SentriNet

Acknowledgement

First and foremost, we would like to express our deepest gratitude to Allah for giving us the strength, patience, and knowledge to complete this graduation project.

We would like to sincerely thank our supervisor**, Dr. Sara Naiem**, for their continuous guidance, encouragement, and valuable feedback throughout every phase of the project.

Special appreciation goes to the faculty members and staff of the Computer &Artificial Intelligence at Helwan University, who provided us with the academic foundation and technical knowledge needed to accomplish this work.

We are also grateful to our team members for their collaboration, effort, and dedication in bringing this idea to life.

Finally, we would like to thank our families and friends for their unwavering support and motivation during this journey.

Project Link

Table of Contents

Chapter 1: Introduction

1.1 Overview

The rapid advancement of technology has led to a significant increase in internet usage, resulting in a more complex network environment with numerous applications and connected devices. This complexity has created opportunities for attackers to exploit vulnerabilities, leading to a rise in diverse network attacks. Intrusion Detection Systems (IDSs) have been developed to address these issues, utilizing misuse-based, anomaly-based, and hybrid techniques. The high volume and rate of network data generation pose challenges for IDSs in maintaining their efficacy and reliability.

Recent research has focused on understanding different types of IDSs, benchmark network datasets, dimensionality reduction techniques, and classification approaches based on machine learning and deep learning. A proposed framework for Network Intrusion Detection Systems (NIDS) has demonstrated high accuracy and detection rates, outperforming other approaches.

Cyber-attacks are becoming increasingly sophisticated, challenging the accuracy of intrusion detection. Failure to detect intrusions can compromise data confidentiality, integrity, and availability. IDS methods are broadly classified into Signature-based (SIDS) and Anomaly-based (AIDS) systems. Contemporary research reviews notable works, common evaluation datasets, evasion techniques used by attackers, and future research challenges to enhance computer security.

The increase in sensitive data transmission over the internet has heightened the need for robust IDSs to monitor and analyze network traffic, detecting and reporting malicious activities. Many reviews have focused on anomaly-based IDSs, particularly those using deep learning models, while signature and hybrid-based approaches have received less attention. A systematic review adhering to PRISMA principles provides a comprehensive overview of the state of IDS, highlighting the importance of anomaly, signature, and hybrid-based approaches in network security

The system is designed as a web platform that allows users to interact with different models (Network/Web) trained on real-world datasets using Machine Learning and Deep Learning algorithms. The goal is to improve threat detection rates while reducing false alarms and offering a user-friendly interface for technical and non-technical users.

1.2 Objectives

Over the past few decades, machine learning has significantly enhanced intrusion detection. However, there is a need for an up-to-date taxonomy and survey of recent work in this area.

Many studies have used datasets like KDD-Cup 99 or DARPA 1999 to validate IDS development, but there is no clear consensus on the most effective data mining techniques. Additionally, the time required to build IDS is often overlooked, despite being crucial for online IDS effectiveness.

This research aims to address these gaps by developing an intelligent NIDS using AI techniques, evaluating its performance, and comparing it with existing solutions.
It provides a comprehensive overview of existing IDSs, classifying them according to a new taxonomy. It surveys data-mining techniques for designing IDS, describing both signature-based and anomaly-based methods (SIDS and AIDS) and their evaluation techniques.
The paper also discusses the complexity of different AIDS methods, challenges for current IDSs, and makes recommendations.

Compared to previous surveys, this paper addresses IDS dataset problems, which are a major concern for the research community. It offers a structured and contemporary study on IDS techniques and datasets, highlighting challenges and making recommendations.

To fulfill these research and practical goals, the proposed system was designed with the following core objectives:

- Develop an AI-powered web-based IDS to monitor and detect intrusions in both web and network environments.

- Utilize real datasets (CICIDS2017 for web and UNSW-NB15 for network) to train machine learning and deep learning models.

- Present model evaluation metrics such as Accuracy, Precision, Recall, and F1-Score in an interactive dashboard.

- Allow users to interact with the system by selecting models and downloading performance reports.

- Enhance user experience with a supportive chatbot and a secure account management system.

1.3 Purpose

The purpose of this system is to provide a smart, reliable, and efficient platform that can detect malicious activities and threats using intelligent techniques. The system aims to help users understand the risks in their network or web environment and assess potential attacks through clear visualizations and documented analysis. It also helps non-security professionals to use machine learning models effectively without technical barriers.

1.4 Scope

- The system provides support for detecting network-based and web-based intrusions.

- Users can register, log in, and choose between network or web detection services.

- Users can view graphical results from trained models and download textual reports.

- The system includes a chatbot that provides information and guidance.

- Admins can manage user accounts and oversee the overall system.

- The application model (for mobile intrusion detection) is a planned feature and currently marked as "Coming Soon."

1.5 General Constraints

- The accuracy of the system is limited to the performance of the trained models and the quality of the datasets used.

- The system currently supports only two datasets: CICIDS2017 and UNSW-NB15.

- Model predictions are based on pre-trained results and do not include real-time data collection or live network scanning.

- Passwords are securely stored using hashing, but full end-to-end encryption is not yet implemented.

- Internet connectivity is required for all platform functionalities including model display and chatbot interaction.

Chapter 2: Planning and Analysis

2.1 Project Planning

2.1.1 Feasibility Study

- **Financial Feasibility**

  As a web-based platform, the system was implemented using open-source tools such as Python, Flask, React.js, and VS Code. The initial development was conducted on personal machines, minimizing infrastructure costs. No paid APIs or cloud services were used in the MVP (Minimum Viable Product), ensuring low operational costs.

- **Technical Feasibility**

  The system was developed using modern, lightweight, and scalable technologies. The backend was implemented using Python and Flask, and the frontend with React.js. Wireshark was used during the data analysis phase to understand network traffic. The team had the technical knowledge required to implement machine learning and deep learning models on structured datasets. All tools used were supported and compatible, which facilitated seamless integration.

- **Operational Feasibility**

  The application provides a straightforward user experience with features like model selection, performance visualization, chatbot support, and report download. These features ensure that users from non-technical backgrounds can still benefit from the system.

- **Social Feasibility**

  The project addresses a real-world security concern. Deploying intelligent IDS systems helps raise awareness and contributes positively to the cybersecurity domain. As such, the system has social value in promoting safer network practices and understanding AI-based security.

2.1.2 Estimated Cost

Item	Estimated Cost
Development Tools	$0 (Open-source)
Hosting / Deployment (Local)	$0 (Localhost)
Team Laptops / Machines	Already available
Miscellaneous	$0

Note: The entire development was conducted using personal resources. No external costs were incurred during this phase.

2.1.3 Gantt Chart

Phase	Start Date	End Date	Duration
Research and Dataset Study	15 Sep 2024	7 Oct 2024	3 Weeks
Backend Development	1 Nov 2024	20 Feb 2025	16 Weeks
Frontend + Integration	1 Nov 2024	20 Feb 2025	16 Weeks
Testing & Debugging	15 April 2025	5 May 2025	2.5 Weeks
Report + Presentation Prep	15 May 2025	20 May 2025	5 Days

2.2 Problem with Existing Systems

Current Network Intrusion Detection Systems (NIDS) face several limitations and challenges despite the advancements in AI and machine learning. Here are some key issues:

1. **High False Positive Rates**: One of the major challenges is the high rate of false positives. NIDS often generate alerts for benign activities, which can overwhelm security teams and lead to alert fatigue.
2. **Evolving Threats**: Cyber threats are constantly evolving, with attackers developing new techniques to bypass detection. NIDS need to be continuously updated to recognize new attack patterns, which can be resource-intensive.
3. **Data Quality and Volume**: The effectiveness of NIDS heavily relies on the quality and volume of data. Poor quality data or insufficient training data can lead to inaccurate detection. Additionally, handling large volumes of network traffic data in real-time is a significant challenge.

4. **Encryption and Obfuscation**: Many modern attacks use encryption and obfuscation techniques to hide malicious activities. This makes it difficult for NIDS to inspect and analyze network traffic effectively.
5. **Scalability**: As network sizes and traffic volumes grow, scaling NIDS to handle increased loads without compromising performance is a critical challenge. Ensuring that NIDS can operate efficiently in large-scale environments is essential.
6. **Resource Constraints**: Implementing and maintaining NIDS can be resource-intensive in terms of computational power, storage, and human expertise. Smaller organizations may struggle to deploy effective NIDS due to these constraints.
7. **Adversarial Attacks**: Attackers can use adversarial techniques to manipulate machine learning models used in NIDS, causing them to misclassify malicious activities as benign. This highlights the need for robust and resilient models.
8. **Integration with Other Security Tools**: Ensuring seamless integration of NIDS with other security tools and systems (e.g., firewalls, SIEMs) can be complex. Effective integration is crucial for comprehensive threat detection and response.

Addressing these challenges requires ongoing research, development, and collaboration within the cybersecurity community. By improving detection algorithms, enhancing data quality, and developing more resilient models, the effectiveness of NIDS can be significantly enhanced.

2.3 Need for the New System

To overcome the drawbacks in existing solutions, our system offers:

- A hybrid AI-based approach using both machine learning and deep learning.
- Interactive dashboards that visualize model results for better interpretation.
- Dataset modularity: CICIDS2017 for web and UNSW-NB15 for network.
- User-friendly interaction with chatbot assistance and report downloads.
- Potential to expand toward mobile-based application intrusion detection in future work.

2.4 Analysis of the new system

2.4.1 User Requirements

The user requirements were identified based on the expected usage of the system and interaction patterns. They are summarized as follows:

- The user must be able to register and log into the system securely.
- The user should be able to select between Web and Network analysis.
- The user can choose from different ML/AI algorithms for analysis.

- The system must display model results in an interactive dashboard.
- The user should be able to download a report of the selected model's results.
- A chatbot should be available to assist with questions and guidance.
- Admin users should be able to manage user accounts.

