**Design Document for Real-Time Network Intrusion Detection System**

**Version:** 1.0

**Group ID:** F24PROJECT5982F

**Supervisor:** Laraib Sana

**Date:** 02/03/2025



Table of Contents

[1. Introduction 3](#__RefHeading___Toc1429_2145273382)

[2. System Overview 3](#__RefHeading___Toc1431_2145273382)

[3. Design Objectives 4](#__RefHeading___Toc1433_2145273382)

[4. Architecture Design 4](#__RefHeading___Toc1435_2145273382)

[4.1. Architecture Diagram 6](#__RefHeading___Toc1437_2145273382)

[5. User Interface Design 7](#__RefHeading___Toc1439_2145273382)

[5.1. UI Layout 8](#__RefHeading___Toc1441_2145273382)

[5.2. Screenshots 8](#__RefHeading___Toc1443_2145273382)

[6. Component Design 9](#__RefHeading___Toc1445_2145273382)

[7. Data Flow Diagrams 11](#__RefHeading___Toc1447_2145273382)

[8. Conclusion and Future Work 13](#__RefHeading___Toc1449_2145273382)

# 1. Introduction

This design document details the architecture and design decisions for the Real-Time Network Intrusion Detection System. The system is engineered to enhance network security by capturing real-time network data via Wireshark, pre-processing the data, and leveraging advanced ensemble learning models (including TabNet, CatBoost, and LightGBM) for accurate intrusion detection. The primary goal is to provide a robust, scalable, and user-friendly solution that can detect and alert on malicious network activities in real time. This document serves as a blueprint for developers, stakeholders, and security analysts, outlining both high-level concepts and detailed design elements that drive the project.

# 2. System Overview

The Real-Time Network Intrusion Detection System is composed of several integrated components that work together to monitor and analyze network traffic. The system is structured into four key layers:

* **Data Capture Layer**:  
  Utilizes Wireshark to capture real-time network traffic. This component exports data as CSV files, capturing essential network parameters such as protocol types, IP addresses, packet sizes, and timestamps.
* **Data Preprocessing Layer**:  
  Once the network data is captured, it is cleaned, encoded, and normalized to remove noise and inconsistencies. The pre-processed data is stored in CSV format, ensuring that it is structured and ready for further analysis.
* **Machine Learning & Analysis Layer**:  
  The pre-processed data is used to train advanced ensemble learning models. This layer includes:
  + **Model Training Module**: Uses techniques like TabNet, CatBoost, and LightGBM to learn from the network data.
  + **Model Evaluation Module**: Assesses the performance of the models using metrics such as accuracy, precision, recall, F1-score, and ROC-AUC.
  + **Intrusion Detection Module**: Deploys the best-performing model to classify network traffic as normal or malicious in real time.
* **User Interface Layer**:  
  A web application, built using Flask, provides a user-friendly interface where users can:
  + Upload CSV files containing network traffic data.
  + Trigger the analysis process.
  + View detailed results and visualizations of the intrusion detection outputs, including key metrics and network parameters influencing the classification.

Hardware requirements primarily include standard server configurations for processing and storage, while software components comprise Python-based tools (Flask for the web application, machine learning libraries, and Wireshark for data capture).

# 3. Design Objectives

The design of this system is driven by the following objectives:

* **Scalability**:  
  Ensure that the system can efficiently process increasing volumes of network data. The architecture is designed to scale both vertically (more processing power per node) and horizontally (additional nodes).
* **Real-Time Processing**:  
  Achieve near real-time data capture and analysis to allow timely detection of network intrusions. The system’s components are optimized for minimal latency between data capture and result display.
* **User-Friendliness**:  
  Provide an intuitive and responsive user interface. The web application is designed to be accessible and straightforward, enabling users to easily upload data, monitor the analysis process, and interpret the results.
* **Security**:  
  Protect both the system and user data from unauthorized access. This includes implementing secure file handling, user authentication (if required), and robust network security measures.
* **Maintainability**:  
  Architect the system in a modular fashion so that components can be updated, replaced, or scaled independently. Clear documentation and adherence to coding best practices ensure that the system can be easily maintained and enhanced over time.

