Real-Time Network Intrusion Detection Using Wireshark and Advanced Ensemble Learning Techniques

Software Requirements Specification

Version 1.0

Group Id:	**F24PROJECTA686A**
Supervisor Name:	**Laraib Sana**

Revision History

Date (dd/mm/yyyy)	Version	Description	Author
13/11/24	1.0	Initial version introducing the real-time network intrusion detection system using Wireshark and machine learning models.	BC210427835

Table of Contents

Contents

1. Scope of the Project

1.1 Overview

This project aims to address the growing need for real-time network intrusion detection by using machine learning techniques and integrating them into a user-friendly web application. By combining network traffic data captured via Wireshark with advanced ensemble learning models, the system will analyze traffic patterns to detect potential threats effectively and efficiently.

1.2 Project Domain

- **Domain**: Networking, Machine Learning, Cybersecurity
- **Category**: Research-driven project focusing on intrusion detection using state-of-the-art machine learning techniques and practical implementation.

1.3 Objectives

- Develop a robust Intrusion Detection System (IDS) capable of analyzing network traffic for malicious activity.
- Train and deploy ensemble learning models, including TabNet, CatBoost, and LightGBM, to classify network traffic as normal or malicious.
- Implement a web-based application to enable users to upload .pcap files, analyze traffic, and display detailed results, ensuring an intuitive user experience.

1.4 Scope Limitations

- The system will only analyze network traffic data captured and exported as .pcap files.
- Real-time analysis may depend on the size of the uploaded .pcap file and system resources.
- The focus is on accuracy and usability; scalability for enterprise-level traffic will not be prioritized.

2. Functional Requirements

2.1 File Upload via Web Application

Users must be able to upload captured .pcap files via the web application for analysis.

2.2 Parsing .pcap Files

The system must parse .pcap files into a structured dataset format (e.g., CSV) using tools like Scapy or PyShark.

2.3 Preprocessing Parsed Data

The system must preprocess parsed data by:

- Cleaning missing or invalid entries.
- Encoding categorical features (e.g., protocols).
- Normalizing numerical values (e.g., packet size, time intervals).

2.4 Dataset Labeling

The system must generate labeled datasets for ML model training, categorizing traffic as "normal" or "malicious."

2.5 Machine Learning Integration

The system must integrate three ensemble learning models (TabNet, CatBoost, LightGBM) for intrusion detection.

2.6 Model Evaluation

The system must evaluate ML models using metrics such as accuracy, precision, recall, F1-score, and ROC-AUC.

2.7 Web Application Interface

The system must provide a Django-based web application for users to upload .pcap files for analysis.

2.8 Analysis Results Display

The system must display analysis results, including:

- Traffic classification (normal/malicious).
- Intrusion likelihood.
- Affected IP addresses and protocols.
- Visualization of traffic patterns.

2.9 Error-Handling Mechanisms

The system must include error-handling mechanisms to detect and inform users about invalid or unsupported .pcap files.

3. Non-Functional Requirements

3.1 Performance

The system must process .pcap files up to 100 MB within 10 minutes under normal conditions, ensuring efficient analysis and result generation. This is critical for maintaining a balance between functionality and user expectations.

3.2 Scalability

The system must be capable of handling up to 5 concurrent .pcap file uploads without degradation in performance or result accuracy. This ensures the system can accommodate multiple users in academic or small-scale professional settings.