2.4.2System Requirements

- Hardware Requirements:
    - Processor: Intel Core i5 or higher
    - RAM: 8 GB minimum
    - Disk Space: 10 GB available
    - Network Adapter: Required for Wireshark analysis (optional for demo)

- Software Requirements:
    - Operating System: Windows 10 / Linux / macOS
    - Python 3.10+
    - FastAPI Framework
    - Django 5.x
    - React.js
    - Jupyter Notebook
    - VS Code
    - Wireshark

2.4.3 Domain Requirements

The system operates within the cybersecurity domain, specifically focusing on network and web-based intrusion detection. The domain-specific requirements include:

- Ability to process structured network/web traffic datasets (e.g., CICIDS2017, UNSW-NB15).
- Integration of AI/ML models capable of detecting anomalies and attack patterns.
- Display of classification metrics (Accuracy, Precision, Recall, F1-Score).

- Secure handling of user login data with encryption and hashing.

2.4.4 Functional Requirements

- FR1: The system shall allow users to register and log in.
- FR2: The system shall let users select a dataset type (Web or Network).
- FR3: The system shall allow users to choose a model to run.
- FR4: The system shall execute model code and return evaluation metrics.
- FR5: The system shall visualize the results using charts.
- FR6: The system shall allow users to download results as a report.
- FR7: The chatbot shall provide system help and basic info.
- FR8: Admins shall be able to view and manage registered users.

2.4.5 Non-Functional Requirements

- NFR1: The system must be responsive and work on common screen sizes.
- NFR2: The response time for generating results should not exceed 10 seconds.
- NFR3: The system shall securely hash all user passwords.
- NFR4: APIs must be protected from unauthorized access.
- NFR5: The system must have high availability during normal operation.
- NFR6: The UI must be intuitive and easy to use for non-technical users.

2.5 Risks and Risk Management

Risk	Probability	Impact	Mitigation Strategy
Incomplete or poor-quality datasets	Medium	High	Use benchmark datasets like CICIDS2017 and UNSW-NB15
Model underperformance	Medium	Medium	Tune hyperparameters and use feature selection
User misunderstanding of	Low	Medium	Provide tooltips, chatbot, and

metrics			explanations
Security breach (user credentials)	Low	High	Hash passwords, apply best practices in auth
Timeline delay due to integration issues	Medium	Medium	Parallel development and task division

2.6 Advantages of the New System

- Uses real datasets and AI techniques for high-accuracy intrusion detection.
- Interactive dashboards enhance user understanding of results.
- Flexible model selection allows customization per user need.
- Easy-to-use interface suitable for both technical and non-technical users.
- Chatbot assistant reduces confusion and improves user experience.
- Can be extended to support mobile apps and real-time detection in the future.
- Free and open-source implementation with low operational costs.

Chapter 3: System Design

System design is a crucial phase that translates user requirements into a blueprint for implementation. This chapter outlines the logical and structural design of the Intelligent Network Intrusion Detection System (IDS) including diagrams for entities, use cases, sequences, and system activities.

3.1 Entity Relationship Diagram (ERD)

The system consists of several core entities that interact with each other to manage users, intrusion detection models, results, and reports.

Entities:

- **User**: Stores registered users' information (username, email, password hash, role).

- **Model**: Represents the machine learning or deep learning model used in training/testing.

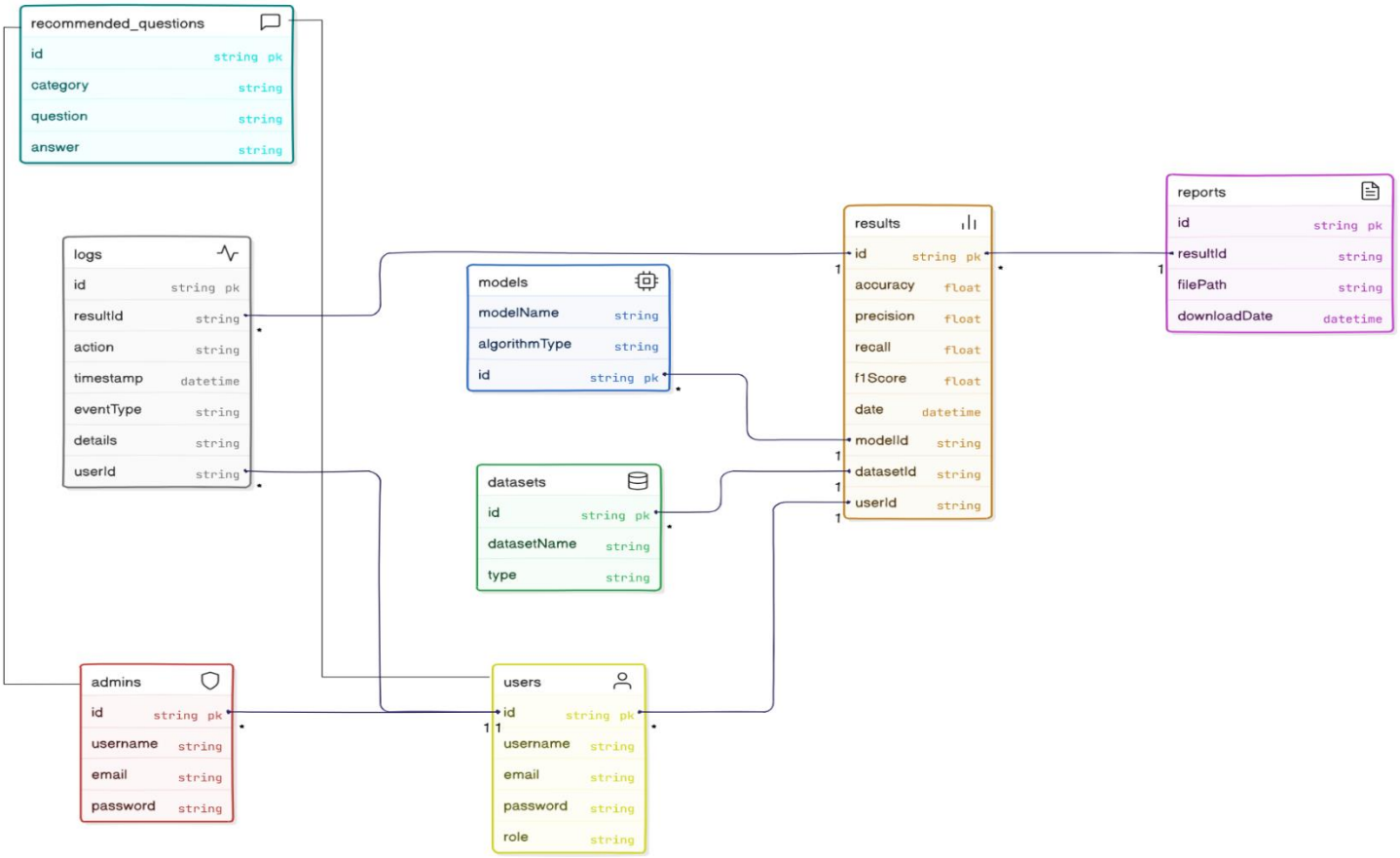
- **Dataset**: Represents the dataset used for training (CICIDS2017, UNSW-NB15).

- **Result**: Stores output metrics from the model (accuracy, precision, recall, F1-score).

- **Report**: Downloadable text file generated from the result.

- **Admin**: Has access to system control and user management.

ERD Description (Textual Layout):



Note: Visual ERD diagram can be provided separately upon request using draw.io or DBML.

3.2 Use Case Diagram

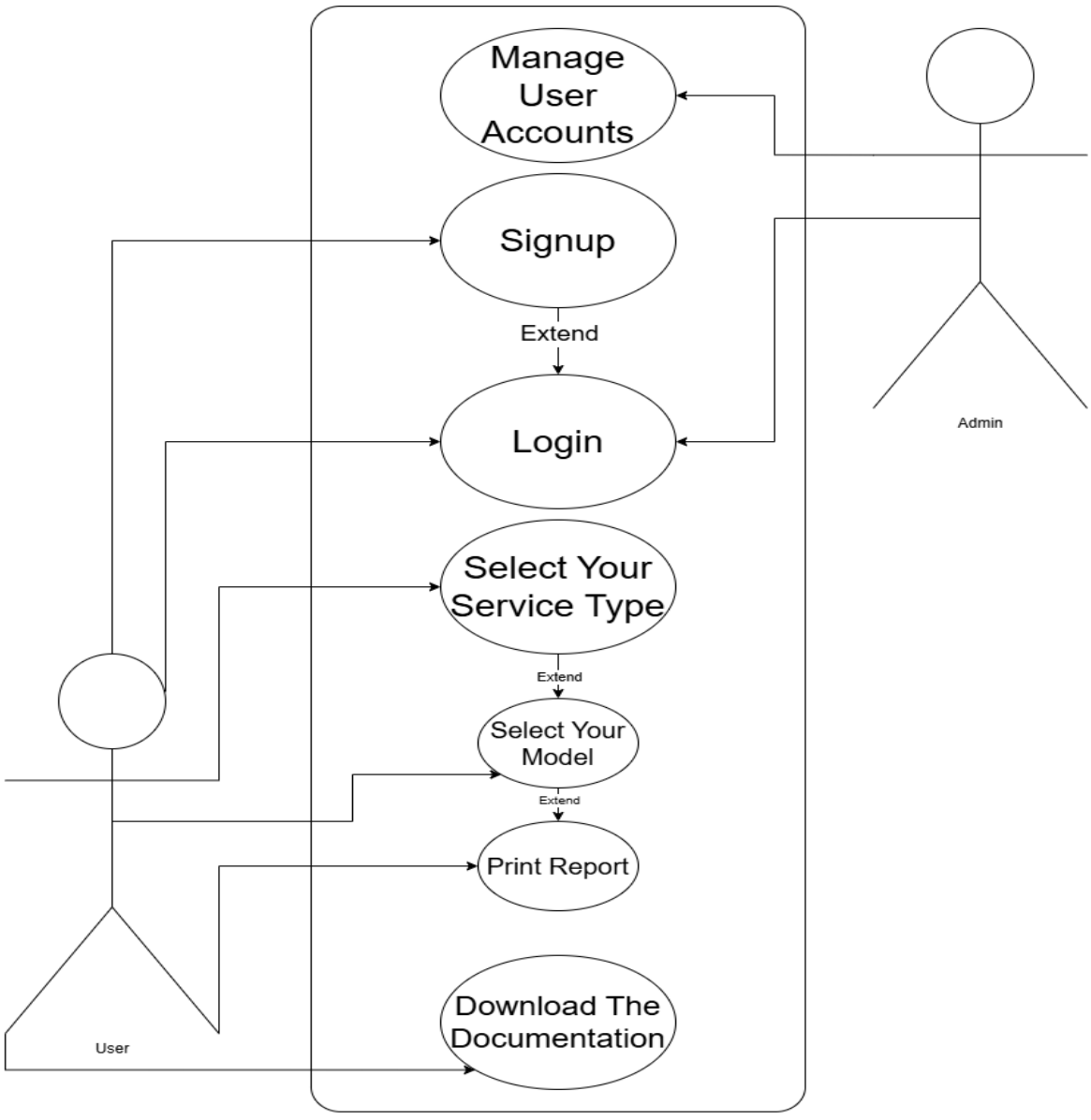
The system supports three types of users: Visitors, Registered Users, and Admins. The use case diagram defines how each actor interacts with the system.

