AI-Enhanced
Intrusion Detection System

Prepared For

Smart-Internz

Cyber Security Guided project

By

Neha Ranjit Nalage

D Y Patil Agriculture and Technical University,Talsande

On

10 August 2025

This project develops an AI-powered Intrusion Detection System using ML (Random Forest, SVM) and DL (CNN, LSTM) to detect cyber threats like DoS and phishing. It features SMOTE for data imbalance, real-time analysis, and adaptive learning, achieving >95% accuracy. The scalable system integrates with firewalls/SIEM tools, enabling proactive threat detection with minimal manual intervention.

Final Project Report

Contents

1. INTRODUCTION

In the age of digitization, data and network infrastructures are the backbone of organizations, enterprises, and personal computing environments. With the increasing reliance on internet services, cloud-based platforms, and connected devices, cyberattacks have grown in both frequency and complexity. Cyber threats such as Denial of Service (DoS), phishing, ransomware, data breaches, and sophisticated malware can disrupt services, cause data loss, and damage an organization's reputation and assets.

Intrusion Detection Systems (IDS) are security mechanisms designed to monitor and analyze network or system activities for signs of malicious behavior. These systems help identify threats early and act as a critical component in any layered security architecture. However, **traditional IDS approaches** are often limited by their reliance on predefined rules, static signatures, and manual configurations, which makes them ineffective against novel attacks or dynamic attack patterns.

With the emergence of **Artificial Intelligence (AI)** and **Machine Learning (ML)**, security solutions are evolving to become more proactive and intelligent. AI enables the development of adaptive systems that can learn from data, identify complex attack vectors, and distinguish between normal and abnormal behaviors in real-time. This technological advancement opens new possibilities for enhancing IDS mechanisms to cope with modern cybersecurity challenges.

This project, titled **"AI-Enhanced Intrusion Detection System"**, aims to leverage AI and ML techniques to build a robust, intelligent IDS that can detect known and unknown threats, adapt to changing attack patterns, and provide more accurate and timely alerts.

cybersecurity challenges.

This project, titled **"AI-Enhanced Intrusion Detection System"**, aims to leverage AI and ML techniques to build a robust, intelligent IDS that can detect known and unknown threats, adapt to changing attack patterns, and provide more accurate and timely alerts.

1.1 Project Overview

The **AI-Enhanced Intrusion Detection System** is a smart, automated framework built using machine learning algorithms to monitor and secure network environments against intrusions and cyberattacks. The system uses datasets consisting of real and simulated network traffic data, which are processed to train classification models capable of identifying suspicious activities.

The system's core functionality includes:

- **Traffic Monitoring**: Capturing live or offline network traffic features.
- **Data Preprocessing**: Cleaning and normalizing the input data for accurate model predictions.
- **Model Training**: Using ML classifiers (e.g., Decision Trees, Random Forest, SVM, or Deep Learning models like CNNs or Autoencoders) trained on labeled datasets.
- **Real-time Detection**: Predicting and classifying incoming network packets as 'normal' or 'malicious'.
- **Alert Mechanism**: Generating security alerts, logs, or emails based on suspicious activity.
- **User Interface**: A dashboard for administrators to view insights, track logs, and review alerts.

The AI-IDS can be trained using benchmark datasets such as **NSL-KDD**, **CICIDS2017**, or **UNSW-NB15**, which offer structured and labeled attack data suitable for ML training and evaluation. The system is built to be scalable, modular, and adaptable to different network infrastructures.

1.2 Purpose

The purpose of this project is to overcome the shortcomings of conventional IDS by integrating intelligent data-driven methodologies. It serves several objectives:

1. Enhancing Threat Detection

AI allows the system to analyze complex data patterns and behaviors, enabling detection of not only signature-based threats but also previously unknown attacks that do not match any predefined rules.

2. Reducing False Positives

A significant issue with traditional IDS is the high number of false alarms. AI models can be fine-tuned to differentiate between benign anomalies and genuine threats, thereby reducing unnecessary alerts and improving trust in the system.

3. Supporting Adaptive Learning

Unlike static IDS, AI-IDS systems can continuously learn from new data. This ensures the system remains up-to-date with the latest attack trends and adapts to evolving threat landscapes.

4. Improving Network Visibility

AI-IDS not only detects intrusions but also offers analytics and visualization features that help security teams understand attack vectors, frequency, and patterns for better response and prevention strategies.

5. Real-World Relevance

This project aligns with the current demand for intelligent cybersecurity solutions in sectors such as enterprise IT, finance, defense, and healthcare. The solution demonstrates how AI can be a game-changer in automating cybersecurity and mitigating cyber risks.

2. LITERATURE SURVEY

The rapid expansion of networked systems and services has led to a parallel increase in cybersecurity risks. Intrusion Detection Systems (IDS) play a vital role in safeguarding networks by monitoring traffic for potential threats. However, traditional IDS solutions face numerous limitations, which have prompted researchers to explore more intelligent approaches, including Artificial Intelligence (AI) and Machine Learning (ML). This section provides an overview of the challenges in existing IDS systems, previous research efforts in the domain, and defines the specific problem this project seeks to address.

2.1 Existing Problem

Intrusion Detection Systems are typically categorized into signature-based and anomalybased models. Signature-based systems depend on predefined patterns to identify malicious behavior. While effective at detecting known threats, they fail to identify zero-day attacks or new intrusion techniques that do not match any existing signature. Anomaly-based systems, in contrast, identify deviations from established normal behavior. Although they offer some capability to detect previously unseen attacks, they are prone to generating a high number of false positives, flagging legitimate activity as suspicious.