# 4. Architecture Design

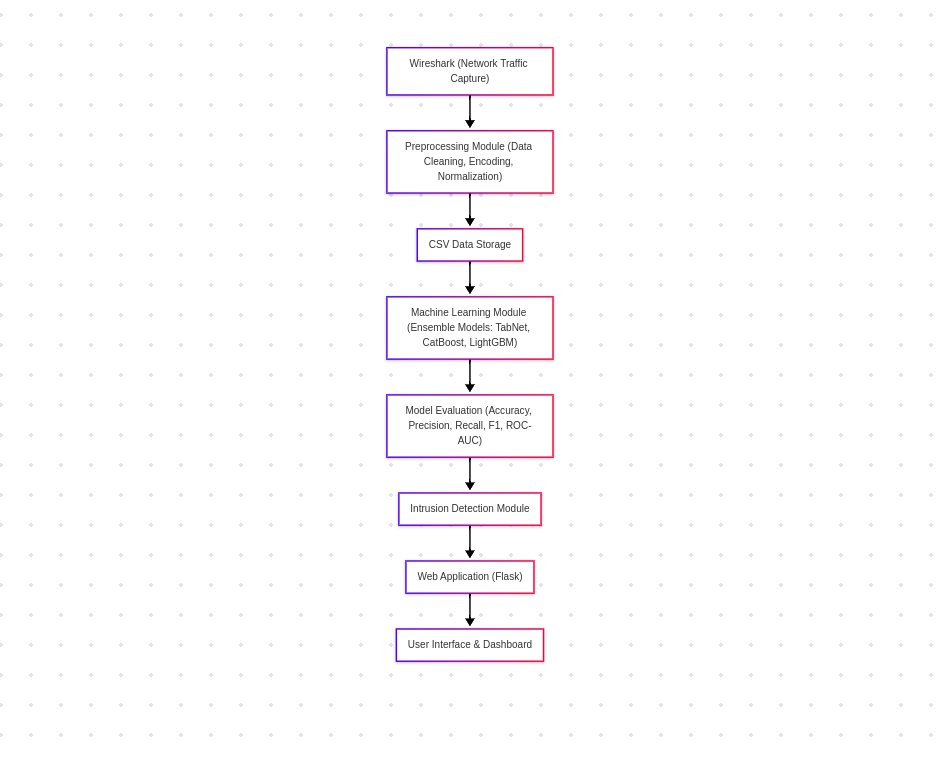
The architecture of the Real-Time Network Intrusion Detection System is organized into logical layers that separate concerns and enhance system maintainability. Below is an overview of the logical and physical architecture:

* **Logical Architecture:**
* **Data Capture Component:**
  + Wireshark acts as the primary data capture tool, intercepting live network traffic from various endpoints. It exports the captured traffic data in CSV format, ensuring that critical information such as protocol types, IP addresses, packet sizes, and time stamps is preserved. This component is designed to operate continuously, ensuring that even transient network anomalies are recorded for further analysis.
* **Data Preprocessing Component:**
  + A dedicated preprocessing module is tasked with cleaning, encoding, and normalizing the raw network data. This process involves removing irrelevant or redundant information, handling missing values, and transforming categorical variables into a numerical format suitable for model training. In addition, this module applies normalization techniques to numerical data (such as packet sizes and time intervals) to standardize the input.
* **Machine Learning Component:**
  + Ensemble models—such as TabNet, CatBoost, and LightGBM—are trained on the cleaned dataset. Each model is fine-tuned using historical network traffic data to identify patterns that differentiate normal behavior from potential intrusions. Model evaluation is carried out using key performance metrics (accuracy, precision, recall, F1-score, and ROC-AUC), and the best-performing model is selected for deployment in real-time intrusion detection scenarios.
* **User Interface Component:**
  + The front-end is powered by a Flask-based web application that serves as the primary interaction point for users. It provides intuitive interfaces for uploading CSV files, monitoring real-time processing status, and visualizing the detection results through interactive dashboards and charts. The UI is designed to be both user-friendly and responsive, ensuring a seamless experience across different devices and screen sizes.