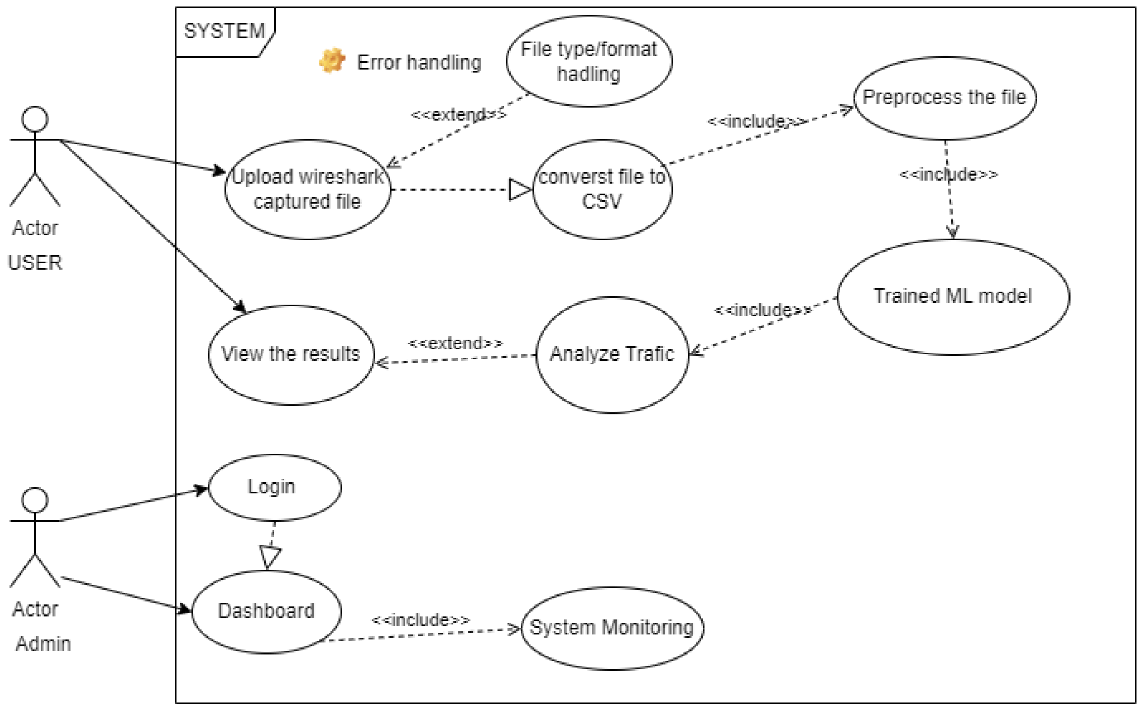
3.3 Reliability

The system must maintain an uptime of at least 95% to ensure availability. Additionally, it must handle unexpected failures gracefully, providing informative error messages and maintaining the integrity of uploaded data.

3.4 Usability

The web application must provide an intuitive user interface that enables non-technical users to easily upload .pcap files, analyze results, and interpret generated reports. This includes clearly labeled fields, accessible visualizations, and step-by-step guidance for file uploads.

4. Use Case Diagram



Actors:

1. ***User***
   - Uploads Wireshark captured file.
   - Views the results after traffic analysis.
2. ***Admin***
   - Logs into the dashboard.
   - Monitors the system.

Use Cases:

1. ***Upload Wireshark Captured File***
   - Handles file upload by the user.
   - Includes file type/format handling and error handling.
2. ***Convert File to CSV***
   - Converts raw Wireshark data into a usable CSV format.
   - Includes preprocessing.
3. ***Preprocess the File***
   - Cleans and organizes the CSV data for further analysis.
   - Used by both conversion and model training.
4. ***Trained ML Model***
   - Mandatory for analyzing traffic.
5. ***Analyze Traffic***
   - Analyzes uploaded network traffic using the trained ML model.

- Extends to result visualization.

6. ***View the Results***
    - User views analysis results (extends Analyze Traffic).
7. ***Dashboard***
    - Centralized hub for admin functionalities.
    - Includes system monitoring.
8. ***System Monitoring***
    - Admin monitors system performance and network activities.

05. Usage Scenarios

Use Case Title	**Upload .pcap File**
Use Case Id	1
Requirement Id	FTN:NIDS-FR-1
Description: In this use case, the user uploads a .pcap file to the system for intrusion detection analysis.	
Pre Conditions: 1. The user has a .pcap file ready for upload. 2. The web application is running and accessible. 3. The user has logged into the system.	
Task Sequence	**Exceptions**
1. The user logs into the web application	
2. The user navigates to the file upload section.	
3. The user selects and uploads the .pcap file.	
4. The system validates the file and starts processing.	
Post Conditions: • The .pcap file is successfully uploaded and validated. • The system initiates the parsing and preprocessing stages.	
• **Unresolved issues:** Handling extremely large .pcap files in the future.	
Authority: user	
Modification history: 1.0 **Author:** F24PROJECTA686A (BC210427835) **Description:** In this scenario, the user uploads a .pcap file to the web application The system validates the file and begins parsing it for further analysis.	

