

# **AI-NIDS: AI-Powered Network Intrusion Detection System Using Machine Learning and Deep Learning**

*Project report submitted to  
Indian Institute of Information Technology, Nagpur,  
in partial fulfillment of the requirements for the Mini Project - II*

at

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

by

**Chetan Rajesh Narware (BT22CSD001)**

**Aman Ravindra Zade (BT22CSD004)**

**Kshitij Tripathi (BT22CSD029)**

**Under the Supervision of**

**Ms. Madhavi Netke**

**Session Period: Jan to May 2025**



**भारतीय सूचना प्रौद्योगिकी संस्थान, नागपुर**

**INDIAN INSTITUTE OF INFORMATION TECHNOLOGY,  
NAGPUR**

**(An Institution of National Importance by Act of Parliament)  
Survey No.140,141/1 behind Br.Sheshrao Wankhede Shetkari Sahkari Soot  
Girni,Nagpur, 441108**



**Declaration**

We, **Mr. Chetan Rajesh Narware, Mr. Aman Ravindra Zade and Mr. Kshitij Tripathi**, hereby declare that this project work titled “**AI-NIDS: AI-Powered Network Intrusion Detection System Using Machine Learning and Deep Learning**” is carried out by us in the Department of Computer Science and Engineering of Indian Institute of Information Technology, Nagpur. The work is original and has not been submitted earlier whole or in part for the award of any other certification programme at this or any other Institution/University.

**Date: 19/05/2025**

Sr. No.	Names	Signature
1	Chetan Rajesh Narware	
2	Aman Ravindra Zade	
3	Kshitij Tripathi	

## **Certificate**

This is to certify that the project titled “**AI-NIDS: AI-Powered Network Intrusion Detection System Using Machine Learning and Deep Learning**”, submitted by **Mr. Chetan Rajesh Narware, Mr. Aman Ravindra Zade and Mr. Kshitij Tripathi** in partial fulfillment of the requirements for the Mini-Project in CSE department IIIT Nagpur. The work is comprehensive, complete and fit for final evaluation.

Date: 19/05/2025

Ms. Madhavi Netke  
Assistant Professor, CSE, IIIT, Nagpur

## **Acknowledgement**

We would like to express our heartfelt gratitude to our supervisor, **Ms. Madhavi Netke**, for her invaluable guidance, encouragement, and support throughout the duration of this project. Her expertise and insights were instrumental in shaping our work and helping us overcome challenges.

We are also grateful to the **Department of Computer Science and Engineering** at the **Indian Institute of Information Technology, Nagpur**, for providing us with the resources and environment conducive to learning and research.

Additionally, we thank our peers and colleagues for their constructive feedback and collaboration, which enriched the experience of working on this project. Finally, we are deeply thankful to our families and friends for their unwavering support and understanding during the course of this work. This project has been a significant learning experience, and we are grateful for the opportunity to contribute to the field of cybersecurity through this endeavor.

**Chetan Rajesh Narware**

**Aman Ravindra Zade**

**Kshitij Tripathi**

## Abstract

With the exponential growth of internet-connected systems and increasing sophistication of cyberattacks, the need for robust and intelligent network security mechanisms has become more critical than ever. Traditional Intrusion Detection Systems (IDS), which rely heavily on predefined rule sets or signature-based detection, often fail to identify novel, complex, or zero-day attacks. This creates a significant vulnerability in modern network environments. In response to these limitations, our project proposes an AI-Powered Network Intrusion Detection System (NIDS) that leverages the capabilities of both Machine Learning (ML) and Deep Learning (DL) models to perform accurate and scalable threat detection in real-time.

The system utilizes a multi-model architecture comprising classical ML classifiers—such as Decision Tree, Random Forest, Logistic Regression, and XGBoost—alongside DistilBERT, a transformer-based deep learning model, for textual and contextual analysis of network data. Feature extraction is performed on network packet flows, and flow-based attributes are combined into structured datasets. The data is then fed into respective models for classification into one of fifteen known intrusion types or as benign traffic. Additionally, DistilBERT enables payload-level analysis by converting flow information into textual representations, which allows for contextual understanding of packet behavior using NLP techniques.