**Physical Architecture:**

* **Server Environment:**
  + The system operates on a centralized server or a distributed cluster, depending on the deployment scale. This environment is responsible for handling data processing, model training, and executing real-time analysis tasks. In addition to running the backend services, the server environment also hosts the Flask web application, ensuring that the user interface remains accessible.
* **Client Environment:**
  + End users interact with the system via standard web browsers on desktops, laptops, tablets, or smartphones. The client environment is optimized to display analysis results, interactive visualizations, and real-time alerts. It relies on secure API calls to communicate with the backend, ensuring that sensitive data is transmitted safely.
* **Data Flow and Storage:**
  + The data flow begins with the capture of raw network traffic by Wireshark, which then passes the data to the preprocessing module. After cleaning and formatting, the data is stored in a dedicated CSV repository. This repository not only acts as a staging area for training the machine learning models but also preserves historical data for future reference and audits.
* **Communication and Integration:**
  + System components interact through well-defined APIs, ensuring that data flows seamlessly from the data capture stage to the final user interface. Secure protocols (such as HTTPS) and encryption mechanisms are employed to safeguard data in transit. This integration strategy not only streamlines the overall data flow but also enhances the system's robustness and reliability.

## 4.1. Architecture Diagram



# 5. User Interface Design

This section outlines the design of the user interface, detailing the layout, navigation, and user interactions.

**Layout and Structure**  
The user interface is designed with a modern, responsive layout to ensure a seamless experience across various devices. The main components include:

* **Landing Page:**
  + **Hero Section:** A visually engaging header with a concise project overview, call-to-action (CTA) buttons, and a brief introduction to the system’s purpose.
  + **Feature Highlights:** Sections that describe key functionalities, such as real-time analysis and intelligent detection, with accompanying icons or images.
* **Data Upload Page:**
  + A central, clearly visible file upload form that allows users to select and upload CSV files containing network traffic data.
  + Visual cues (e.g., drag-and-drop functionality, progress bars, and success/error notifications) to provide feedback during the upload process.
* **Results Dashboard:**
  + An interactive dashboard that displays analysis outcomes, including numerical metrics (accuracy, precision, recall) and visualizations (charts and graphs).
  + Options for users to drill down into specific details via modal pop-ups or separate detail pages.

**Navigation**  
The navigation structure is kept simple and intuitive:

* A persistent header includes links to key sections (Home, Upload Data, and Results), ensuring easy access from any part of the application.
* Breadcrumbs and clear call-to-action buttons further guide the user throughout the workflow.