Use Case Title	**View Analysis Results**	
Use Case Id	2	
Requirement Id	FTN:NIDS-FR-2	
Description: In this use case, the user views the analysis results of a previously uploaded .pcap file.		
Pre Conditions: 1. The user has uploaded a .pcap file. 2. The system has completed the analysis. 3. The user is logged into the system.		
Task Sequence		**Exceptions**
1. The user logs into the web application		
2. The user navigates to the "Results" section		
3. The system displays a detailed report, including intrusion likelihood, affected IPs, and traffic patterns.		
Post Conditions: • The user has viewed the analysis report. • The system logs the user's activity for security purposes.		
• **Unresolved issues:** Future support for exporting reports in PDF format.		
Authority: user		
Modification history: 1.0 **Author:** F24PROJECTA686A (BC210427835) **Description:** The user accesses the analysis report generated after processing a .pcap file. The system displays detailed results to assist the user in identifying potential network intrusions.		

Use Case Title	**Train Machine Learning Model**	
Use Case Id	3	
Requirement Id	FTN:NIDS-FR-3	
Description: In this use case, the system trains machine learning models using labeled datasets for intrusion detection.		
Pre Conditions: 1. A labeled dataset is available for training. 2. The system has sufficient computational resources. 3. The administrator initiates the training process.		
Task Sequence		**Exceptions**
1. The administrator uploads the training dataset.		
2. The system validates the dataset and starts the training process.		
3. The system trains the models (TabNet, CatBoost, LightGBM) and evaluates their performance.		
4. The system trains the models (TabNet, CatBoost, LightGBM) and evaluates their performance.		
Post Conditions: • The models are trained and ready for integration. • The system logs the training session details.		
• **Unresolved issues:** Support for automated retraining based on new datasets.		
Authority: Administrator		
Modification history: 1.0 **Author:** F24PROJECTA686A (BC210427835) **Description:** In this scenario, the system processes labeled datasets to train machine learning models for traffic classification. The trained models are stored for future analysis tasks.		

6. Adopted Methodology

Different Software development life cycle models were explored and compared before selection. An **incremental model** was selected for this project. Comparison and reason of choice is mentioned below:

Waterfall Model : The Waterfall Model is a traditional, linear, and sequential SDLC approach. It requires each phase to be completed before moving to the next.

Why Waterfall May Not Be Ideal

1. Changes in requirements or unforeseen issues (e.g., ML model performance, data quality) can disrupt the process, requiring backtracking and increasing the risk of missed deadlines.
2. Testing occurs only after development is complete. If issues are found during the prototype or final deliverable phase, it could lead to significant rework and delays.
3. My project involves multiple interdependent components (data capture, ML, web app). Testing them only at the end could lead to integration problems.
4. With only two weeks for prototyping (Mar 4–17), the linear structure of Waterfall may leave little time for iterative refinement or adjustments.

Key Differences

Aspect	**Waterfall Model**	**Incremental Model**
Flexibility	Rigid structure, no room for changes during phases.	Highly flexible, supports iterative development and refinements.
Risk Management	Risks are addressed late during testing.	Early testing in increments reduces risks progressively.
Testing Approach	Performed only after development is complete.	Performed after each increment, ensuring functional components.
Prototyping	Prototyping is limited, with adjustments made late in the cycle.	Prototyping is integral, allowing continuous user feedback.

Why Spiral Model May Not Be Ideal

The Spiral Model, while effective for large, high-risk, and complex projects, was not selected for this project due to its heavy reliance on extensive risk analysis, detailed documentation, and iterative cycles, which can be time-consuming. As this is a solo university project with a tight 4-month deadline, the Spiral Model's thoroughness and complexity introduce unnecessary overhead. Additionally, the model's focus on risk management and iterative prototyping makes it more suitable for projects with higher levels of uncertainty or significant resource availability, which does not align with the requirements of this academic project.

Aspect	**Spiral Model**	**Incremental Model**
Risk Management	Focuses heavily on risk analysis at each cycle, making it suitable for high-risk projects.	Manages risks within each increment but without formal risk analysis, aligning better with our project.
Complexity	Designed for large, complex projects with high uncertainty.	Suited for moderately complex projects
Timeline	Time-intensive due to detailed risk assessment and documentation.	Delivers functional modules quickly, fitting within our 4-month deadline.
Suitability	Not suitable for my academic project due to its complexity.	It fits well with my project’s scope, allowing manageable progress and early deliverables.