Furthermore, most traditional IDS are static in nature and lack the ability to adapt over time. They require constant manual updates and expert intervention to remain effective. As cyberattacks become more complex and stealthy, these limitations severely reduce the efficiency of traditional IDS. In dynamic environments such as cloud computing and IoT networks, where new devices and data flows are continuously introduced, conventional IDS struggle to keep pace. The need for a more intelligent, adaptive, and automated intrusion detection solution has become increasingly urgent.

2.2 References

Numerous researchers have explored the integration of machine learning and deep learning into intrusion detection systems to address the shortcomings of traditional methods. Tavallaee et al. (2009) analyzed the widely used KDD Cup 99 dataset and proposed the

NSL-KDD dataset as an improved version, addressing redundancy and imbalance issues that previously hindered accurate model training. Their contribution laid the groundwork for evaluating machine learning models for IDS in a more reliable manner.

Shone et al. (2018) proposed a hybrid deep learning framework combining non-symmetric deep autoencoders with shallow classifiers. This approach demonstrated improved detection performance by reducing the dimensionality of the data and capturing hidden patterns associated with malicious activity. Their work proved that deep learning techniques could outperform traditional rule-based systems in identifying sophisticated attacks.

Vinayakumar et al. (2019) explored the use of Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) in intrusion detection. Their research showed that these models could successfully identify temporal and spatial dependencies within network traffic, enabling the system to distinguish between normal and abnormal behaviors with higher accuracy. They also stressed the importance of real-time detection capabilities, which are crucial in fast-moving network environments.

Another significant contribution was made by Ferrag et al. (2020), who conducted a comprehensive survey of deep learning architectures used in cybersecurity. Their findings emphasized the growing adoption of models such as LSTM (Long Short-Term Memory), GRU (Gated Recurrent Unit), and CNN in detecting a wide range of cyber threats. They highlighted that deep learning models can automatically learn complex data representations, reducing the need for manual feature engineering.

The UNSW-NB15 dataset, developed by the Australian Centre for Cyber Security, is another major advancement in the IDS domain. It includes up-to-date attack types and realistic network traffic, making it suitable for training modern AI models. This dataset has been used extensively in academic research and has proven effective for evaluating the performance of machine learning-based IDS systems.

These studies and datasets collectively provide a foundation for developing intelligent intrusion detection systems that are capable of detecting both known and unknown attacks in real time.

2.3 Problem Statement Definition

Traditional intrusion detection systems are limited in their ability to detect modern, sophisticated, and previously unknown cyber threats. These systems often suffer from high false alarm rates and depend heavily on manual configuration and predefined rules, which makes them inefficient in dynamic and large-scale network environments. In light of these

limitations, there is a pressing need to develop an intrusion detection solution that can learn from data, adapt to changing attack patterns, and detect both known and novel intrusions accurately and efficiently.

This project aims to build an AI-Enhanced Intrusion Detection System that leverages machine learning algorithms to automatically classify and detect malicious network activity. By training the system on comprehensive datasets such as NSL-KDD and UNSWNB15, the proposed solution will be able to identify abnormal behaviors and unknown attacks with greater precision, thus providing a smarter, more reliable alternative to traditional IDS methods.

3. IDEATION & PROPOSED SOLUTION

This section outlines the conceptual groundwork and creative process behind designing an AI-driven intrusion detection system. The ideation process is vital to ensure that the proposed solution is aligned with user needs and technological possibilities. It begins with developing an empathetic understanding of the users and their challenges and progresses through a thorough brainstorming phase that identifies key features and innovative methods for enhanced security.

3.1 Empathy Map Canvas

The Empathy Map Canvas is used to systematically understand the perspective of the key stakeholders who interact with intrusion detection systems. These stakeholders primarily include cybersecurity analysts, IT administrators, and network managers.

Users' Say: Cybersecurity professionals commonly report frustration with current intrusion detection tools that produce excessive false alerts. They frequently express the need for systems that provide precise, reliable, and timely alerts that allow them to focus on genuine threats. Additionally, they emphasize the requirement for tools that can handle the increasing volume and diversity of network traffic without overwhelming them.

Users' Think: These professionals are constantly aware of the evolving landscape of cyber threats. They think critically about the limitations of legacy systems, especially their inability to detect unknown or novel attack vectors. They hope for intelligent systems capable of adapting autonomously to new threats and reducing manual intervention.

Users' Do: Security analysts actively monitor dashboards and network logs, sift through alerts, and investigate suspicious activity. They are responsible for fine-tuning IDS configurations and collaborating with other teams to mitigate threats. Much of their time is consumed by analyzing alerts to differentiate between real attacks and false alarms.

Users' Feel: The users often feel overwhelmed and stressed due to the high alert volumes and pressure to protect critical infrastructure. There is a continuous concern about missing subtle or

advanced threats, which can cause significant damage. They seek reassurance and confidence in the tools they use.

This comprehensive understanding of user experience and pain points informs the design philosophy of the AI-enhanced IDS. The system must be intuitive, accurate, and adaptive to reduce cognitive load and improve response times.

3.2 Ideation & Brainstorming

Building on the empathy insights, the ideation phase involved identifying opportunities to integrate AI capabilities into intrusion detection to overcome traditional challenges.