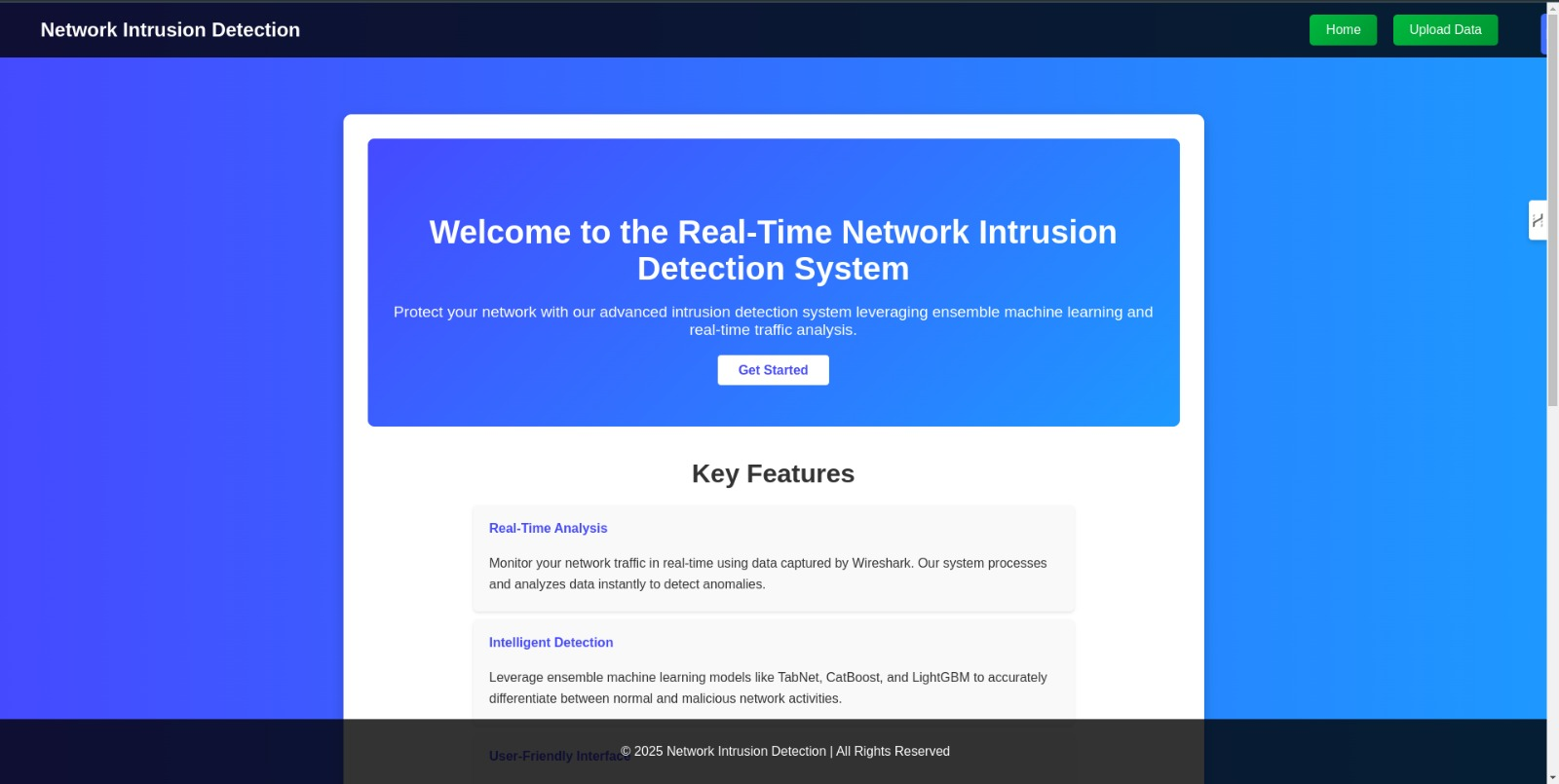
**User Interaction**  
User interactions are designed to be smooth and responsive:

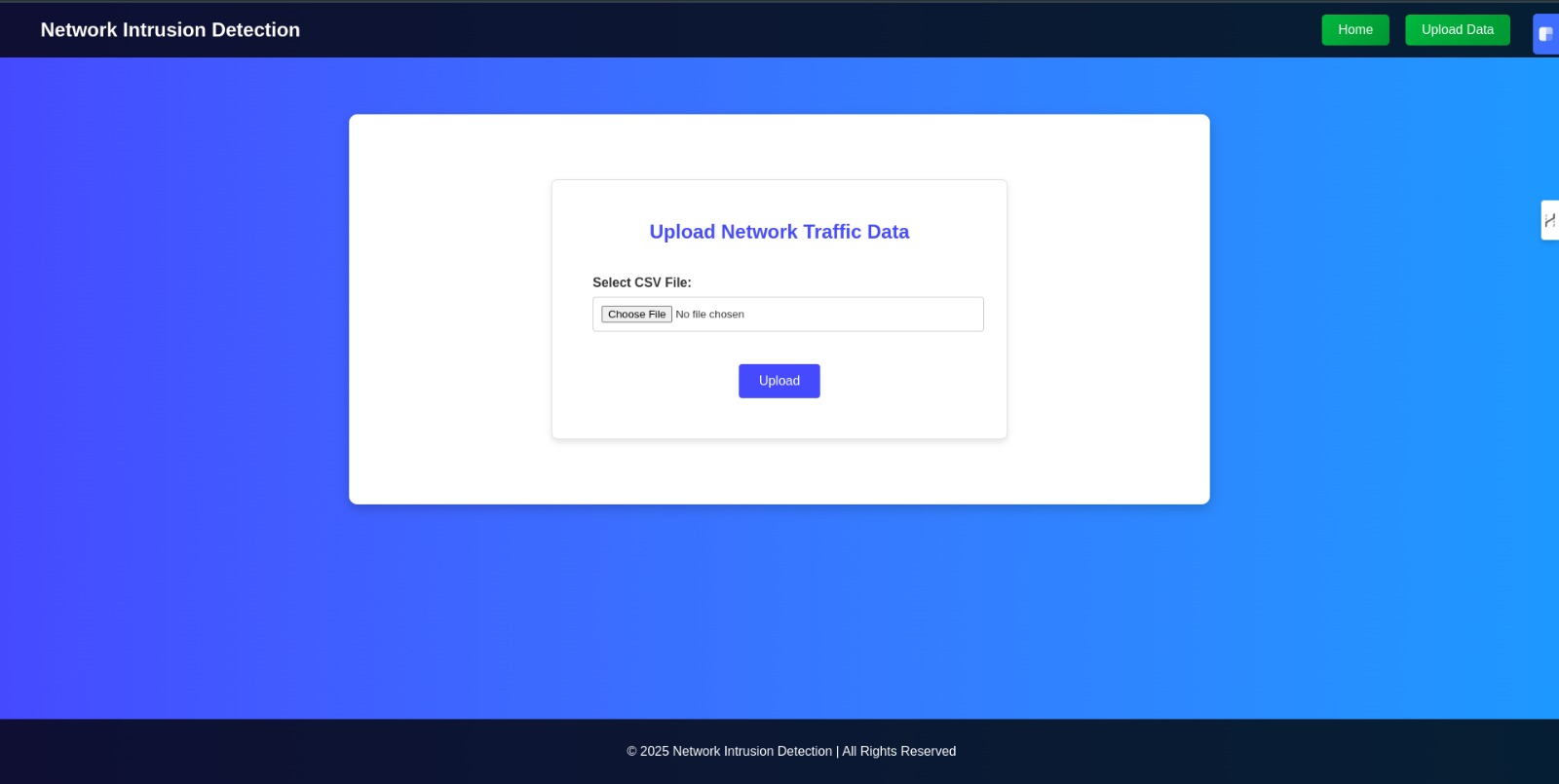
* **File Upload:** Instant feedback is provided upon file selection, with validation to ensure only CSV files are accepted.
* **Analysis Trigger:** Once data is uploaded, the system displays real-time processing indicators (such as loading spinners) until results are ready.
* **Results Visualization:** Interactive charts and tables allow users to view, filter, and analyze the detection outcomes.
* **Responsive Design:** The interface uses responsive design principles to adapt to various screen sizes, ensuring usability on desktops, tablets, and smartphones.

## 5.1. UI Layout

The UI is pretty minimalist with a blue background and basic flask routing. Screenshots of the UI and the pages are included in the section 5.2.

## 5.2. Screenshots





# 

# 6. Component Design

This section describes the design of individual system components and modules, explaining their responsibilities, interfaces, and interactions.

**1. Data Capture Module**

* **Responsibilities:**
  + Captures live network traffic using Wireshark.
  + Exports the captured data into CSV files containing essential network features (e.g., protocol types, IP addresses, packet sizes, timestamps).
* **Interfaces:**
  + Direct integration with Wireshark for data capture.
  + File system interface for storing exported CSV files.

**2. Data Preprocessing Module**

* **Responsibilities:**
  + Cleans and filters raw network data to remove noise and inconsistencies.
  + Encodes categorical features and normalizes numerical attributes to prepare the data for model training.
* **Interfaces:**
  + Reads CSV files from the data capture module.
  + Outputs cleaned and processed CSV files for the machine learning module.

**3. Machine Learning Module**

* **Sub-components:**
  + **Model Training:** Utilizes advanced ensemble learning algorithms (TabNet, CatBoost, LightGBM, and stacking ensembles) to train on the preprocessed data.
  + **Model Evaluation:** Computes performance metrics such as accuracy, precision, recall, F1-score, and ROC-AUC to select the best-performing model.
* **Responsibilities:**
  + Train and fine-tune the machine learning models using historical network traffic data.
  + Store the trained model in a repository for subsequent real-time analysis.
* **Interfaces:**
  + Accesses preprocessed CSV data.
  + Interacts with the Intrusion Detection Module for model deployment.