Why did I choose Incremental Model over other model?

- The complexity of my project, involving ML models, preprocessing pipelines, and a web application, requires flexibility and early testing.
- Incremental delivery allows me to showcase progress at each milestone, which is critical for academic evaluations.
- Time-sensitive tasks like prototyping are better supported by the iterative nature of Incremental.

Incremental Model is the better choice for my university project. It accommodates complexity, ensures steady progress, and allows early testing and feedback to minimize risks. Waterfall may seem suitable due to its simplicity, but its rigidity could jeopardize the timely delivery of a functional and well-integrated system.

6.1 Incremental Development Model

The Incremental Model develops the system in smaller parts (increments), delivering functional subsets progressively. Each increment builds on the previous one, and testing occurs iteratively.

Increment 1: Wireshark Setup, Data Capture, and Dataset Preparation

- Install and configure Wireshark for network traffic capture.
- Collect real-world network traffic data.
- Capture network traffic data using Wireshark.
- Export captured traffic into CSV files with features like protocol types, IP addresses, packet sizes, and timestamps.
- Merge Wireshark-captured data with online datasets to form a comprehensive dataset.
- Clean and preprocess the combined dataset:
    - Handle missing or invalid entries.
    - Encode categorical data (e.g., protocol types).
    - Normalize numerical features (e.g., packet sizes and time intervals).
- Validate the correctness and quality of the preprocessed dataset.
- Ensure the dataset is structured and ready for ML model training.

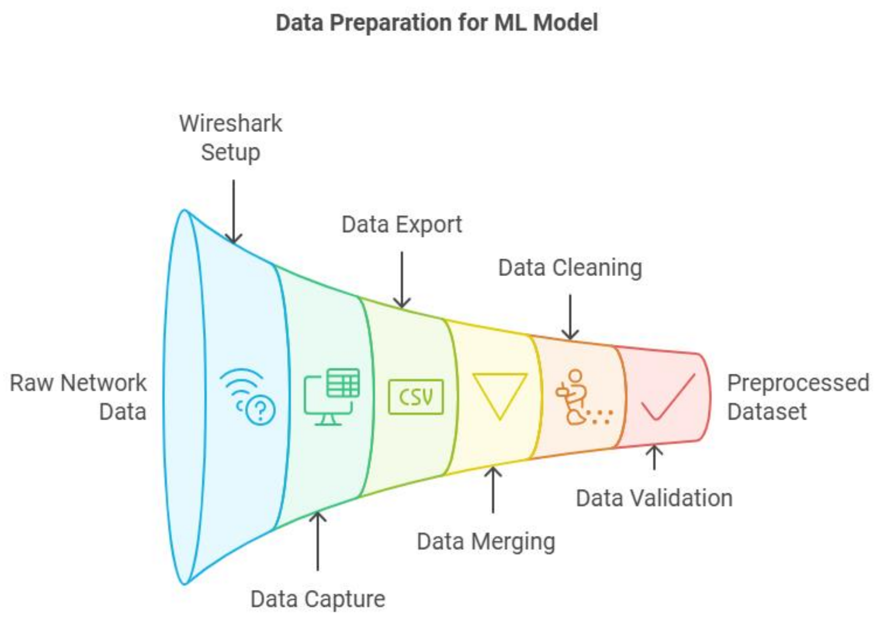


Figure 1 Traffic capture and building datasets

Increment 2: Machine Learning Model Development

- Split the dataset into training and testing sets.
- Train ensemble learning models (TabNet, CatBoost, LightGBM) for intrusion detection on the preprocessed dataset.
- Evaluate models using performance metrics:
  - Accuracy, Precision, Recall, F1-Score, and ROC-AUC.
- Fine-tune hyperparameters for optimal model performance.
- Test models on unseen data to ensure reliability.
- Compare the performance of TabNet, CatBoost, and LightGBM.