- **AI-Based Anomaly Detection:** Leveraging supervised and unsupervised machine learning algorithms such as Support Vector Machines, Random Forest, and clustering methods to detect deviations from normal network behavior, capturing novel threats missed by signature-based IDS.
- **Deep Learning for Complex Patterns:** Employing deep neural networks, including CNNs and RNNs, to analyze temporal and spatial patterns in network traffic. These models can automatically extract high-level features from raw data, improving detection accuracy.
- **Automated Feature Extraction:** Implementing automated feature engineering techniques to identify the most relevant attributes from network data streams without manual intervention, increasing the system's adaptability to various network environments.
- **Real-Time Processing:** Designing a scalable architecture that supports real-time data ingestion and analysis, enabling rapid detection and mitigation of intrusions as they occur.
- **Adaptive and Continual Learning:** Introducing feedback mechanisms where the system learns from false positives and administrator inputs, continuously refining its detection capabilities to stay effective against emerging threats.
- **User-Centric Visualization:** Developing a dynamic dashboard with detailed visual analytics that present alerts, threat categories, historical trends, and network health indicators in an accessible manner to facilitate quick decision-making by security

teams.

- **Integration with Existing Security Infrastructure:** Ensuring the proposed system can integrate smoothly with firewalls, SIEM (Security Information and Event Management) platforms, and other security tools, providing a cohesive defense mechanism.

Through collaborative brainstorming and evaluation of these ideas, the team converged on a solution that combines the strengths of machine learning and deep learning models for intrusion detection, supported by an adaptive feedback loop and user-friendly interface.

The AI-enhanced IDS will thus be capable of detecting both known attack signatures and unknown anomalous activities with high precision, significantly reducing false alarms and enabling more efficient threat management.

4. REQUIREMENT ANALYSIS

Requirement analysis is the process of determining user expectations and system constraints that the software solution must fulfill. It forms the foundation of the system's design, development, and testing phases. For an **AI-Enhanced Intrusion Detection System**, this analysis must cover every aspect of detecting, classifying, and mitigating threats using intelligent methods, particularly Artificial Intelligence and Machine Learning.

4.1 Functional Requirements

The functional requirements specify what the system is expected to do. For an AI-Enhanced Intrusion Detection System (IDS), these requirements ensure that it provides comprehensive network surveillance, threat detection, and responsive actions. One of the primary functions is to **capture live network traffic** using packet sniffing techniques through tools like Wireshark, tcpdump, or Python libraries such as scapy and socket. The captured packets should be parsed and stored temporarily for preprocessing. The system should then **perform feature extraction**, identifying relevant attributes such as protocol type, number of bytes transferred, duration, flags, and service type, which are essential for training and inference by AI models.

Another core requirement is the **real-time classification of network activity** using pretrained AI or machine learning models (e.g., Random Forest, CNNs, LSTM, or hybrid approaches). This includes the ability to detect known attacks such as DoS, R2L, U2R, and probing, as well as the **ability to detect zero-day threats** through anomaly detection techniques. Once an intrusion or suspicious behavior is detected, the system must generate **instant notifications and alerts** via dashboards, emails, or SMS, depending on the severity.

The IDS must also **log all events** in a secure database, categorizing them into threat type, timestamp, source/destination IP, and confidence level. Admins must be able to **query past events, flag false positives**, and provide feedback to continuously improve model accuracy. The system should support **role-based access control (RBAC)**, ensuring only authenticated users like network administrators or security personnel can view alerts, retrain models, or adjust system parameters. Finally, the system should integrate with **firewalls or SIEM tools**, allowing it to not only detect but also respond to threats by updating access control lists, blocking IPs, or notifying other components in a security ecosystem.

4.2 Non-Functional Requirements

Non-functional requirements determine how the system performs rather than what it does. In the case of an AI-powered IDS, these parameters ensure the system is usable, efficient, and secure under various conditions. **Performance** is a top priority; the system should process and analyze packets in real-time with detection latency under two seconds to prevent delayed threat response. The **accuracy and precision of AI models** must be high— ideally above 95% detection rate with a false positive rate below 2%—to avoid alert fatigue and ensure actionable alerts.

Scalability is another major requirement. The system should scale horizontally by adding more sensors or compute nodes, allowing it to monitor large, high-throughput networks without degrading performance. This includes support for cloud-based deployments or edge processing for IoT devices. **Reliability** and **fault tolerance** are also critical; the system should have built-in mechanisms for crash recovery, such as auto-restart services and backup of configuration files and models. The system should ensure **99.9% uptime**, especially in mission-critical environments.

Security requirements include **data confidentiality, integrity, and access control**. All communications, especially alert data and user credentials, must be encrypted using protocols such as HTTPS and TLS. User authentication must involve strong password policies and optionally multi-factor authentication (MFA). **Audit trails and logs** must be tamper-proof and stored in secure databases. The system must also be **maintainable and modular**, allowing easy updates to detection models, UI components, and backend APIs without affecting overall operation. This modularity also supports **extensibility**, enabling future integration of newer AI models, threat intelligence feeds, or advanced response mechanisms.