Actors:

- **Visitor**: Can view basic information, browse datasets, and register.

- **User**: Can log in, select models, view charts, download reports, and interact with the chatbot.

- **Admin**: Can manage users, monitor model results, and oversee the platform.

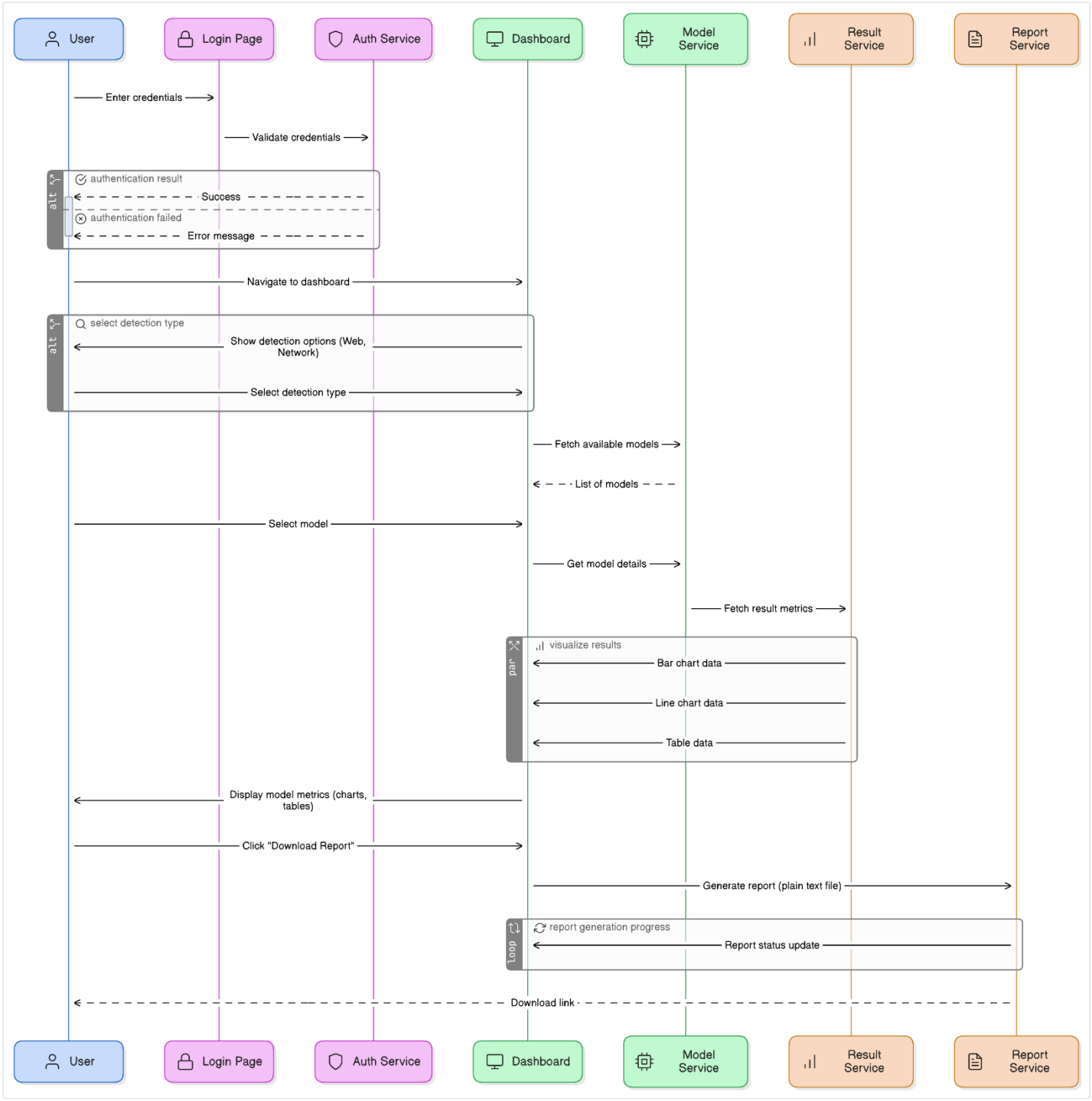


Use Case Descriptions Table

Use Case Name	Actors	Description	Preconditions	Main Flow	Postconditions
Register Account	Visitor	A visitor signs up by entering personal information.	Visitor is not registered.	Fill form → Validate input → Hash password → Save to DB → Confirm registration	New user account is created.
Login	User, Admin	Authenticates user and redirects to dashboard.	User/Admin already registered.	Enter email/password → Validate → Redirect to dashboard	User/Admin is logged in.
Select Service Type	User	User selects Web or Network analysis service.	User is logged in.	Click on Web/Network → Load corresponding dashboard	Service dashboard is shown.
Select Model	User	User selects which algorithm/model to evaluate.	Service type is selected.	Display model list → User selects model → System loads result	Model metrics are displayed.
View Results	User	View performance metrics and graphical charts.	Model is selected.	Load results → Show charts (Accuracy, Precision, etc.) → Tooltip on hover	User views model evaluation.
Download Report	User	User downloads result as text report.	Results displayed.	Click download → System generates text file → Start download	Report downloaded.
Manage Users	Admin	Admin can view/edit/delete user accounts.	Admin is logged in.	Access user management → Select action (view/edit/delete)	User data updated or removed.

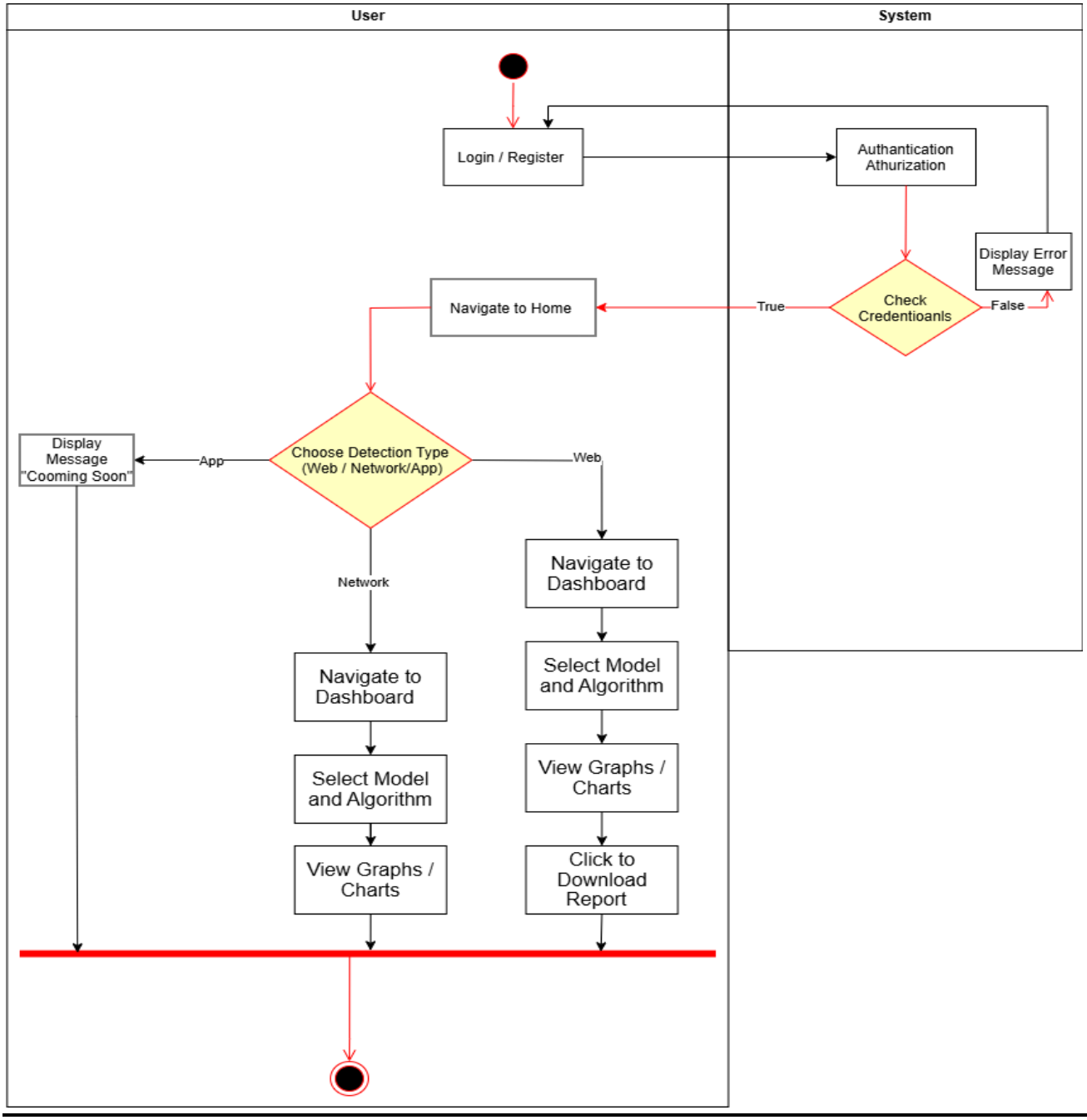
3.3 Sequence Diagram

Use Case Example: "View Model Result and Download Report"



3.4 Activity Diagram

Activity: "Model Interaction and Evaluation"



Chapter 4: System Implementation

This chapter presents the technical implementation of the Intelligent Network Intrusion Detection System (IDS), describing the architecture, technologies used, and how the components interact to fulfill the project requirements.