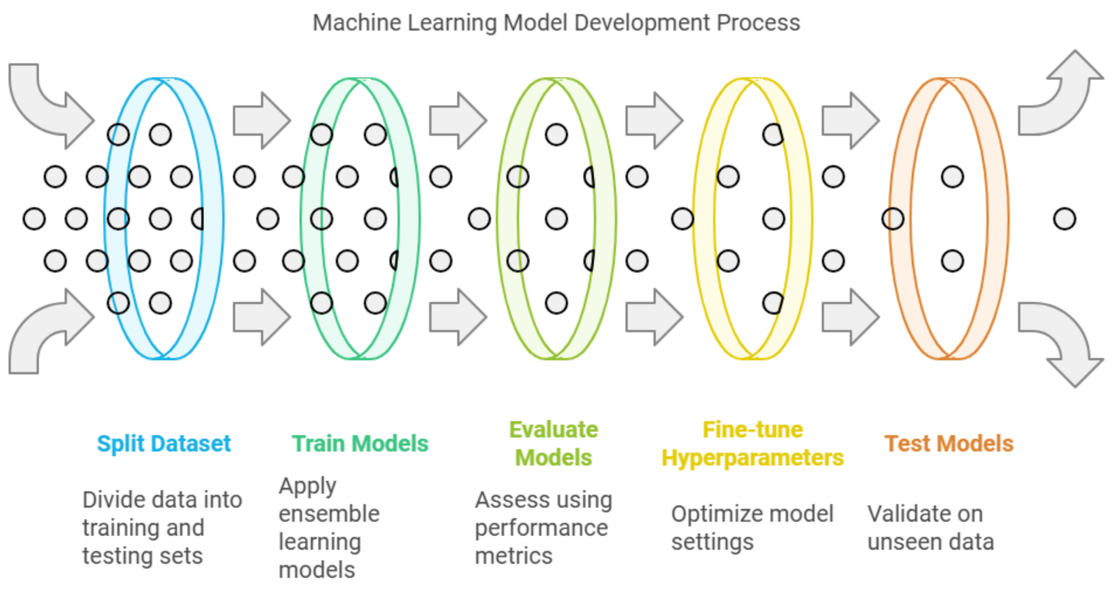

Machine Learning Model Development Process
Split Dataset
Divide data into training and testing sets
Train Models
Apply ensemble learning models
Evaluate Models
Assess using performance metrics
Fine-tune Hyperparameters
Optimize model settings
Test Models
Validate on unseen data

Figure 2 Developing tand Training ML Model

Increment 3: Web Application Development

- Develop a Django-based web application to integrate the ML model.
- Build the backend of the web app using Django:
  - Implement APIs for data upload and ML model inference.
- Create a frontend interface:
  - Allow users to upload network traffic data.
  - Display intrusion detection results based on ML predictions.
- Integrate the trained ML model into the backend for real-time analysis.
- Test the web app for usability and functionality.

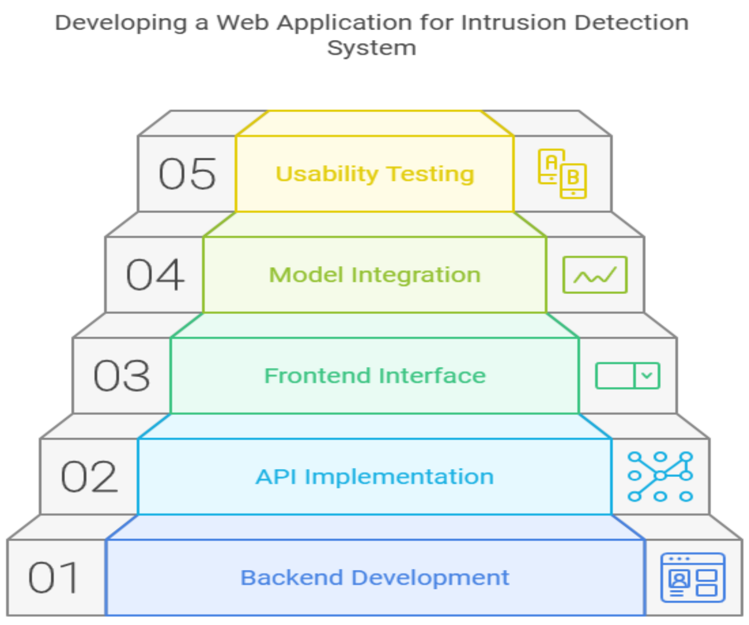


Figure 3 Web app development

Increment 4: Final Integration and Testing

- Integrate Web app, ML model for intrudion detection and pipline to preprocess user data
- Start testing as user upload wireshark captured data.
- Preprocessing pipeline for live data.
- ML models for real-time intrusion detection.
- Django web application for user interaction.
- Optimize the system for scalability, performance, and security.
- Conduct end-to-end testing with real-world traffic data.