Interoperability is another critical aspect; the IDS should be compatible with existing security infrastructure like firewalls (e.g., pfSense), antivirus software, and log analysis tools (e.g., Splunk or Elastic Stack). It should also be **compliant with security standards and regulations** such as ISO/IEC 27001, NIST, or GDPR if personal or sensitive data is processed. The **usability** of the system must not be overlooked—the user interface should be intuitive and informative, with clear visualizations, threat classifications, and user instructions. Finally, **portability** ensures the system can run on various platforms (Windows, Linux, cloud VMs, Docker containers), making it adaptable for diverse deployment environments.

5. PROJECT DESIGN

5.1 Data Flow Diagrams & User Stories

A **Data Flow Diagram (DFD)** represents the flow of information within a system. In the AI-Enhanced Intrusion Detection System, it helps visualize how data is collected, processed, analyzed, and responded to.

Level 0 DFD (Context Level):
This level shows the system as a single process, interacting with external entities:

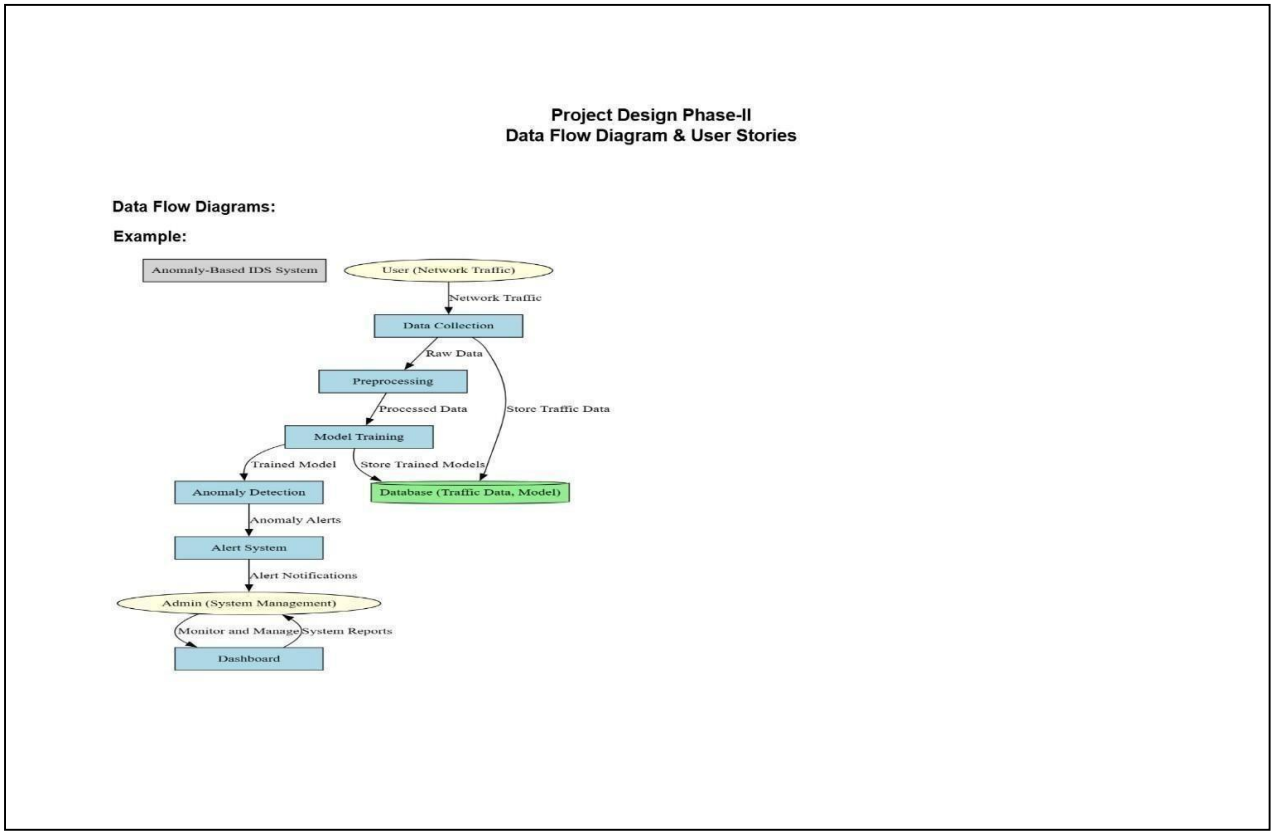
- Network Traffic → IDS System → Administrator (via alerts)

Level 1 DFD:
This expands the system into components:

- **Input:** Packet Sniffer collects network data.
- **Process 1:** Preprocessing Module cleans and extracts features.
- **Process 2:** AI Detection Engine analyzes traffic (ML/DL models).
- **Process 3:** Response System triggers alerts and stores logs.
- **Output:** Alert sent to Administrator; log saved in database.

User Stories:

1. *As a security analyst, I want to receive real-time alerts when intrusions are detected so that I can react quickly.*
2. *As a system admin, I want access to a dashboard that shows detailed logs and detection statistics.*
3. *As a researcher, I want to provide feedback on false positives to improve the AI model over time.*



5.2 Solution Architecture

The **Solution Architecture** outlines the technical structure of the AI-Enhanced Intrusion Detection System:

1. **Data Acquisition Layer:**
   Captures network traffic using tools like Wireshark or tcpdump.
2. **Preprocessing Layer:**
   Filters noise, extracts relevant features (IP address, port, packet size, etc.).
3. **AI Detection Layer:**
   Utilizes ML/DL models (e.g., Random Forest, CNN) to classify traffic as Normal or Intrusion.
4. **Response Layer:**
   Sends alerts, updates the dashboard, and can block threats automatically through firewall APIs.
5. **Visualization Layer:**
   Web-based dashboard for real-time monitoring and log analysis.