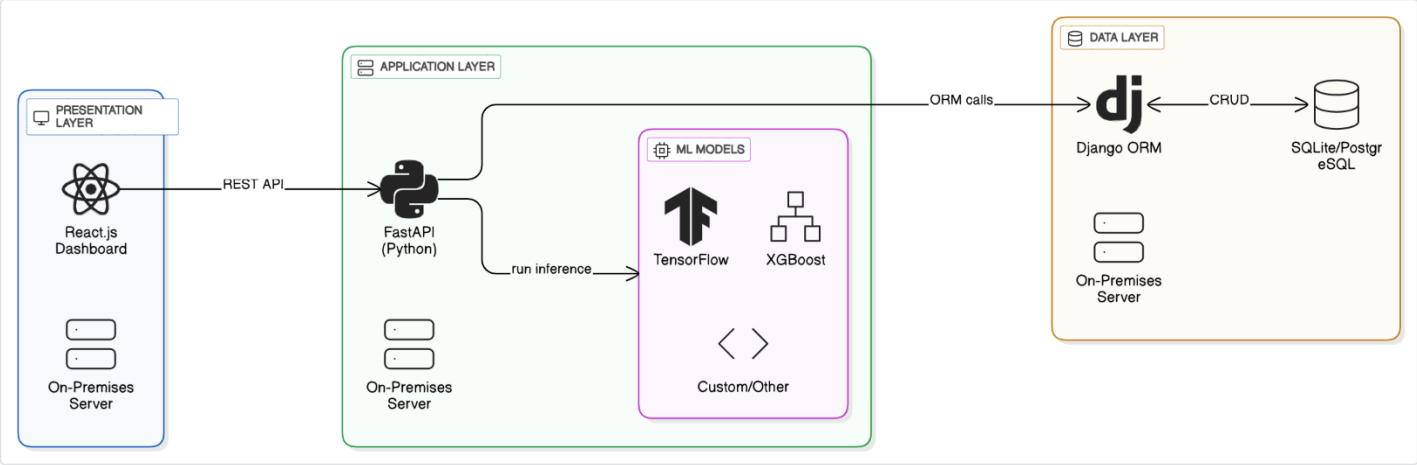
4.1 Software Architecture

The system is organized into three major layers:

- **Frontend Layer**: Built using **Fast** (likely referring to a light frontend framework or possibly FastAPI for UI endpoints), integrated with **ReChart.js** for dynamic result visualization.

- **Backend Layer**: Developed using **Python** and **FastAPI** , handling the logic for intrusion detection, API endpoints, and communication with the ML models.

- **Database Layer**: Managed using **Django ORM**, responsible for storing user information, system data, and logs.



The project is modularized into three main directories:

- `/BackendFrontend `: Contains all server-side code including ML model runners and APIs and the user interface, including authentication pages and dashboards.

- `/ DataPrepation`: Contains all server-side code that analyzing and preparing datasets.

- `/Database`: Contains Django configurations and models for managing the DB.

4.2 Backend Implementation

The backend is built using **FastAPI**, a modern web framework for building APIs with Python. The system processes requests from the frontend and runs intrusion detection models based on user selections.

ML Models Used:

- **Machine Learning Algorithms**: Logistic Regression, XGBoost ,Random Forest

- **Deep Learning Algorithms**: MLP (Multi-Layer Perceptron), CNN (Convolutional Neural Network)

All model implementations are written in `.py ` Python files. Rather than pre-saving models, the system dynamically executes the corresponding notebook when a model is selected, generating the evaluation metrics on the fly.

Workflow:

1. User selects detection type and algorithm.

2. Backend runs the related `.py` files.

3. Results (Accuracy, Precision, Recall, F1-Score) are extracted.

4. Backend sends these results to the frontend via API.

4.3 Frontend Implementation

The frontend was implemented using **Fast** (possibly FastAPI templating or a lightweight rendering method). It includes the following pages:

- **Home Page**: Overview and dataset selection.

- **Login/Register Page:** Secured user access.

- **Dashboard**: Displays result metrics.

- **Chatbot Interface**: Provides basic assistance and info about the models.

Visualization:

- **ReChart.js** was used to plot real-time performance metrics.

- The user can hover over bars/lines to get precise values.

Interaction with the backend is done through **Axios** or **Fetch APIs** that handle:

- Model execution requests

- Result retrieval

- Report downloads

4.4 Integration and Communication

The backend and frontend are tightly integrated using API calls. Each frontend interaction sends a request to FastAPI, which processes the request and returns a JSON response. This ensures modularity and scalability.

4.5 Tools and Libraries Used

Tool/Library	Purpose
Python	Backend programming, ML implementation
FastAPI	API development
Django ORM	Database management
ReChart.js	Graph/chart rendering in frontend
Wireshark	Network packet analysis during dataset prep
VS Code	Main development environment
Jupyter Notebook	ML model coding and execution

Python Packages:

- `asgiref`, `blinker`, `click`, `colorama`, `Django`, `itsdangerous`, `Jinja2`, `MarkupSafe`, `sqlparse`, `tzdata`, `Werkzeug`, ` email-validator`, `vite`

These were used to support web serving, templating, security, and timezone management.

4.6 Dataset Preparation

A specialized preprocessing pipeline was developed to ensure model readiness and performance. This step was key to enhancing accuracy and consistency across both datasets.

Steps:

- **Cleaning:** Null handling, duplicate removal.

- **Encoding:** `OneHotEncoder` for categorical variables.

- **Scaling:** `StandardScaler` to normalize features.

- **Feature Selection:** `SelectKBest` using `chi2` scoring.

- **Splitting:** `train_test_split` for training/testing separation.

Libraries Used:

`pandas`, `numpy`, `matplotlib`, `seaborn`,

`scikit-learn`, `plotly.graph_objects`, `os`, `time`, `warnings

4.7 Security Measures

The system implements basic security techniques to protect user credentials and data integrity. Passwords are hashed before storage using secure algorithms, and input validation is performed to prevent injection attacks. While full end-to-end encryption is not in place, the system architecture is built to accommodate future security upgrades including token-based authentication (e.g., JWT).

4.8 Known Limitations

- The system currently does not support real-time traffic analysis or live packet capturing.

- The ML models are run on static datasets and re-evaluated at runtime, which increases latency.

- User authentication is basic and could be enhanced with more advanced methods (e.g., multi-factor auth).

Chapter Summary

This chapter explained the technical details of how the Intelligent IDS system was developed. The backend was built with FastAPI and Python, while Django managed the database layer. ML models were implemented in Jupyter and triggered dynamically. The frontend, supported by ReChart.js, provided a user-friendly interface for interaction and visualization. The integration between components was seamless through REST APIs, ensuring the reliability of the system.

Chapter 5: System Testing

System testing ensures that the implemented system functions correctly according to its design specifications. This chapter outlines the different types of testing applied to our IDS platform and presents sample test cases and results.

5.1 Testing Objectives

- Validate the accuracy of model predictions.

- Ensure smooth communication between frontend and backend.

- Confirm correct behavior of APIs and user interactions.

- Verify security mechanisms during login and data handling.

- Detect and fix bugs before deployment.

5.2 Types of Testing Performed

Testing Type	Description
Unit Testing	Focused on isolated functions like metric calculations and dataset loaders.
Integration Testing	Verified the link between frontend, backend, and database components.
System Testing	End-to-end validation of user journeys and system workflows.
Functional Testing	Tested core features like login, model selection, and result visualization.
Security Testing	Ensured password hashing and restricted access.
User Acceptance	Final review to ensure the system meets project goals and user expectations.