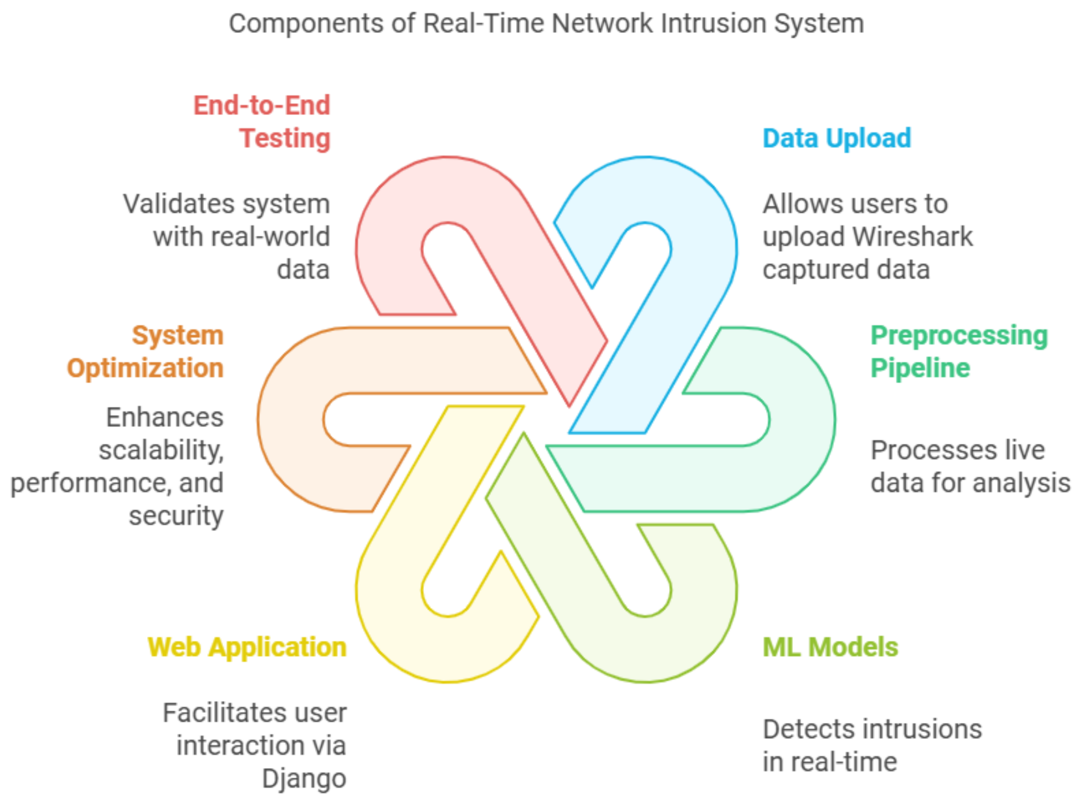

Components of Real-Time Network Intrusion System
End-to-End Testing
Validates system with real-world data
Data Upload
Allows users to upload Wireshark captured data
System Optimization
Enhances scalability, performance, and security
Preprocessing Pipeline
Processes live data for analysis
Web Application
Facilitates user interaction via Django
ML Models
Detects intrusions in real-time

Figure 4 Integration of all components

Increment 5: Final Deliverable submission .