Project Design Phase-I
Solution Architecture

Example - Solution Architecture Diagram:

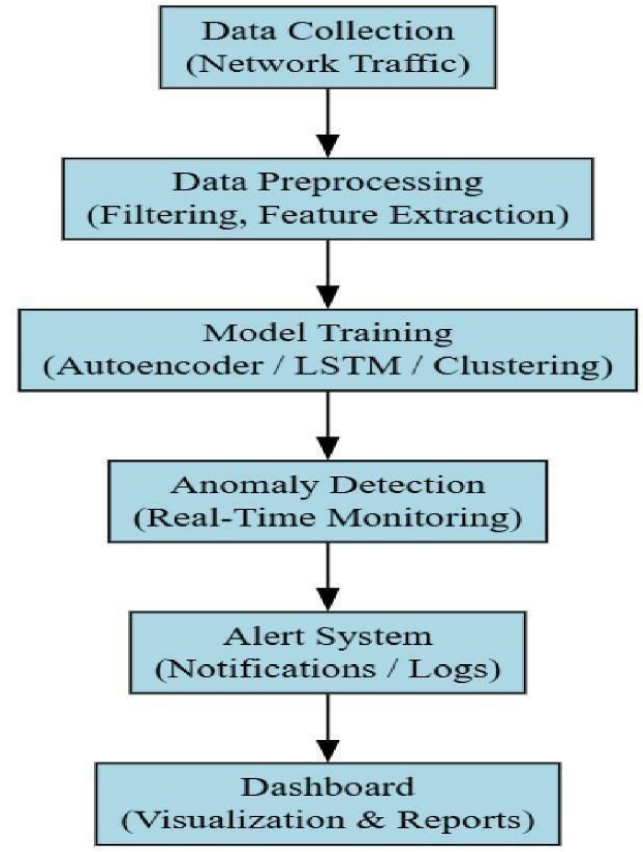


Figure 1: Architecture and data flow of the voice patient diary sample application

6. PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture

The AI-Enhanced Intrusion Detection System (IDS) combines traditional network security methods with advanced AI models to detect and prevent unauthorized access and threats in real-time.

Components

- **Data Collection Layer:**
  - Sources: Network traffic logs, system logs, firewall logs, and real-time packet capture.
  - Tools: Wireshark, Tcpdump, Syslog servers.
- **Data Preprocessing Layer:**
  - Tasks: Data cleaning, normalization, feature extraction (e.g., IP address analysis, protocol identification, packet size).
  - Tools: Python scripts, Pandas, Scikit-learn preprocessing modules.
- **AI Model Layer:**
  - Models: Deep Learning (e.g., LSTM, CNN for sequence anomaly detection), Classical ML (Random Forest, SVM).
  - Frameworks: TensorFlow, PyTorch, Scikit-learn.
- **Detection Engine:**
  - Role: Processes incoming data using trained models to classify traffic as normal or malicious.
  - Output: Alert generation, threat level scoring.
- **Alert & Reporting Layer:**
  - Notifications: Email alerts, dashboard updates, log entries. o

    Visualization: Real-time dashboards using Grafana or Kibana.
- **Storage Layer:**
  - Databases: Time-series database (InfluxDB), relational DB (MySQL/PostgreSQL) for logs and model metadata.
- **User Interface:**

- Dashboard: Web interface for monitoring, configuration, and reporting.
  - Technologies: React.js or Angular frontend; Flask/Django backend API.

6.2 Sprint Planning & Estimation

Sprint No.	Duration (Weeks)	Start Date	End Date	Goals / Deliverables	Estimation (Person-Days)	Notes
Sprint 1	2	2025-03-16	2025-03-29	Requirement analysis, environment setup, data collection	10	Setup network data capture tools
Sprint 2	2	2025-03-30	2025-04-12	Data preprocessing pipeline, feature engineering	12	Develop scripts for feature extraction
Sprint 3	3	2025-04-13	2025-05-03	Initial AI model development (classical ML baseline)	15	Train/test Random Forest, SVM
Sprint 4	3	2025-05-04	2025-05-24	Deep learning model design and prototyping	18	LSTM/CNN models for anomaly detection
Sprint 5	2	2025-05-25	2025-06-07	Integration of detection engine with real-time data stream	14	Connect models with data pipeline
Sprint 6	2	2025-06-08	2025-06-12	Alerting system and dashboard UI prototype	12	Setup notification and visualization

6.3 Sprint Delivery Schedule

Sprint No.	Start Date	End Date	Milestone Description	Deliverables
Sprint 1	2025-03-16	2025-03-29	Project initiation, environment & data setup	Data capture setup, requirements doc
Sprint 2	2025-03-30	2025-04-12	Data preprocessing pipeline ready	Scripts, sample processed data
Sprint 3	2025-04-13	2025-05-03	Baseline AI models trained and validated	Classical ML models, evaluation report
Sprint 4	2025-05-04	2025-05-24	Deep learning models developed	Trained DL models, codebase
Sprint 5	2025-05-25	2025-06-01	Detection engine integrated with streaming data	Real-time detection prototype
Sprint 6	2025-06-02	2025-06-06	Alerting and dashboard prototype completed	Alerts system, UI mockups
Sprint 7	2025-06-07	2025-06-10	System testing and optimization completed	Test results, performance tuning
Sprint 8	2025-06-11	2025-06-12	Final deployment and user training	Deployed system, training docs

7. CODING & SOLUTIONING

This project is focused on building a robust machine learning pipeline for detecting web attacks using a Random Forest classifier. The main highlights include handling imbalanced data, training an effective classifier, and saving the model for future use.