**4. Intrusion Detection Module**

* **Responsibilities:**
  + Deploys the trained machine learning model to perform real-time intrusion detection.
  + Classifies incoming network traffic data as normal or malicious.
* **Interfaces:**
  + Accepts live or uploaded network data from the Web Application Interface.
  + Returns detection results (including key network parameters and performance metrics) to be displayed to the user.

**5. Web Application Interface (Frontend)**

* **Responsibilities:**
  + Provides a user-friendly interface for file uploads, status monitoring, and result visualization.
  + Facilitates secure interaction between the user and the backend processing modules.
* **Interfaces:**
  + Communicates with backend modules via RESTful API endpoints.
  + Displays real-time feedback and interactive visualizations (charts, tables, graphs) based on analysis results.

**Interactions Between Components:**

* **Data Flow:**
  + Network data is captured, preprocessed, and stored as CSV files.
  + Preprocessed data is used to train and evaluate machine learning models.
  + The best model is deployed in the Intrusion Detection Module, which processes live data or uploaded files.
  + Results are then communicated back to the Web Application Interface for user consumption.
* **Modularity:**
  + Each component is designed as an independent module with clear input/output interfaces, ensuring maintainability and ease of future updates.

# 7. Data Flow Diagrams

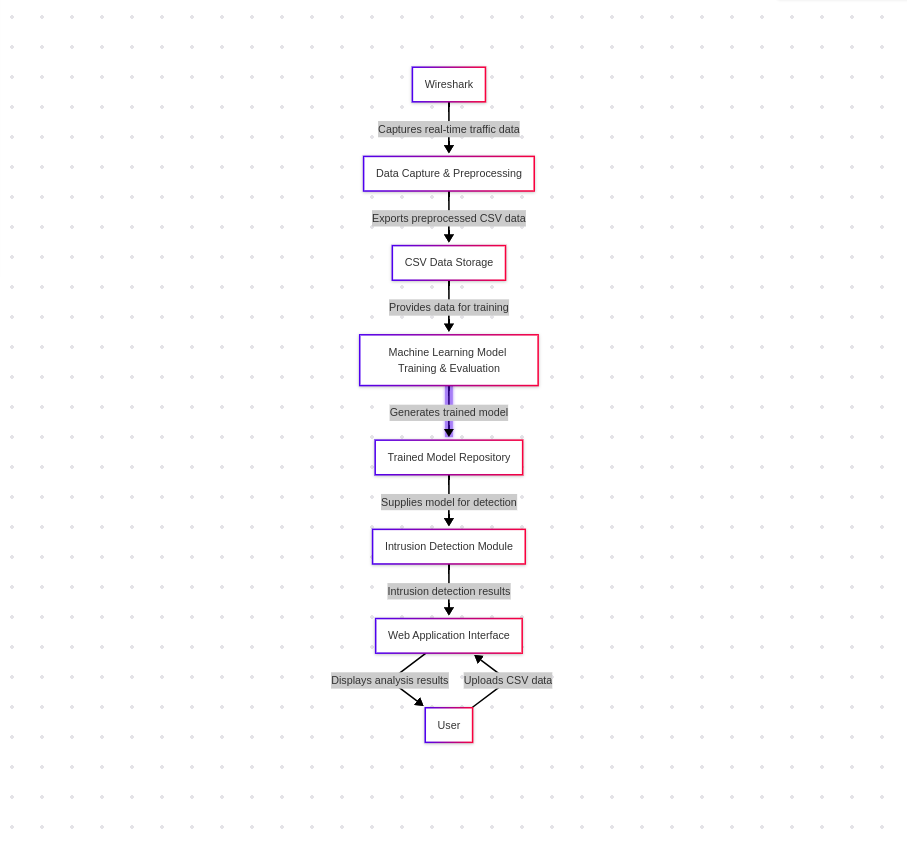
IThis section provides an overview of how data moves through the system, illustrating the key inputs, outputs, and transformation processes.

**Overview:**  
The Data Flow Diagram (DFD) depicts the system’s data flow across external entities, processes, and data stores. It serves as a visual representation of how raw network traffic is captured, processed, analyzed, and presented to the user.