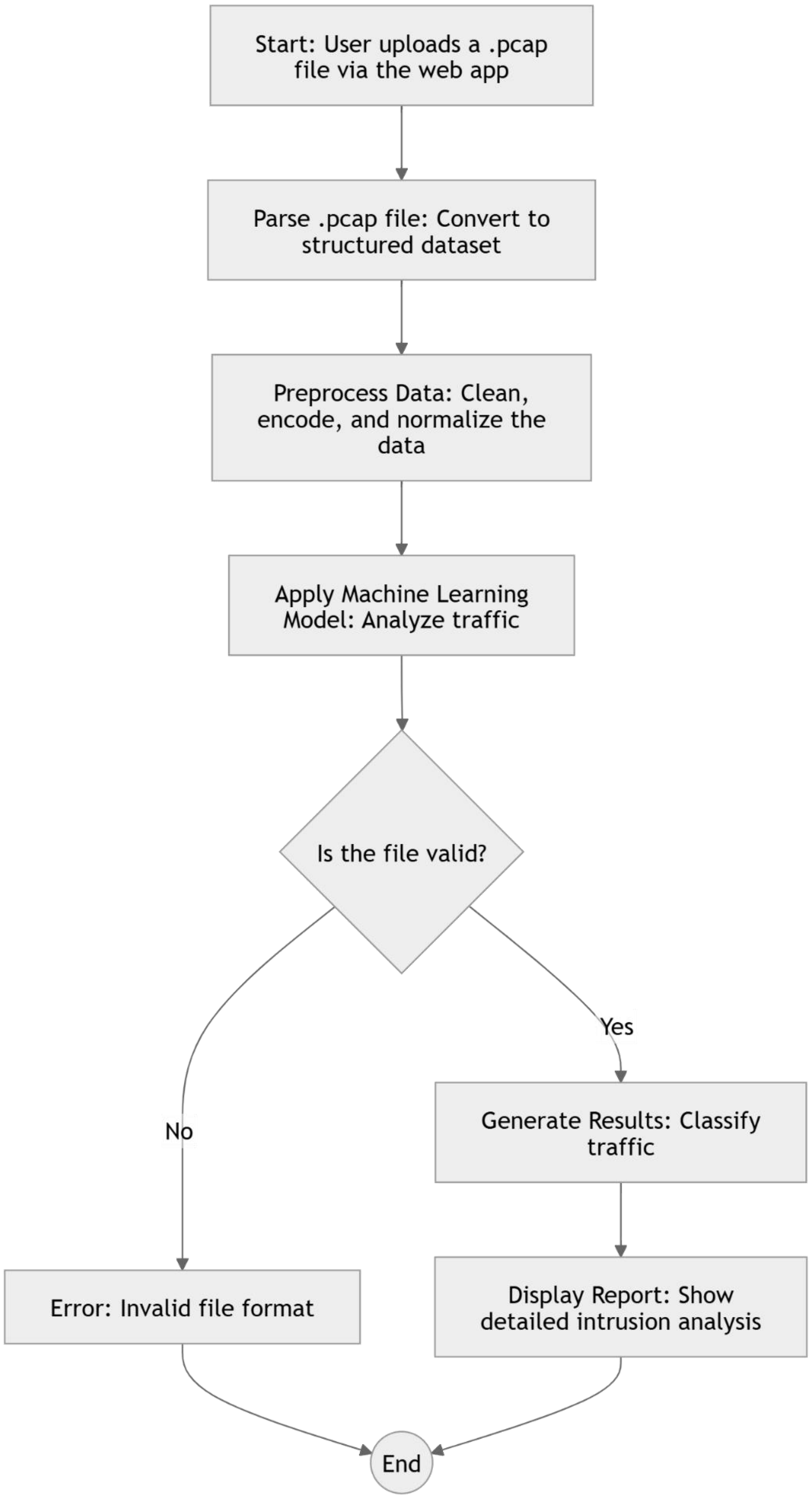
6.2 Machine Learning Techniques

- **Models**: TabNet, CatBoost, LightGBM
- **Process**: Data preparation, model training, performance evaluation, and inference integration.