To support real-time data processing, the system is implemented as a modular, containerized web application. Apache Kafka serves as the core message broker for streaming data between various components such as the packet producer, flow processor, and model consumers. Flask serves as the backend API layer, while a React.js-based frontend provides an interactive dashboard for users to visualize results. MongoDB and Firestore are employed for data logging and real-time storage of threat detection outcomes. The entire architecture is containerized using Docker to ensure scalability, maintainability, and platform independence.

Experimental evaluation conducted on the CICIDS2017 dataset demonstrates high classification accuracy across all models, with significantly reduced false positives and better generalization to unseen attacks. The system's modular design allows for easy integration of new models and datasets in the future. Overall, this AI-powered NIDS represents a significant step toward more intelligent, adaptive, and explainable intrusion detection, capable of responding to today's complex cyber threat landscape.

## Table of Contents

<b>Sr No.</b>	<b>Chapters</b>	<b>Page No.</b>
1	List of figures ,Tables,Symbols, Abbreviations or Nomenclature	1
2	Introduction	2
3	Literature Review	4
4	Work Done	8
5	Results and Discussions	19
6	Summary and Conclusions	30
7	References	31
8	Appendices	32

## **List of Figures And Tables**

### **Figures:**

- Fig 1: System Architecture
- Fig 2: Home
- Fig 3: Live Model Predictions
- Fig 4: Model Testing
- Fig 5: Live Component Logs
- Fig 6: NIDS Control API
- Fig 7: About Page

### **Tables:**

- Table 1: RandomForest Classification Report
- Table 2: DecisionTree Classification Report
- Table 3: LogisticRegression Classification Report
- Table 4: XGBoost Classification Report

### **List of Symbols, Abbreviations or Nomenclature:**

- LSTM: Long Short-Term Memory
- BERT: Bidirectional Encoder Representations from Transformers
- DistilBERT: Distilled BERT (smaller, faster BERT model)
- CICIDS2017: Canadian Institute for Cybersecurity Intrusion Detection System 2017 Dataset
- Docker: Containerization Platform
- Kafka: Distributed Event Streaming Platform
- XAI: Explainable Artificial Intelligence
- SHAP: SHapley Additive exPlanations
- LIME: Local Interpretable Model-agnostic Explanations
- NIDS: Network Intrusion Detection System
- DDoS: Distributed Denial of Service

# 1. Introduction

In today's digital era, cybersecurity has become a critical concern for organizations and individuals alike. The increasing frequency and sophistication of cyberattacks—such as Distributed Denial of Service (DDoS), malware, phishing, and brute-force login attempts—pose serious threats to network integrity and data privacy. As traditional Intrusion Detection Systems (IDS) struggle to keep pace with evolving attack patterns, there is an urgent need for intelligent, adaptive, and real-time detection solutions.

To address these challenges, we have developed an **AI-Powered Network Intrusion Detection System (NIDS)** that integrates classical Machine Learning (ML) models with state-of-the-art Deep Learning (DL) architectures. The system employs Decision Tree, Random Forest, Logistic Regression, and XGBoost for structured, flow-based classification, while the transformer-based DistilBERT model is used to analyze network traffic in a natural language-like format, enabling deeper contextual understanding of threats. The entire pipeline is designed to operate in real time, processing packet data as it is captured, classifying it, and visualizing results instantly on an interactive dashboard.

Our system processes live or uploaded network traffic using a modular streaming architecture. Components such as the **packet producer**, **processing consumer**, and **model consumers** communicate via Apache Kafka, ensuring high throughput and scalability. Captured traffic is analyzed to extract flow-based features, which are then passed through AI models for multi-class intrusion classification. Benign and malicious traffic is logged and stored using **MongoDB** and **Firestore**, supporting further analysis and alert generation. The web-based dashboard, built using **React** and **Flask**, enables users to visualize real-time results, monitor logs, and manage model pipelines with ease.

## 1.1 Key Features and Impact

- **Enhanced Network Security:** Real-time classification and detection of a wide range of cyberattacks significantly reduce the risk of data breaches and system compromise.
- **Multi-Model Intelligence:** Combining multiple ML and DL models increases accuracy, reduces false positives, and ensures better generalization across different attack types.
- **Research and Testing Platform:** The modular design enables researchers to integrate and benchmark their own models on the same pipeline.
- **Scalable & Cloud-Ready:** Built with Docker, Kafka, and cloud-compatible databases, the system can be deployed across distributed environments for enterprise use.