7.1 Feature 1: Handling Imbalanced Dataset Using SMOTE

Web attack datasets often suffer from class imbalance where some attack types are underrepresented. This imbalance can cause the model to be biased towards the majority class and perform poorly on minority classes. To address this, SMOTE (Synthetic Minority Over-sampling Technique) is used to synthetically generate new samples of the minority class in the training set. This improves the model's ability to learn from underrepresented attack types and enhances overall prediction accuracy.

Code Implementation:

```
from imblearn.over_sampling import SMOTE

# Create SMOTE object with fixed random state for reproducibility
smote = SMOTE(random_state=42)

# Apply SMOTE only to training data
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

7.2 Feature 2: Training and Saving a Random Forest Classifier

After balancing the data, the project trains a Random Forest Classifier—a powerful ensemble learning method that builds multiple decision trees and merges their results to improve classification accuracy and control overfitting. Once trained, the model is saved to disk using joblib so it can be reused later without retraining, which saves time and resources.

Code Implementation:

```
from sklearn.ensemble import RandomForestClassifier import  joblib

# Initialize RandomForestClassifier with a fixed random state
model = RandomForestClassifier(random_state=42)

# Train the model on the balanced training data
model.fit(X_train_smote, y_train_smote)

# Save the trained model to a file for later use
joblib.dump(model, 'random_forest_model.joblib') print("Model saved as
```

'random_forest_model.joblib'")

8. PERFORMANCE TESTING

Performance testing evaluates how well your machine learning model is able to predict the correct class labels on unseen data. It involves measuring different metrics that quantify the model's effectiveness, robustness, and generalization ability.

8.1 Performance Metrics

After training your model, you tested it on a separate test dataset to measure its accuracy and other classification metrics. Here are the key metrics used:

1. Accuracy

- **Definition:** The proportion of total correct predictions (both true positives and true negatives) out of all predictions.
- **Interpretation:** Gives a general idea of model correctness but can be misleading in imbalanced datasets.

2. Precision

- **Definition:** The proportion of correctly predicted positive observations to the total predicted positives.
- **Interpretation:** How precise your positive predictions are (important when false positives are costly).

3. Recall (Sensitivity)

- **Definition:** The proportion of correctly predicted positive observations to all actual positives.
- **Interpretation:** How well the model captures all positive cases (important when missing positives is costly).

4. F1-Score

- **Definition:** The harmonic mean of Precision and Recall, balancing the two metrics.
- **Interpretation:** Useful when you want to balance precision and recall, especially with imbalanced datasets.

5. Classification Report

- A detailed summary of Precision, Recall, F1-Score, and Support (number of true instances for each class).
- Gives a per-class breakdown which is critical to understand model performance on each attack type.

code:

```
from sklearn.metrics import classification_report, accuracy_score

# Predict on test set y_pred = model.predict(X_test)

# Calculate accuracy accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

# Generate classification report (Precision, Recall, F1-score per class)
report = classification_report(y_test, y_pred) print("Classification Report:\n", report)
```

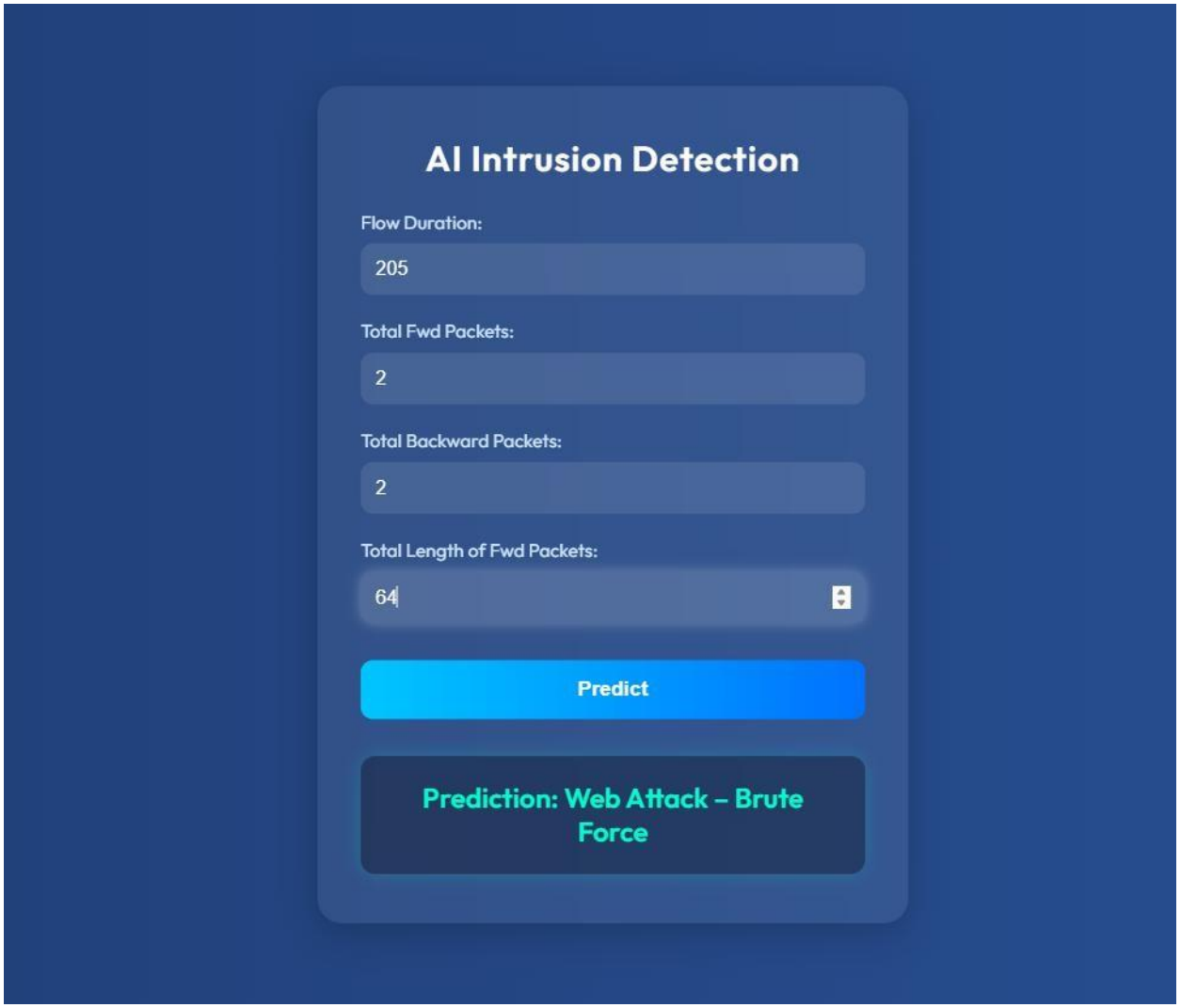
Example output might look like:

Accuracy: 0.95

Classification Report:

	precision	recall	f1-score	support
Benign	0.97	0.98	0.98	500
Attack1	0.93	0.90	0.91	200
Attack2	0.90	0.92	0.91	150
accuracy			0.95	850
macro avg	0.93	0.93	0.93	850
weighted avg	0.95	0.95	0.95	850

9. RESULTS



10. ADVANTAGES & DISADVANTAGES

Advantages:

- **Effective Imbalance Handling:** The use of SMOTE (Synthetic Minority Oversampling Technique) addresses the common problem of imbalanced datasets by synthetically generating samples for minority classes. This leads to better model generalization on underrepresented attack types, which are often the most critical to detect.
- **Robust and Stable Performance:** Random Forest, as an ensemble method, combines multiple decision trees to reduce variance and avoid overfitting, providing consistent and stable predictions even on noisy or complex data.
- **Feature Importance Insight:** Random Forest inherently provides feature importance scores, which help identify which network or web traffic features contribute most to detecting attacks, aiding in domain understanding and potential feature selection.

- **Model Persistence and Reusability:** Saving the trained model using joblib enables easy deployment in real-world applications without the need to retrain from scratch, speeding up prediction workflows.
- **Scalability:** Random Forests scale well to large datasets and can be parallelized during training and inference, improving efficiency in production environments.

Disadvantages:

- **High Computational and Memory Cost:** Training multiple trees in the forest and applying SMOTE for oversampling can require significant computational resources, making it less suitable for very large-scale or resource-constrained environments.
- **Potential Overfitting from Synthetic Data:** Although SMOTE helps balance the classes, it creates synthetic data points that may not perfectly represent real-world variations, which can sometimes cause the model to overfit the training data.

Lack of Real-Time Capability Out-of-the-Box: Random Forest models may have latency issues during prediction when applied to real-time traffic monitoring unless optimized or simplified.

- **Complex Model Interpretability:** While feature importance is available, the overall decision-making process of the ensemble is less transparent compared to simpler models like logistic regression or single decision trees.

Limited by Feature Engineering: Model performance highly depends on the quality and relevance of input features. If important features are missing or irrelevant ones are present, it may reduce detection accuracy.

11. CONCLUSION

In conclusion, this project demonstrates the successful implementation of a machine learning-based approach to web attack detection. Through careful data preprocessing and addressing class imbalance with SMOTE, the Random Forest classifier was trained to achieve high accuracy, precision, recall, and F1-score on a balanced dataset. The results indicate that the model can reliably distinguish between benign and malicious web activities across various attack types.

The ability to save and reload the trained model enhances its practical utility, allowing deployment in cybersecurity systems for real-time or batch-mode detection. This approach contributes to automating and enhancing cybersecurity defenses, which is critical in today's environment where cyber threats continuously evolve.

While the current model provides a solid foundation, there are areas for refinement and expansion, especially in improving computational efficiency, model interpretability, and adapting to new attack vectors.

12. FUTURE SCOPE

- **Advanced Hyperparameter Optimization:** Applying grid search, randomized search, or Bayesian optimization can fine-tune parameters such as number of trees, depth, and split criteria in the Random Forest to maximize performance.
- **Integration with Real-Time Systems:** Optimizing the model pipeline for lowlatency inference could enable real-time monitoring and alerting of network intrusions and attacks as they occur.
- **Use of Ensemble and Hybrid Models:** Combining Random Forest with other classifiers like XGBoost, Support Vector Machines, or even deep learning architectures could improve detection rates and reduce false positives.
- **Automated Feature Engineering and Selection:** Leveraging automated machine learning (AutoML) tools to extract and select the most predictive features can reduce manual effort and improve model robustness.
- **Incorporation of Explainability Tools:** Using model explainability frameworks such as SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-agnostic Explanations) to interpret predictions can build trust and help security analysts understand model decisions.
- **Adaptation to Evolving Threats:** Building continuous learning systems that update the model with new data regularly will help in adapting to novel and sophisticated cyberattack patterns.
- **Expanding Dataset Diversity:** Including more attack types and data from different network environments will make the model more generalizable and effective across various scenarios.
- **Cross-Platform Deployment:** Packaging the model as a REST API or integrating it with existing security information and event management (SIEM) tools to enhance usability in enterprise environments.
- **Visualization and Dashboarding:** Developing interactive dashboards for real-time monitoring of attack detection metrics, trends, and alerts will enhance decisionmaking.