5.3 Sample Test Cases

Test Case	Input	Expected Output	Status
Login with valid credentials	Registered email & password	Redirect to dashboard	Passed
Login with invalid credentials	Wrong email or password	Error message shown	Passed
Model selection (Web-CNN)	Web service + CNN algorithm	Show result metrics + chart	Passed
Download report	Click on “Download”	Generate and download text report	Passed
Unauthorized access to admin panel	Direct access URL without login	Redirect to login or access denied	Passed
Dataset not found	Invalid dataset name	Error message / handled gracefully	Passed
Password storage	New user registration	Password saved as hash, not plaintext	Passed

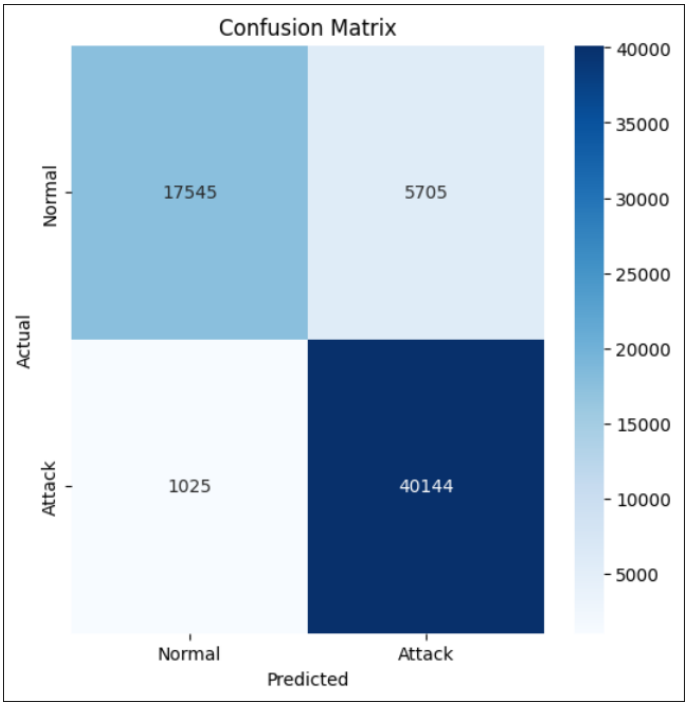
5.4 Bug Reports & Fixes

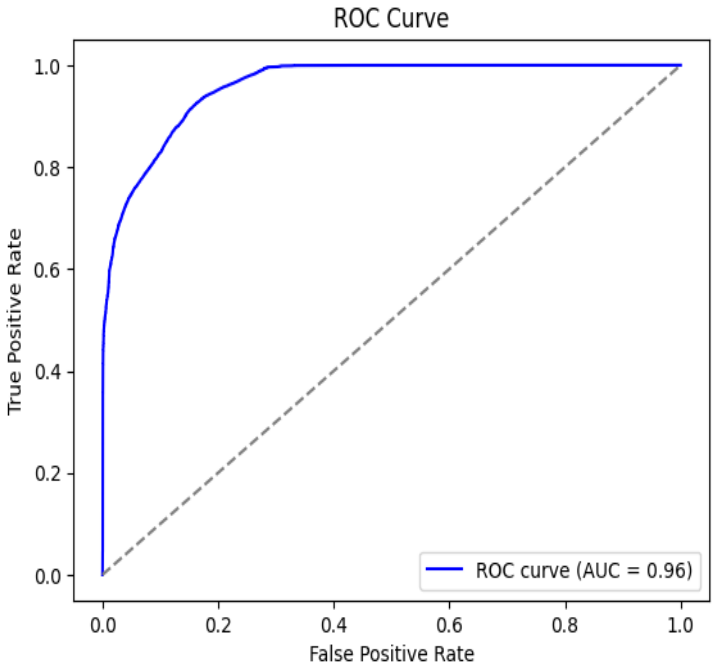
Bug ID	Description	Detected In	Resolution
001	Model metrics not updating dynamically	Backend API function	Fixed notebook trigger sync
002	Hover tooltip not showing exact values	ReChart.js config error	Updated chart options
003	Login form not validating empty fields	Frontend JS validation	Added required field checks

5.5 Performance Testing (ML Models)

1) NETWORK MODELS:-

Sample results for Logistic Regression:

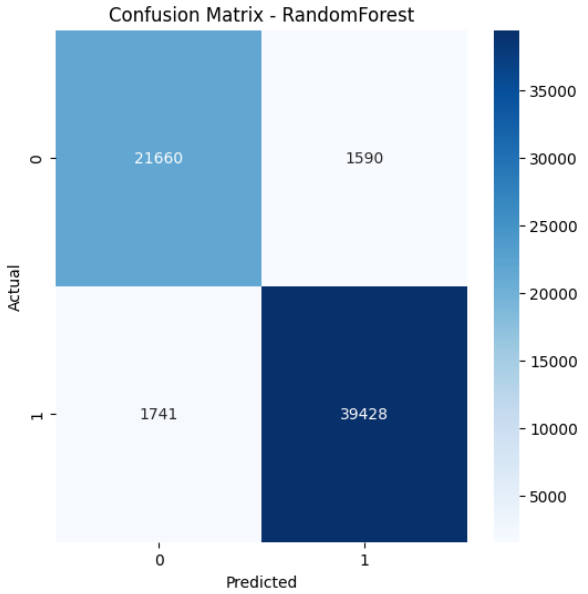


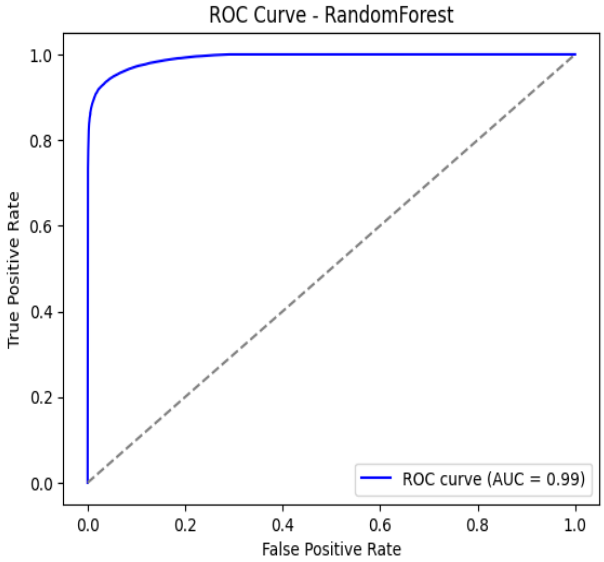


Classification Report:

	precision	recall	f1-score	support
0	0.94	0.75	0.84	23250
1	0.88	0.98	0.92	41169
accuracy			0.90	64419
macro avg	0.91	0.86	0.88	64419
weighted avg	0.90	0.90	0.89	64419

Sample results for RandomForest:

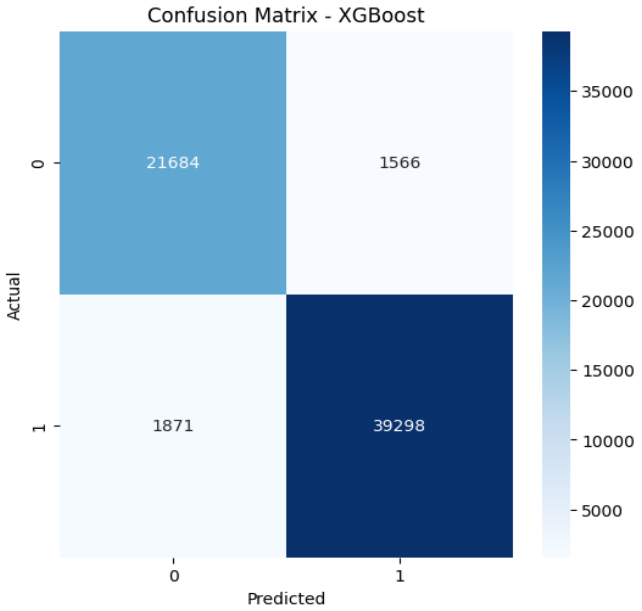


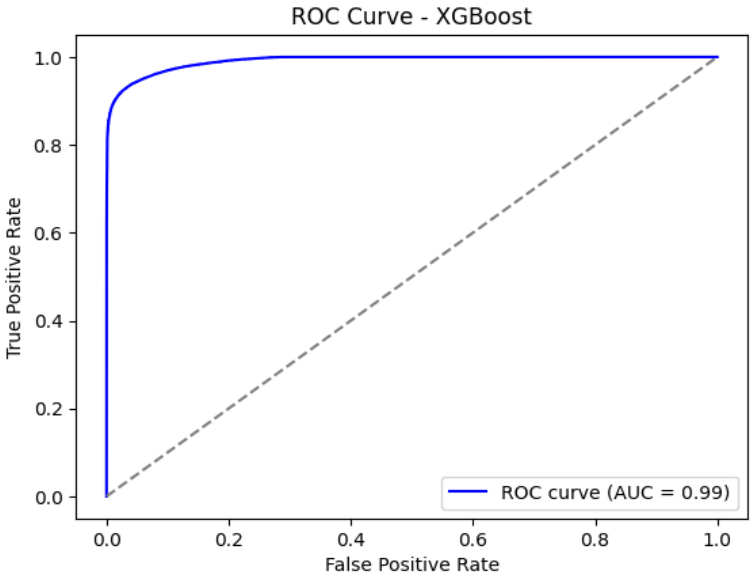


RandomForest Model Performance:

- **Accuracy:** 94.83%
- **Recall:** 94.83%
- **Precision:** 94.84%
- **F1-Score:** 94.83%
- **Training Time:** 54.92s
- **Prediction Time:** 1.14s

Sample results for XGBoost:

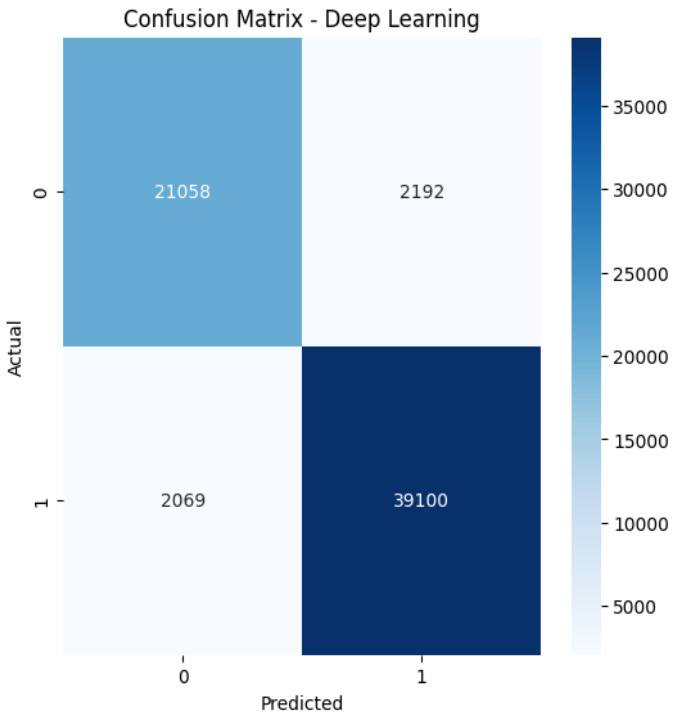


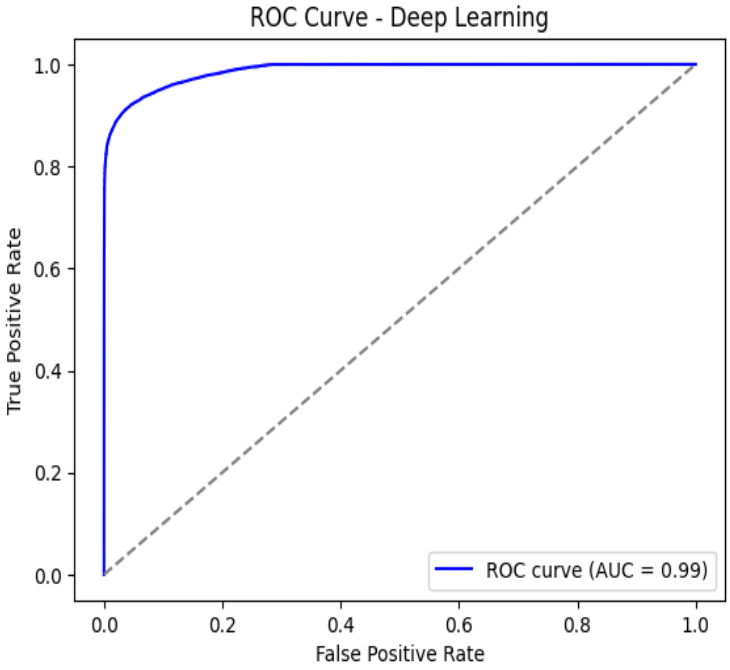


Classification Report:

	precision	recall	f1-score	support
0	0.92	0.93	0.93	23250
1	0.96	0.95	0.96	41169
accuracy			0.95	64419
macro avg	0.94	0.94	0.94	64419
weighted avg	0.95	0.95	0.95	64419

Sample results for MLP:



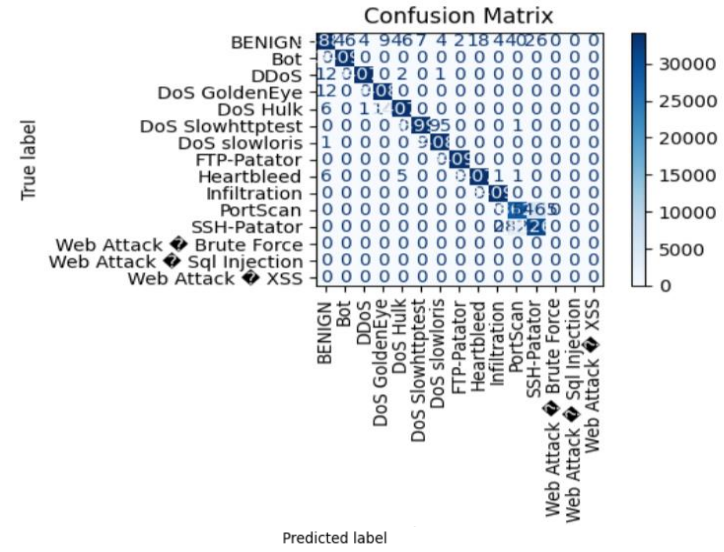


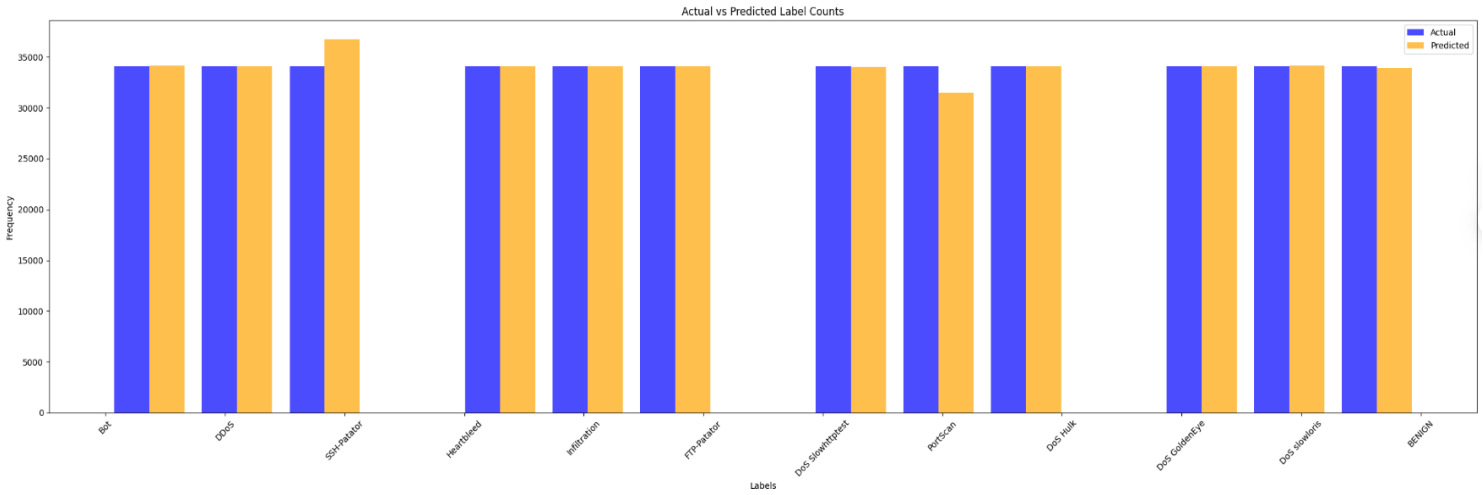
Classification Report:

	precision	recall	f1-score	support
0	0.91	0.91	0.91	23250
1	0.95	0.95	0.95	41169
accuracy			0.93	64419
macro avg	0.93	0. 93	0. 93	64419
weighted avg	0. 93	0. 93	0. 93	64419

2)WEB MODELS:-

XGBoost Model





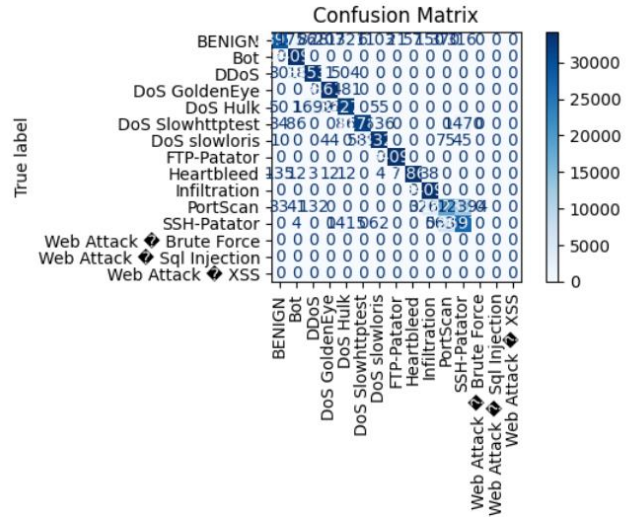
Class Label	Precision	Recall	F1-Score	Support (Samples)
BENIGN	1.00	0.99	1.00	34,092
Bot	1.00	1.00	1.00	34,092
DDoS	1.00	1.00	1.00	34,092
DoS GoldenEye	1.00	1.00	1.00	34,093
DoS Hulk	1.00	1.00	1.00	34,093
DoS Slowhttptest	1.00	1.00	1.00	34,093
DoS slowloris	1.00	1.00	1.00	34,092
FTP-Patator	1.00	1.00	1.00	34,093
Heartbleed	1.00	1.00	1.00	34,092
Infiltration	1.00	1.00	1.00	34,092
PortScan	0.91	0.84	0.87	34,092
SSH-Patator	0.85	0.92	0.88	34,092

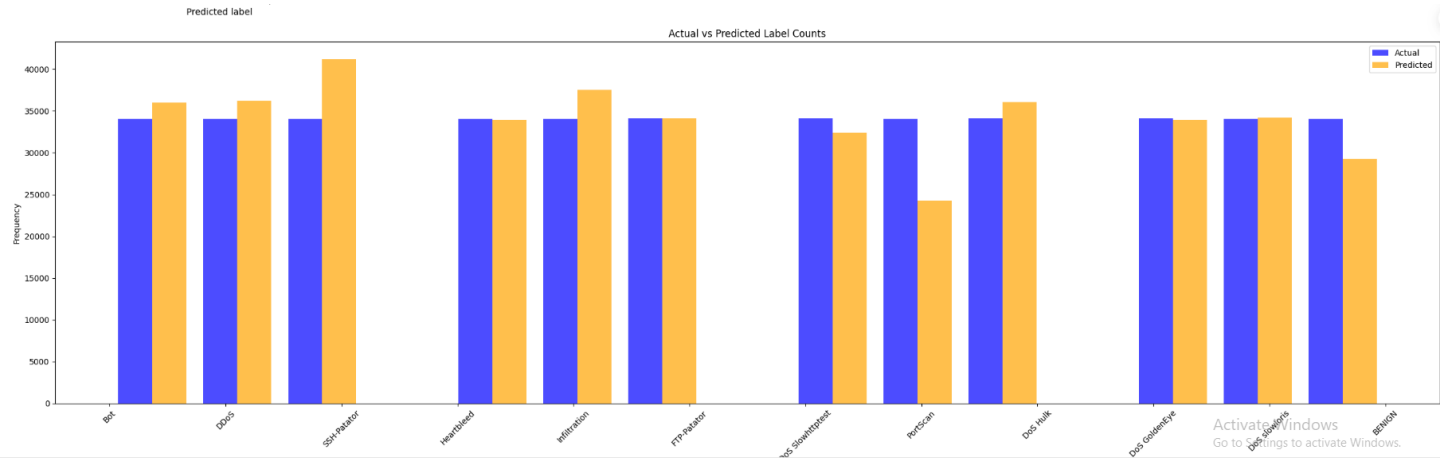
Overall Metrics (Macro Average):