By integrating AI, streaming analytics, and modern web technologies, this project aims to provide a scalable and intelligent solution to one of today's most pressing cybersecurity challenges: the timely and accurate detection of network intrusions.

## **1.2 System Architecture and Data Flow**

The architecture follows a modular microservices pattern, allowing individual components to operate independently. Network packets are captured via tools like scapy, and flows are reconstructed using feature extraction tools. Apache Kafka serves as the central message broker, decoupling data producers and consumers and enabling asynchronous processing at scale. Machine learning models are containerized using Docker, which facilitates parallel model inference and resource isolation.

## **1.3 Model Training and Data Handling**

The machine learning pipeline supports both offline and incremental training. Feature normalization, dimensionality reduction, and class balancing are handled during preprocessing using libraries like Scikit-learn and Pandas. Model selection and hyperparameter tuning are performed through grid search and cross-validation. Labelled datasets are stored in MongoDB for training, while predictions and metadata are stored in Firestore for easy integration with the web frontend.

## **1.4 Web Interface and Monitoring**

The frontend dashboard is built with React and communicates with the Flask backend via RESTful APIs. It provides logs, visual summaries of network activity, traffic statistics, and model outputs in near real time. User roles and access control mechanisms are under development to support different levels of system interaction, including monitoring, configuration, and model updates.

## **2. Literature Review**

The integration of artificial intelligence into network intrusion detection systems (NIDS) has transformed cybersecurity by enabling automated, scalable, and real-time threat detection. Traditional intrusion detection methods based on rule-matching or signature databases often fail to detect sophisticated or zero-day attacks. Recent literature presents diverse AI-powered approaches ranging from supervised machine learning to deep learning, real-time deployment pipelines, and explainable models. This chapter explores key contributions, categorized by methodology, that informed the development of our AI-powered NIDS.

### **2.1 Supervised Learning-Based Intrusion Detection**

Supervised learning algorithms are widely used for classifying known attack types by learning from labeled datasets. Their simplicity, interpretability, and relatively low computational cost make them suitable for early-stage NIDS implementations.

#### **2.1.1.Improved ANN Model for Structured Detection**

An enhanced Artificial Neural Network (ANN) proposed in [1] achieved 92% accuracy on intrusion datasets by addressing class imbalance using oversampling and class weighting. The model's architecture was optimized for fast execution, making it suitable for real-time environments.

#### **2.1.2. Comparative Evaluation of Classical Models**

In [2], a comparative study on Decision Tree, Random Forest, and Logistic Regression showed that ensemble models like Random Forest offered higher recall and robustness, while Logistic Regression excelled in interpretability and low-resource deployment. A broader benchmark across datasets was discussed in [3], which emphasized selecting models based on dataset distribution and deployment constraints.

#### **2.1.3. Multi-Model Modular Architecture**

The pipeline presented in [4] utilized a combination of lightweight ML classifiers within a modular design. This allowed for rapid inference and scalability, especially when working with large datasets like CICIDS2017 and NSL-KDD. The flexibility of swapping or combining models supported rapid experimentation.

## **2.2 Deep Learning Architectures for Flow and Payload Analysis**

Deep learning models provide enhanced performance in modeling complex, temporal, or high-dimensional patterns in network traffic. They are particularly suited for detecting stealthy or slowly evolving attacks.

### **2.2.1. CNN-LSTM-Based Sequential Detection**

The hybrid architecture in [5] combined Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) units to detect flow-level anomalies. LSTM's ability to preserve long-term dependencies made it effective for slow attacks like DoS-Slowloris and Heartbleed.

### **2.2.2. NLP-Based Transformer Models (DistilBERT)**

In [6], the transformer-based DistilBERT model was trained to treat flow features as text sequences. This NLP-style encoding allowed the model to learn contextual relationships across features, enabling accurate multi-class intrusion detection. Its generalization to unseen attack types demonstrated the strength of transformer architectures in IDS.

