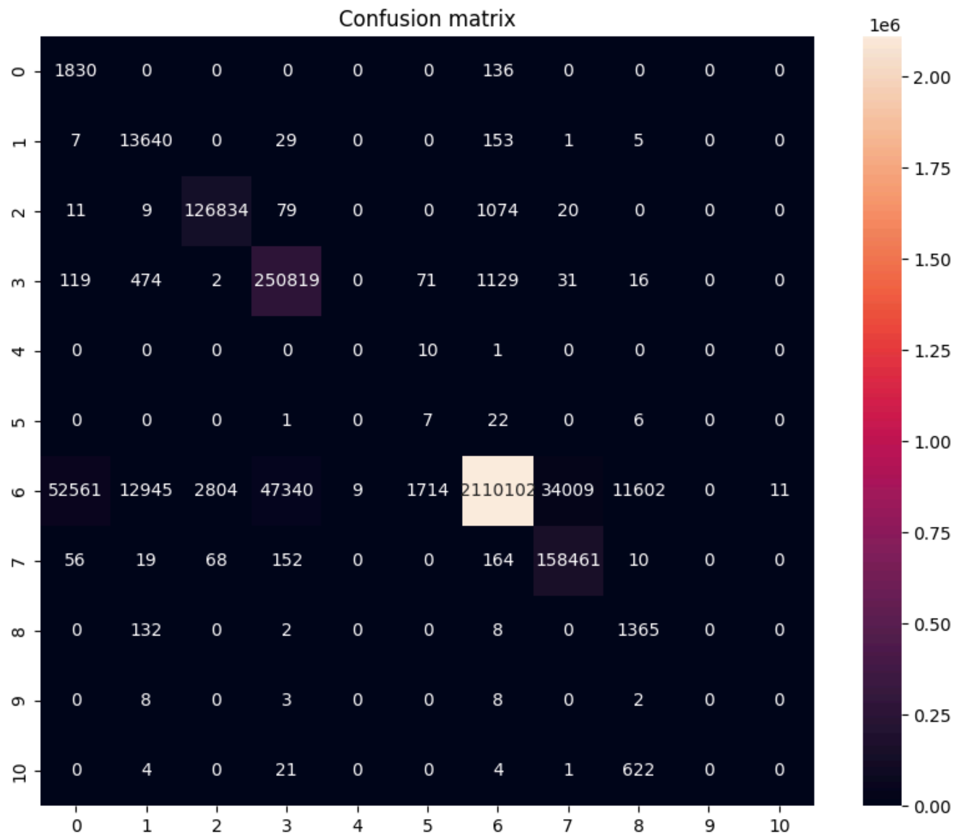
Chapter 6: UI Design with Screenshots

```
[9:29:24 PM] Checking preprocessed artifacts...
[9:29:24 PM] Artifacts found in data/artifacts/
[9:29:25 PM] Loading trained model...
[9:29:25 PM] Model: random_forest_model.pkl
[9:29:26 PM] Activating virtual environment...
[9:29:26 PM] Virtual environment activated
[9:29:28 PM] Running model evaluation...
[9:29:29 PM] Accuracy: 99.2%
[9:29:29 PM] Precision: 98.7%
[9:29:29 PM] Recall: 99.1%
[9:29:30 PM] Opening confusion matrix visualization...
[9:29:30 PM] Confusion matrix displayed
[9:29:32 PM] Starting live detection...
[9:29:32 PM] Monitoring network traffic...
[9:29:32 PM] Press 'Stop Demo' to terminate
```

```
PS D:\NetworkIDS> …
====================================
  NETWORK INTRUSION DETECTION DEMO
====================================

[1/9] Navigating to project...
Current directory: D:\NetworkIDS

[2/9] Project structure:
Folder PATH listing for volume New Volume
Volume serial number is 0000001B 0095:7C06
D:\NETWORKIDS\SRC
|   capsnet.py
|   config.yaml
|   dataset.py
|   Dockerfile
|   evaluate.py
|   feature_extractor.py
|   hybrid_model.py
|   live_inference.py
|   merge_align.py
|   preprocess.py
|   server.py
|   test_pcap_inference.py
|   train.py
|   utils.py
|   __init__.py
|
\---__pycache__

[3/9] Dataset files:


[4/9] Preprocessed data:

[5/9] Trained model:
```

```
[6/9] Activating virtual environment...

[7/9] Evaluating model performance...
D:\NetworkIDS\venv\Lib\site-packages\sklearn\metrics\_classification.py:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0
 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
D:\NetworkIDS\venv\Lib\site-packages\sklearn\metrics\_classification.py:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0
 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
D:\NetworkIDS\venv\Lib\site-packages\sklearn\metrics\_classification.py:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0
 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
              precision    recall  f1-score   support

           0     0.0335    0.9308    0.0647      1966
           1     0.5009    0.9859    0.6643     13835
           2     0.9778    0.9907    0.9842    128027
           3     0.8404    0.9927    0.9102    252661
           4     0.0000    0.0000    0.0000        11
           5     0.0039    0.1944    0.0076        36
           6     0.9987    0.9283    0.9622   2273097
           7     0.8231    0.9970    0.9017    158930
           8     0.1002    0.9058    0.1804      1507
           9     0.0000    0.0000    0.0000        21
          10     0.0000    0.0000    0.0000       652

    accuracy                         0.9408   2830743
   macro avg     0.3890    0.6296    0.4250   2830743
weighted avg     0.9700    0.9408    0.9524   2830743

Saved confusion matrix to results/confusion_matrix.png

[8/9] Opening confusion matrix...

========================================
         DEMO COMPLETE!
========================================
```

```
Name                                                         Size(MB)
----                                                         --------
Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv                91.65
Friday-WorkingHours-Afternoon-PortScan.pcap_ISCX.csv            97.16
Friday-WorkingHours-Morning.pcap_ISCX.csv                       71.89
Monday-WorkingHours.pcap_ISCX.csv                               256.2
Thursday-WorkingHours-Afternoon-Infilteration.pcap_ISCX.csv    103.69
Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv          87.77
Tuesday-WorkingHours.pcap_ISCX.csv                              166.6
Wednesday-workingHours.pcap_ISCX.csv                           272.41
label_encoder.pkl                                                   0
master_features.pkl                                                 0
scaler.pkl                                                          0
X_all.npy                                                      842.28
X_sample.npy                                                   168.46
y_all.npy                                                        21.6
y_sample.npy                                                     4.32
best.pt                                                          0.63
best.py                                                             0
model_info.json                                                     0
```



This matrix shows how well the model correctly identifies each attack type.

Bright diagonal cells indicate high correct classifications, while off-diagonal cells show misclassifications.