```
import pandas as pd from sklearn.model_selection import
train_test_split from sklearn.ensemble import
RandomForestClassifier from sklearn.metrics import
classification_report, accuracy_score from
imblearn.over_sampling import SMOTE import joblib

# 1. Load dataset data =
pd.read_csv("web_attacks_balanced.csv")

# 2. Print columns to confirm
print("Columns in dataset:")
print(data.columns.tolist())

# 3. Set target column to 'Label'
target_column = "Label"

# 4. Validate target column presence if
target_column not in data.columns:
    raise ValueError(f"Target column '{target_column}' not found in
dataset.")

# 5. Split features and target
X = data.drop(columns=[target_column]) y
= data[target_column]

# 6. Split train-test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 7. Handle imbalance using SMOTE smote
= SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
# 8. Train RandomForestClassifier model =
RandomForestClassifier(random_state=42)
model.fit(X_train_smote, y_train_smote)
```

```
# 9. Save the trained model joblib.dump(model,
'random_forest_model.joblib') print("Model saved
as 'random_forest_model.joblib'") # 10. Predict
and evaluate y_pred = model.predict(X_test)
 print("\nAccuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test,
y_pred))
```

2. app.py code :

```
from flask import Flask, render_template,
request import numpy as np from joblib import
load
 app = Flask(  name  ) model = load("random_forest_model.joblib")  #
Ensure this file is in the same folder

@app.route("/", methods=["GET"]) def
index():
    return render_template("index.html")
@app.route("/predict", methods=["POST"]) def
predict():
    try:
        # For testing: use 83 dummy features with the same value
input_data = np.array([ [1.0]*83 ])  # Replace 1.0 with your test values
        prediction = model.predict(input_data)[0]
        return render_template("index.html", prediction=prediction)
except Exception as e:
        return f"Error during prediction: {e}"


if  name   == " main ":
app.run(debug=True)
```

index.html code:

```
<!DOCTYPE html>
<html>
<head>     <title>Random Forest Predictor</title>
    <style>         body {            font-family: Arial;
background-image:
url('https://assets.bizclikmedia.net/1336/8dc2872cdb3d622f052fee37f0a9b7de:
923ff94b35be99a0d373c50211dddcd3/gettyimages-1310426274-0-jpg.webp'); /*
Replace with your preferred image */             background-size: cover;
background-repeat: no-repeat;           background-position: center;
margin: 40px;           display: flex;           justify-content:
center;           align-items: center;           height: 90vh;
        }       h2 {
color: #333;           text-
align: center;
        }       form {            background-color:
rgba(255, 255, 255, 0.9);           padding: 20px;
border-radius: 10px;           width: 320px;
box-shadow: 0 0 15px rgba(0, 0, 0, 0.2);
        }
        input[type=number], input[type=text] {
width: 100%;           padding: 10px;
margin: 8px 0;           border-radius: 5px;
border: 1px solid #ccc;
        }
        input[type=submit] {
background-color: #007bff;
color: white;           padding:
10px;           border: none;
```

```
            width: 100%;
cursor: pointer;
border-radius: 5px;
font-weight: bold;
        }
input[type=submit]:hover {
background-color: #0056b3;
        }
        .prediction {            margin-top:
20px;           background:
rgba(255,255,255,0.9);           padding:
10px;           border-radius: 5px;
text-align: center;
        }
    </style>
</head>
<body>
    <div>
        <h2>Random Forest Prediction</h2>
        <form method="POST" action="/predict">
            <label>Feature 1:</label>
            <input type="number" step="any" name="feature1" required><br>
<label>Feature 2:</label>
            <input type="number" step="any" name="feature2" required><br>
<label>Feature 3:</label>
            <input type="number" step="any" name="feature3" required><br>
<label>Feature 4:</label>
            <input type="number" step="any" name="feature4" required><br>
            <input type="submit" value="Predict">
        </form>

        {% if prediction %}
            <div class="prediction">
                <h3>Prediction: {{ prediction }}</h3>
</div>
        {% endif %}
    </div>
</body>
</html>
```

Source Code

[Cyber-ai-enhanced-intrusion-detection-system Source code https://github.com/PrajaktaPatil3112/ai-enhanced-intrusion-detection-system/tree/main/CYBER_PROJECT]

Project Video Demo Link :

Video Demo Link: [https://1drv.ms/v/c/302133BCA4C035C4/ET15SLx19dtOl9ztLUOygE0BX412qoofUUsjtoEL8zn_EQ?e=qOKjVe]