### **2.2.3. Unsupervised Detection using Autoencoders and GANs**

The use of Autoencoders and Generative Adversarial Networks (GANs) in [7] allowed the model to learn normal traffic patterns and detect deviations. GANs were also used to generate synthetic attacks to improve the training set's diversity, making the IDS more robust to edge cases

## **2.3 Real-Time Intrusion Detection with Scalable Architectures**

Real-time NIDS systems must process live data streams, scale horizontally, and maintain low latency. Researchers have focused on event-streaming platforms, containerized deployment, and cloud integration to meet these requirements.

### **2.3.1. Apache Kafka for Streaming Pipelines**

The system described in [8] utilized Apache Kafka as a backbone for real-time packet streaming. It allowed individual modules such as packet producers, feature processors, and model consumers to function asynchronously, ensuring high throughput with minimal data loss

### **2.3.2. Docker-Based Deployment Strategies**

In [9], containerization using Docker enabled modular deployment of ML models as independent services. This approach allowed developers to manage load balancing, perform model upgrades, and scale horizontally across nodes. REST APIs were used to update dashboards in real time.

### **2.3.3. Hybrid Architectures for Cloud and Edge Environments**

The hybrid IDS architecture proposed in [10] combined high-accuracy deep learning models for batch processing with lightweight ML models for edge inference. This dual approach achieved both accuracy and latency efficiency, making it suitable for deployment in enterprise or IoT environments.

## **2.4 Explainable and Interpretable Intrusion Detection Systems**

With growing model complexity, explainability has become essential in IDS for regulatory compliance, analyst trust, and operational transparency.

### **2.4.1. SHAP and LIME for Model Interpretability**

The work in [11] introduced SHAP (Shapley Additive Explanations) and LIME (Local Interpretable Model-Agnostic Explanations) into deep learning-based IDS. These tools enabled analysts to visualize how specific features—like destination port, packet size, or protocol flags—influenced the model’s decision, making the detection process auditable.

### **2.4.2 Role of Explainable IDS (X-IDS) in Security Operations**

Explainable IDS frameworks helped security analysts identify biases, false positives, and feature importance. The study emphasized that XAI bridges the gap between automated detection and human decision-making, especially in high-risk environments like government, finance, or healthcare.

The AI NIDS project incorporates these research-driven innovations by combining supervised learning models for efficient traffic classification and deep learning architectures like DistilBERT for enhanced contextual analysis of network behavior. Real-time detection is enabled through Kafka-based streaming pipelines, while Docker containerization ensures scalable and modular deployment. Additionally, explainable AI techniques improve model transparency, allowing cybersecurity professionals to understand and trust the system’s decisions. These

technologies collectively strengthen the ability of AI NIDS to proactively detect and mitigate a wide range of cyber threats in real-world environment

### 3. Work Done

This section describes the detailed methodology and components implemented in the AI-Powered Network Intrusion Detection System (AI NIDS) project. The system integrates multiple machine learning and deep learning models, a real-time streaming pipeline, and an interactive web dashboard to achieve accurate and scalable intrusion detection.

The AI NIDS architecture is designed for modularity, scalability, and real-time performance. It consists of four major components: Frontend, Backend, Machine Learning Models, and Data Streaming & Storage. Each component is described in detail below :