6.3 Tools and Technologies

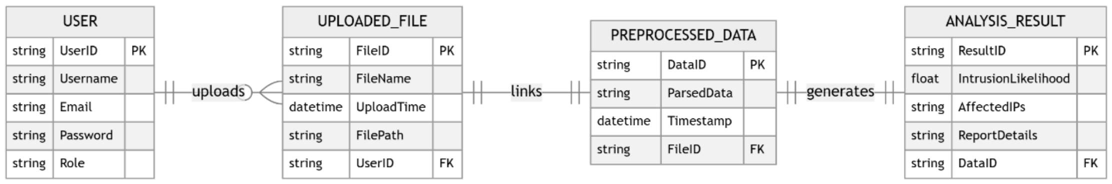
- **Traffic Capture**: Wireshark
- **Data Processing**: Python (Scapy, PyShark, Pandas)
- **Machine Learning**: Scikit-learn, TabNet, CatBoost, LightGBM
- **Web Development**: Django
- **Visualization**: Matplotlib, Seaborn
- **Version Control**: Git

7. Workflow Diagram:



Start: User uploads a .pcap file via the web app
Parse .pcap file: Convert to structured dataset
Preprocess Data: Clean, encode, and normalize the data
Apply Machine Learning Model: Analyze traffic
Is the file valid?
No
Yes
Generate Results: Classify traffic
Error: Invalid file format
Display Report: Show detailed intrusion analysis
End

8. ER Diagram for Webapp:

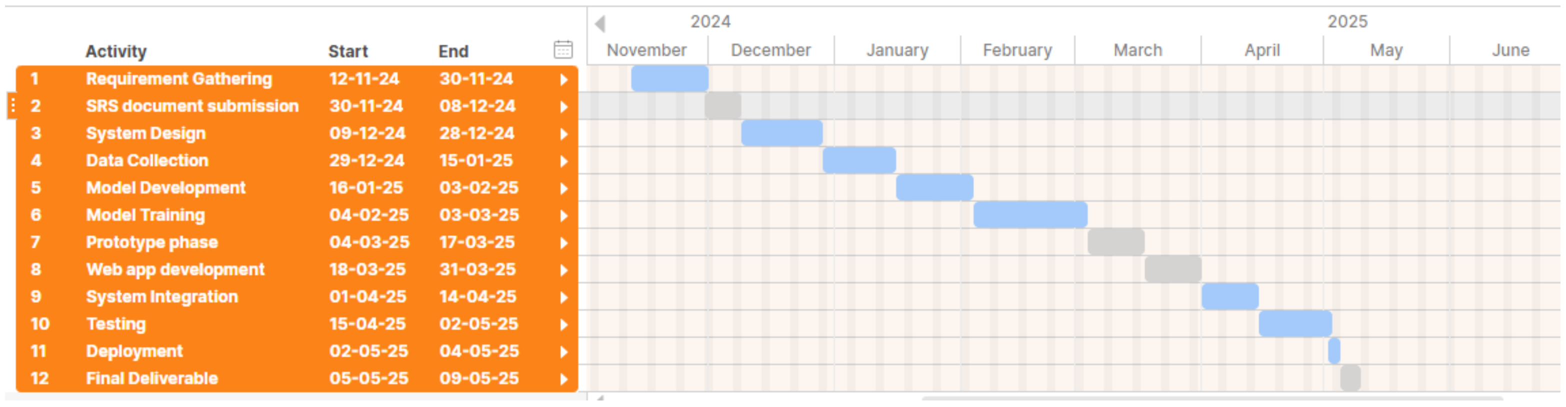


9. Work Plan

9.1 Phases and Deliverables

1. ***SRS and Requirements Gathering:***
   - Timeline: Nov 12 – Dec 8
2. ***Design and System Architecture:***
   - Timeline: Dec 9 – Mar 3
3. ***Prototype Development:***
   - Timeline: Mar 4 – Mar 17
4. ***Final Integration and Testing:***
   - Timeline: Mar 18 – May 9
   - Deliverable: Fully functional IDS ready for deployment.

9.2 Gantt charts



	Activity	Start	End
1	Requirement Gathering	12-11-24	30-11-24
2	SRS document submission	30-11-24	08-12-24
3	System Design	09-12-24	28-12-24
4	Data Collection	29-12-24	15-01-25
5	Model Development	16-01-25	03-02-25
6	Model Training	04-02-25	03-03-25
7	Prototype phase	04-03-25	17-03-25
8	Web app development	18-03-25	31-03-25
9	System Integration	01-04-25	14-04-25
10	Testing	15-04-25	02-05-25
11	Deployment	02-05-25	04-05-25
12	Final Deliverable	05-05-25	09-05-25