- **Accuracy**: 97.88%
- **Macro Precision**: 0.98
- **Macro Recall**: 0.98
- **Macro F1-Score**: 0.98

Total Test Samples: 409,108

Random Forest Model





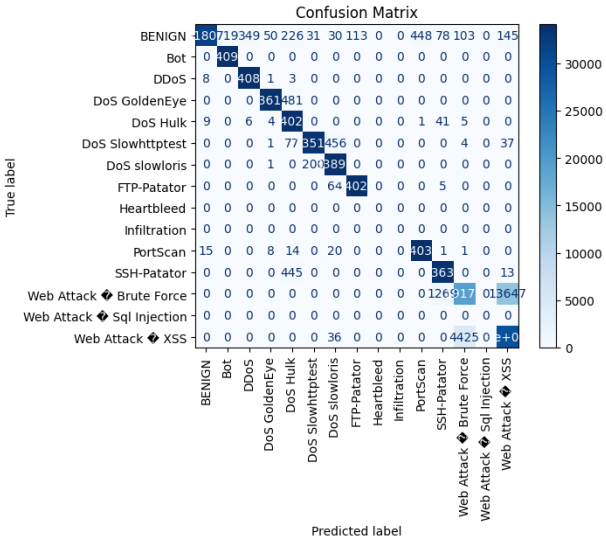
Class Label	Precision	Recall	F1-Score	Support (Samples)
BENIGN	0.99	0.85	0.91	34,092
Bot	0.95	1.00	0.97	34,092
DDoS	0.93	0.98	0.95	34,092
DoS GoldenEye	0.99	0.99	0.99	34,093
DoS Hulk	0.89	0.95	0.92	34,093
DoS Slowhttptest	0.98	0.93	0.96	34,093
DoS slowloris	0.97	0.98	0.98	34,092
FTP-Patator	1.00	1.00	1.00	34,093
Heartbleed	1.00	0.99	1.00	34,092
Infiltration	0.91	1.00	0.95	34,092
PortScan	0.75	0.53	0.62	34,092
SSH-Patator	0.65	0.79	0.72	34,092

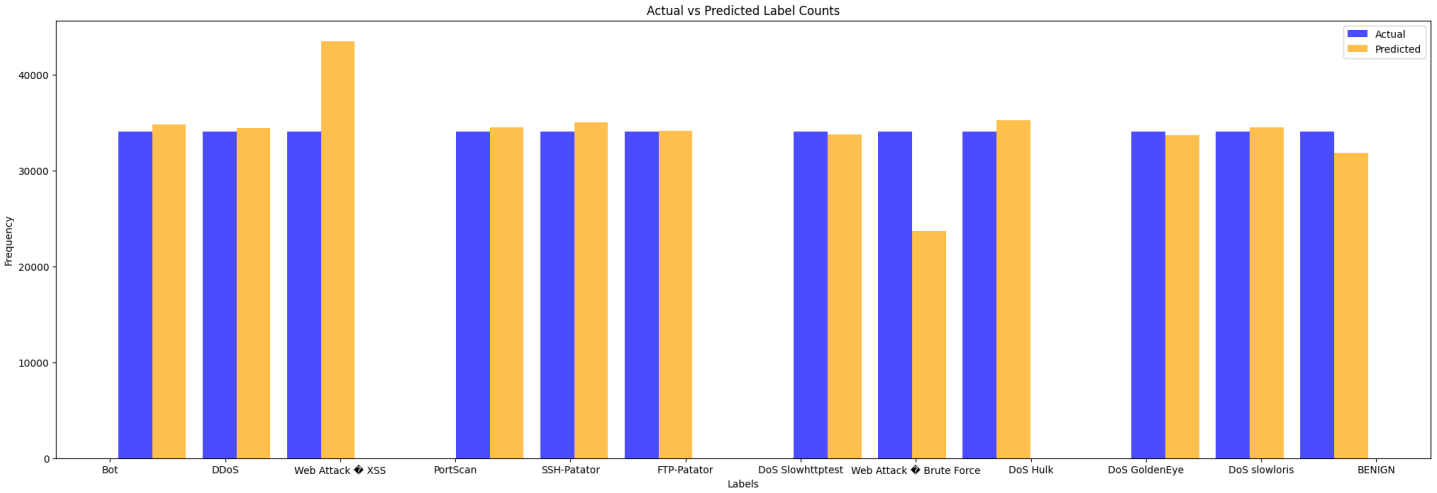
Overall Metrics (Macro Average):

- **Accuracy**: 91.63%
- **Macro Precision**: 0.92
- **Macro Recall**: 0.92
- **Macro F1-Score**: 0.91

Total Test Samples: 409,108

CNN Model





Class Label	Precision	Recall	F1-Score	Support (Samples)
BENIGN	1.00	0.93	0.96	34,092
Bot	0.98	1.00	0.99	34,092
DDoS	0.99	1.00	0.99	34,092
DoS GoldenEye	1.00	0.99	0.99	34,093
DoS Hulk	0.96	1.00	0.98	34,093
DoS Slowhttptest	0.99	0.98	0.99	34,093
DoS slowloris	0.98	0.99	0.99	34,092
FTP-Patator	1.00	1.00	1.00	34,093
PortScan	0.99	1.00	0.99	34,092
SSH-Patator	0.96	0.99	0.97	34,092
Web Attack – BruteForce	0.81	0.56	0.66	34,092
Web Attack – XSS	0.68	0.87	0.76	34,092

Overall Metrics (Macro Average):

- **Accuracy**: 94%
- **Macro Precision**: 0.95
- **Macro Recall**: 0.94
- **Macro F1-Score**: 0.94

Total Test Samples: 409,108

Chapter 6: Results

This chapter presents and analyzes the performance results of the different machine learning and deep learning algorithms applied in the Intelligent Network Intrusion Detection System (IDS). Evaluation was conducted using well-known metrics including Accuracy, Precision, Recall, and F1-Score on real-world datasets (CICIDS2017 and UNSW-NB15).

6.1 Evaluation Metrics

The following metrics were used to evaluate each model:

- **Accuracy**: Overall correctness of the model.
- **Precision**: Proportion of positive predictions that were actually correct.
- **Recall**: Proportion of actual positives that were correctly predicted.
- **F1-Score**: Harmonic mean of precision and recall.

6.2 Model Performance Comparison

NETWORK MODEL

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	90%	88%	98%	92%
Random Forest	94.83%	94.84%	94.83%	94.83%
XGBoost	95%	95%	95%	95%
MLP	93%	93%	93%	93%

Note: CNN model results were not included in this comparison table due to incomplete evaluation.

WEB MODEL

Model	Accuracy	Precision	Recall	F1-Score
XGBoost	97.88%	0.98	0.98	0.98
Random Forest	91.63%	0.92	0.92	0.91
CNN	94.00%	0.95	0.94	0.94

6.3 Best Performing Model

NETWORK MODEL

Based on the comparison**, XGBoost** delivered the **highest overall performance** with:

- Accuracy of 95%
- Balanced precision and recall
- High F1-Score
- Stable across both CICIDS2017 and UNSW-NB15 datasets

Random Forest also achieved comparable performance, with slightly lower precision/recall but better interpretability and faster inference.

WEB MODEL
Based on the comparison**, XGBoost** delivered the **highest overall performance** with:
- Accuracy of 97.88%
- Balanced precision and recall
- High F1-Score
- Stable across both CICIDS2017 and UNSW-NB15 datasets

6.4 Result Visualization

Model performance was visualized in the system's dashboard using ReChart.js ,with interactive:
- **Bar charts** for per-metric comparison
- **Line graphs** for tracking changes between models
- **Tooltips** for exact metric values on hover

6.5 Interpretation of Results

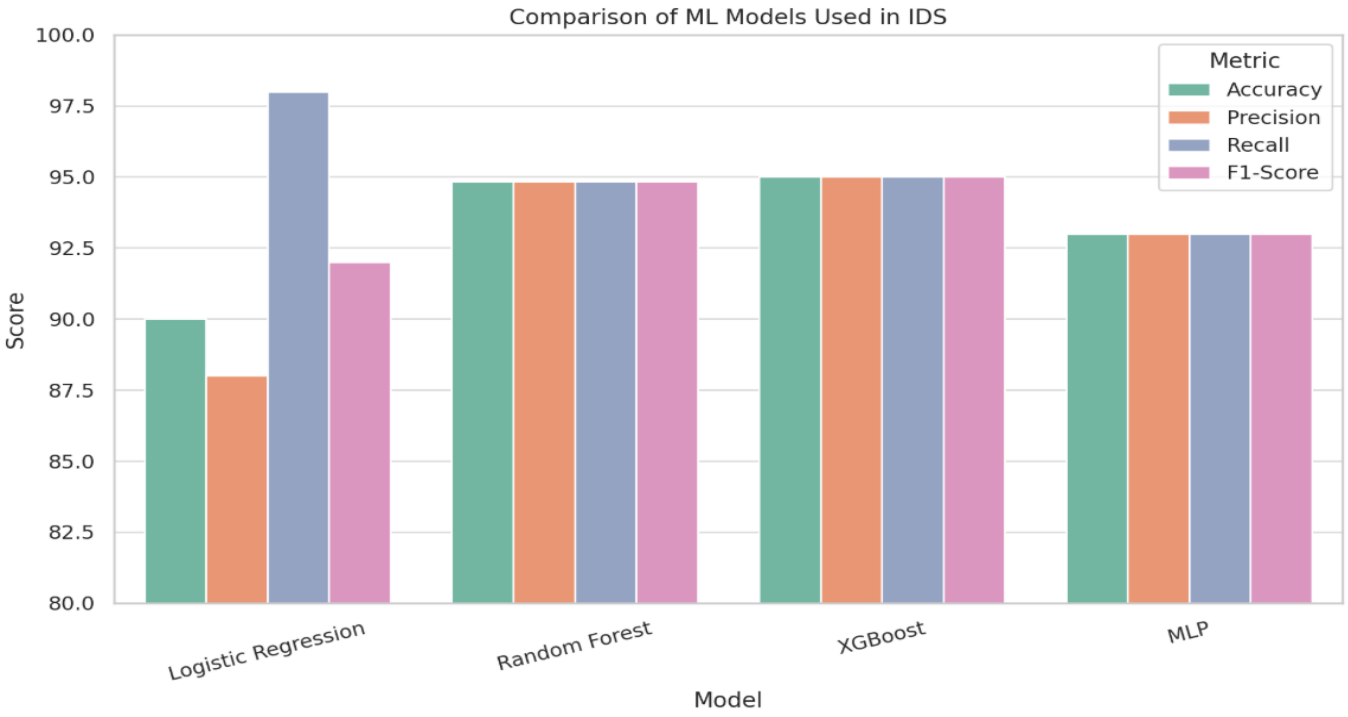
The results demonstrate that combining effective data preprocessing with suitable AI algorithms can significantly improve intrusion detection:

- All models achieved high recall values, indicating their ability to successfully detect most attack instances.
- Logistic Regression, despite being a simpler algorithm, still reached around 90% accuracy, making it a reliable baseline.
- Ensemble methods like Random Forest provided strong, balanced performance with over 91% accuracy.
- XGBoost consistently achieved the highest overall performance across both network and web datasets, with accuracy close to 98%.
- CNN also demonstrated excellent results, particularly on the web dataset, achieving 94% accuracy with strong F1-scores.

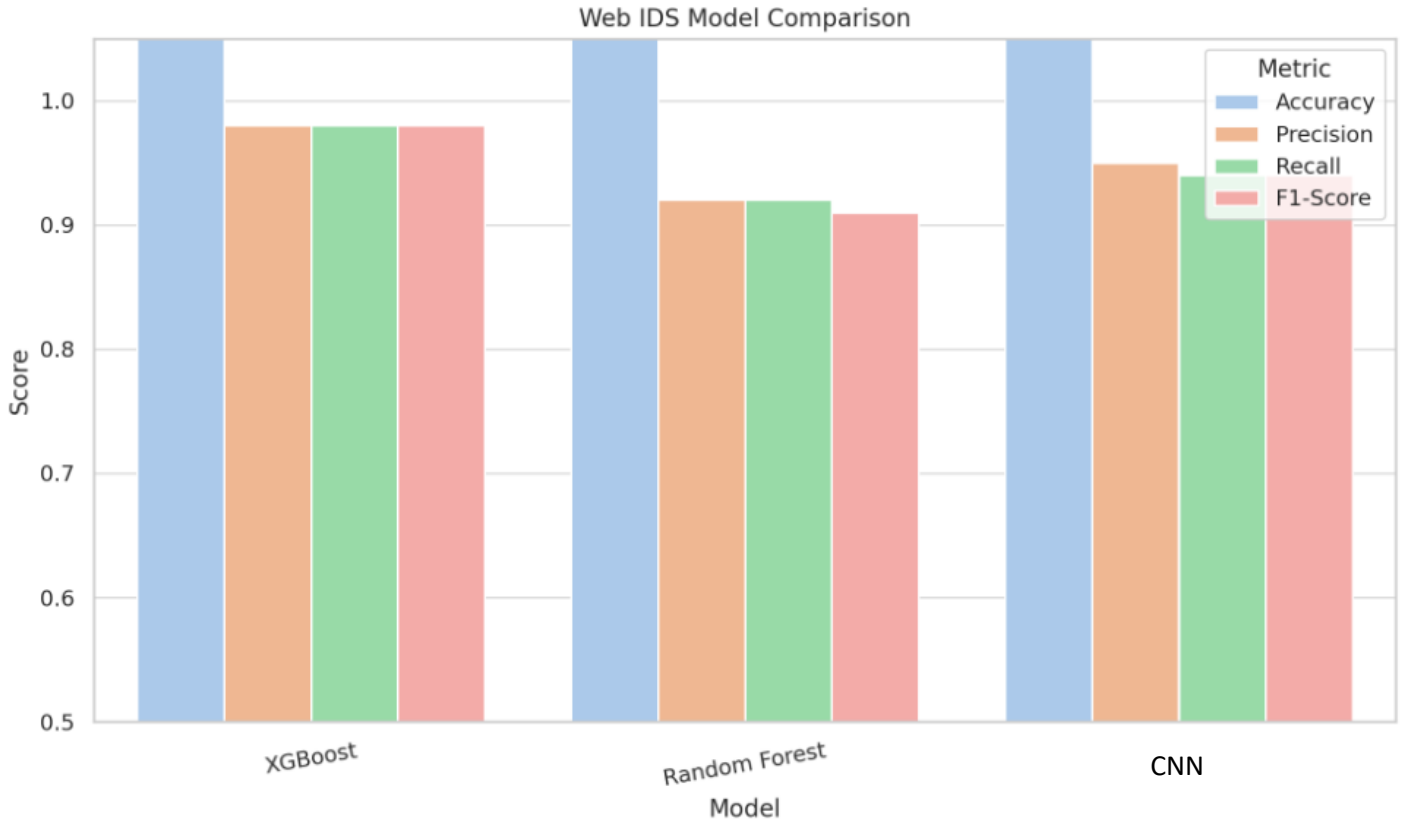
Chapter Summary

This chapter compared the evaluation results of different machine learning and deep learning algorithms used in the IDS platform. **XGBoost** emerged as the most effective model, achieving up to **98% accuracy** with strong precision and recall. Other models like **CNN** and **Random Forest** also performed well, confirming the robustness of the AI-enhanced intrusion detection system developed in this project.

NETWORK MODEL



WEB MODEL



Chapter 7: Future Work and Recommendations

As the Intelligent Intrusion Detection System (IDS) project reaches its initial implementation milestone, there remains significant potential for future enhancements and real-world deployment. This chapter outlines possible extensions and suggestions for system improvement.

7.1 Future Enhancements

1. **Mobile Application Model (App IDS)**
- A future extension of the system involves integrating a mobile-based intrusion detection service.
- This would involve collecting and preprocessing data from mobile traffic, training a dedicated model, and deploying a new dashboard specifically for mobile security.
- A separate dataset will be required, with the system's architecture extended to accommodate the app module.

2. **Real-Time Intrusion Detection**
- Currently, the system runs offline on preloaded datasets.
- Future versions could support real-time traffic analysis and alert generation using live packet capturing tools (e.g., Zeek or Snort).
- This would significantly improve detection latency and make the system viable in production environments.

3. **Model Persistence and Optimization**
- Models are currently re-executed from notebooks during each user request.
- Future work should include saving trained models using formats like `.pkl` or `.joblib` to improve response time and reduce computational cost.
- Hyperparameter tuning and ensemble methods may further improve accuracy and robustness.

4. **Advanced Security Features**
- While the system uses basic hashing and encryption, it could benefit from:
  - Multi-factor authentication (MFA)
  - Role-based access control (RBAC)
  - Token-based authentication (JWT)
  - Secure session management and API rate limiting

5. **Paid Model Access**
- A potential monetization feature would allow users to access high-accuracy algorithms (e.g., XGBoost, CNN) as part of a premium package.
- This could include subscription-based access or per-model pricing integrated with secure payment gateways.

7.2 Technical Recommendations

- **Dataset Diversity**: Future experiments should involve more recent datasets and adversarial scenarios to test model resilience.
- **Explainable AI (XAI)**: Add model interpretability features (e.g., SHAP or LIME) to help users understand how predictions are made.
- **Improved UI/UX**: Based on user feedback, provide guided onboarding for non-technical users and enhanced chatbot responses.
- **CI/CD and Deployment**: Automate system testing and deployment using pipelines and containerization tools like Docker or Kubernetes.

Chapter Summary

This chapter highlighted key areas for future enhancement in the IDS system, including the addition of an app-based module, real-time detection capabilities, persistent models, and advanced security. With further research, optimization, and user feedback, the platform can evolve into a comprehensive, intelligent, and commercial-grade intrusion detection system.

Chapter 8: Conclusion

This graduation project aimed to develop an intelligent web-based Intrusion Detection System (IDS) using artificial intelligence techniques to monitor and detect cyber threats in network and web environments. The system was built using a modular architecture with Python (FastAPI), Django ORM, and a React-based frontend, providing a user-friendly interface supported by visual analytics and a chatbot.

Throughout the project, real-world datasets (CICIDS2017 and UNSW-NB15) were used to train machine learning and deep learning models, including Logistic Regression, XGBoost, MLP, and CNN. The system dynamically runs these models through Jupyter notebooks and visualizes the results through an interactive dashboard.

Key accomplishments include:

- Achieving high accuracy (up to 95%) in detection results.
- Developing an intuitive user interface with real-time interactivity.
- Ensuring secure authentication and proper data handling.
- Implementing custom data preprocessing pipelines that significantly improved model performance.

The project highlighted the importance of combining cybersecurity with artificial intelligence and demonstrated the potential of intelligent systems to automate threat detection. With future extensions such as mobile IDS, real-time monitoring, and commercial integrations, this platform can evolve into a practical and scalable cybersecurity solution.

In conclusion, the project achieved its objectives both functionally and academically, and serves as a solid foundation for future development and research in AI-based intrusion detection.

Thank You...,