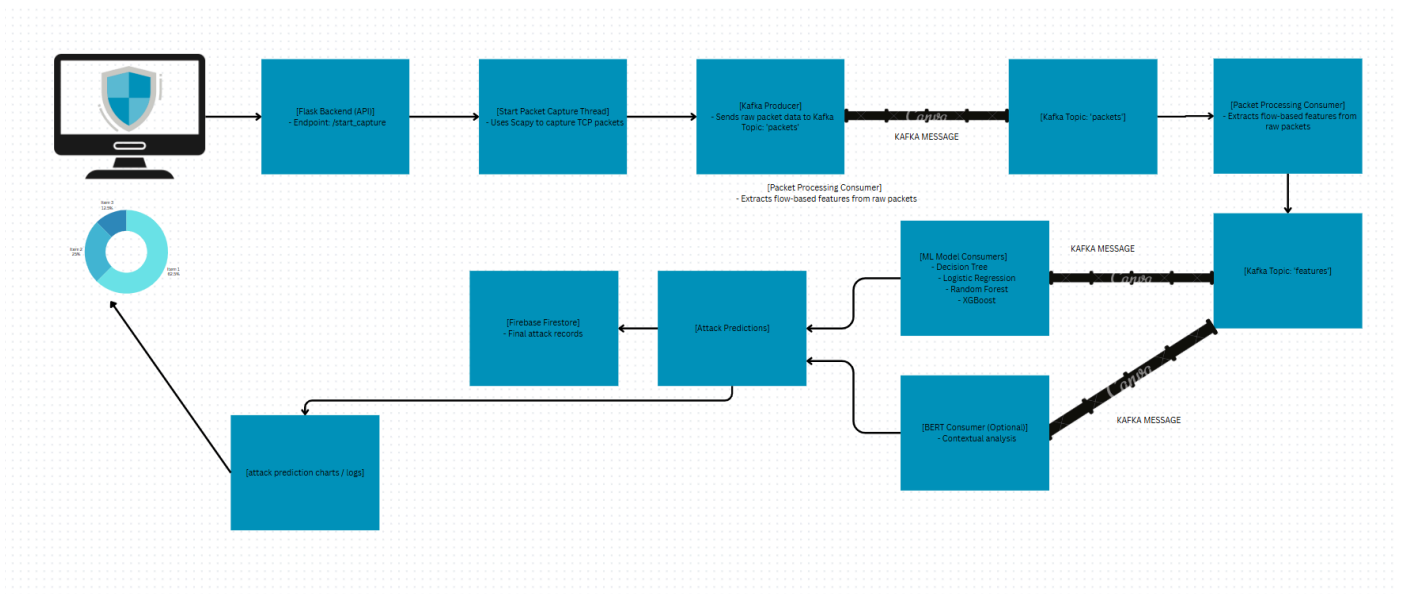


Fig 1. System Architecture

#### 3.1 Backend

The backend system is a Flask-based server application developed for real-time network intrusion detection using machine learning and NLP models. It leverages multiple technologies such as Kafka for message queuing, MongoDB and Firebase Firestore for storage, and integrates pre-trained models to perform threat classification.

##### 3.1.1. Technology Stack:

Programming Language : Python

## Frameworks & Libraries:

- Flask (web server)
- Scapy (packet sniffing)
- Kafka (Apache Kafka for stream processing)
- MongoDB (NoSQL database)
- Firebase Firestore (NoSQL cloud database)
- PyTorch + HuggingFace Transformers (BERT model for text classification)
- Scikit-learn, XGBoost, Joblib (for ML model loading and prediction)
- Matplotlib (for dynamic chart generation)
- Flask-CORS (for cross-origin access support)

### 3.1.2. System Components:

#### A. Packet Producer

- Captures live TCP packets using Scapy.
- Extracts metadata such as source/destination IPs, ports, protocol, and packet size.
- Pushes packet data to Kafka topic packets.

#### B. Processing Consumer

- Consumes TCP packet data from Kafka.
- Aggregates packets into network flows.
- Extracts flow-based statistical features such as duration, inter-arrival time (IAT), packet size metrics.
- Publishes processed flow features to Kafka for further ML inference.

#### C. ML Consumers

- Loads multiple pre-trained ML models:
  - Decision Tree
  - Logistic Regression
  - Random Forest
  - XGBoost
- Receives feature vectors from Kafka and performs threat prediction.
- Stores predictions in MongoDB and optionally logs threats to Firebase Firestore.

## **D. BERT Consumer**

- Loads a fine-tuned DistilBERT model for binary classification of textual log data.
- Tokenizes and processes network payload or metadata text using DistilBertTokenizer.
- Performs inference and logs predictions with timestamps.

### **3.1.3. Data Storage**

#### **MongoDB Collections:**

- raw\_packets: Original packet data.
- flow\_features: Extracted flow-based feature vectors.
- predictions: ML model predictions.

#### **Firebase Firestore**

- Stores cleaned threat prediction data for real-time syncing with front-end dashboards.

### **3.1.4. Model Management**

- Models are stored as .joblib files for ML models and PyTorch format for BERT.
- Loaded during backend initialization.
- Stored at structured paths under a predefined models/ directory.

### **3.1.5. Chart Generation**

- Generates pie charts representing threat classification distribution.
- Uses Matplotlib with a circular donut-style chart and custom labeling.
- Charts are rendered in-memory using io.BytesIO and returned as PNGs.

### **3.1.6. RESTful API and CORS**

- Flask app is enabled with CORS to allow frontend interactions.
- APIs support data retrieval, control signal updates, and dynamic chart rendering.