**Key Data Flows and Processes:**

1. **Data Capture:**
   * **Source:** Wireshark captures live network traffic data.
   * **Process:** The raw data, including protocol types, IP addresses, and packet sizes, is exported as a CSV file.
   * **Data Store:** The CSV file is stored temporarily for preprocessing.
2. **Data Preprocessing:**
   * **Input:** The raw CSV file from Wireshark.
   * **Process:** Data is cleaned (removal of noise and inconsistencies), categorical variables are encoded, and numerical values are normalized.
   * **Output:** A preprocessed CSV file, ready for machine learning model training.
   * **Data Store:** The preprocessed data is stored in a dedicated repository.
3. **Machine Learning Model Training & Evaluation:**
   * **Input:** Preprocessed CSV data.
   * **Process:** Ensemble models (TabNet, CatBoost, LightGBM) are trained on the data. Performance metrics such as accuracy, precision, and recall are computed to evaluate model performance.
   * **Output:** The best-performing model is selected and stored.
   * **Data Store:** The trained model is stored in the model repository.
4. **Intrusion Detection:**
   * **Input:** Real-time or uploaded CSV data.
   * **Process:** The deployed model processes the incoming data, classifying each data packet as normal or malicious.
   * **Output:** Detection results, including performance metrics and key influencing network parameters.
5. **User Interface Interaction:**
   * **Input:** The user uploads CSV files or requests a status update.
   * **Process:** The Web Application Interface processes the user's input and triggers the intrusion detection process.
   * **Output:** Results are displayed on the user dashboard via interactive charts and tables.

**Diagrammatic Representation :**



# 8. Conclusion and Future Work

**Conclusion:**  
This design document has outlined the architecture and design decisions for the Real-Time Network Intrusion Detection System. Key decisions include:

* **Data Capture and Preprocessing:**  
  The system utilizes Wireshark for real-time network data capture, exporting the data as CSV files. A dedicated preprocessing module cleans, encodes, and normalizes the data, ensuring a robust input for subsequent analysis.
* **Machine Learning Integration:**  
  Advanced ensemble learning models—namely TabNet, CatBoost, and LightGBM—have been chosen to effectively classify network traffic. The training, evaluation, and deployment of these models have been designed to achieve high accuracy and real-time detection capabilities.
* **User Interface:**  
  A user-friendly web application built with Flask provides intuitive file upload functionality, real-time feedback during data processing, and interactive visualization of results. This ensures that even non-expert users can effectively utilize the system for network security analysis.
* **Modular Architecture:**  
  The system is organized into distinct layers (data capture, preprocessing, machine learning, and user interface), promoting scalability, maintainability, and ease of future enhancements.

**Future Work:**  
While the current design meets the project’s core objectives, several areas for improvement and enhancement are identified for future iterations:

* **Enhanced Real-Time Processing:**  
  Implementing a more robust streaming data pipeline (e.g., using Apache Kafka or similar technologies) to handle continuous network traffic data could further reduce latency and improve real-time responsiveness.
* **Model Optimization and Continuous Learning:**  
  Future work could explore techniques for continuous model retraining and adaptive learning, allowing the system to adjust to evolving network behaviors and emerging threats dynamically.
* **Expanded Data Sources and Integration:**  
  Integrating additional data sources beyond Wireshark, such as logs from firewalls and intrusion detection systems, would provide a more comprehensive view of network activity and potentially improve detection accuracy.
* **Scalability and Cloud Deployment:**  
  Migrating the system to a cloud-based infrastructure could enhance scalability, facilitate distributed processing, and enable real-time collaboration across multiple stakeholders.
* **Advanced Visualization and User Interaction:**  
  Future enhancements could include more sophisticated data visualizations, customizable dashboards, and advanced analytics features (e.g., trend analysis, anomaly detection insights) to further empower users in understanding and responding to network threats.
* **Security Hardening:**  
  As the system processes sensitive network data, ongoing work to strengthen security measures—including secure authentication, data encryption, and compliance with industry standards—will be crucial to protect against potential vulnerabilities.

In summary, the design decisions made for this system provide a solid foundation for real-time network intrusion detection. With the proposed future enhancements, the system can evolve to offer even greater accuracy, efficiency, and user satisfaction in combating emerging cybersecurity threats.