### **3.1.7. Logging and Monitoring**

- In-memory logs are maintained for each system component.
- Each log entry is timestamped and limited to a maximum of 200 entries to manage memory usage.



### 3.1.8. Configuration

- Kafka:localhost:9092 (external)
- host.docker.internal:9092 (Docker internal bridge)
- MongoDB URI: mongodb://localhost:27017
- Firebase: Uses a service account JSON to authenticate.

### 3.1.9. Feature Engineering

Extracted flow features include:

- Packet counts and lengths
- Flow duration and rates
- Inter-arrival times
- TCP flag counts (e.g., SYN, ACK, FIN)

These are mapped into a consistent feature order expected by the machine learning models.

## 3.2 Frontend:

The frontend is a React.js web application that provides a user-friendly interface for interacting with a backend-powered AI-based intrusion detection system. It supports both live monitoring and offline testing modes, making it suitable for real-time analysis as well as experimental evaluation.

### 3.2.1. Features and Functional Flow

#### A . Landing Page (Home.js):

Displays two main options:

- Go Live
- Test

Acts as the central navigation hub for the application.

#### B. Go Live (GoLive.js)

This section is used for real-time intrusion detection.

Functionalities:

- Start/Stop Pipeline Provides buttons to start or stop the real-time packet capture and analysis pipeline by

calling backend APIs.

- Live Donut Charts: Displays donut charts for each ML model used in the backend (e.g., Random Forest, XGBoost, etc.). Each chart visualizes prediction confidence or class distribution in real time.
- Live Logs Link:
- A link to the Logs Page, which displays backend logs streamed live (e.g., from Firebase or MongoDB).
- Allows users to monitor model predictions, packet logs, or BERT classifications.

### **C. Logs Page (LogsPage.js)**

Displays live logs fetched from a backend database.

Useful for:

- Reviewing classified packets.
- Monitoring which model predicted what.
- System auditing and debugging.

### **D. Test (Test.js)**

This section enables offline testing of the models using uploaded CSV data.

Functionalities:

- CSV Upload : Users can upload a CSV file containing the same feature columns expected by the ML models.
- Backend API Integration: Upon upload, the frontend sends the CSV data to the backend via an API call.
- Backend processes the data and returns: Model-wise predictions ,Evaluation metrics (accuracy, precision, recall, etc.) and Donut Charts and Metrics
- The frontend displays: Donut charts to visualize class-wise prediction distribution, Evaluation metrics for each model on the uploaded dataset.

## **3.3 ML Model Development:**

### **3.3.1. Dataset and Preprocessing:**

- Dataset Used: CICIDS2017, a widely used intrusion detection dataset that includes various attack scenarios and normal traffic.
- Source: Kaggle.
- Data Consolidation: Multiple .csv files were merged into a single DataFrame for consistency and ease of

processing.

### **3.3.2. Cleaning:**

- Removed rows with missing values.
- Mapped and standardized class labels. For example: Attacks like DoS Hulk, DoS GoldenEye, and Heartbleed were mapped to a common class like DoS.
- BENIGN traffic was separated clearly as the normal class.
- Encoding: Label encoding was used for categorical values, especially for class labels.
- Scaling: Feature values were normalized using Min-Max scaling to ensure uniformity for model training.

### **3.3.3. Feature Selection:**

Initial Approach:

- Removed low-variance features which contribute little to model performance.
- Removed features with high correlation to avoid redundancy and multicollinearity.

Final Selection Criteria:

- Choose features with high importance scores using models like Random Forest and XGBoost.
- Ensured selected features had domain relevance, particularly for network traffic characteristics.

### **3.3.4 Chosen Features:**

#### **1. Destination Port**

Definition: The port number on the destination machine that is receiving the packet.

Relevance: Certain ports (e.g., 22 for SSH, 80 for HTTP) are commonly used and can be targets for specific types of attacks. Unusual or high-numbered ports may indicate suspicious activity.

#### **2. Flow Duration**

Definition: The total time duration of a flow from the first packet to the last packet.

Relevance: Abnormally short or long flows may indicate attacks such as scanning, flooding, or long-lived connections used in DoS attacks.

#### **3. Total Fwd Packets**

Definition: Total number of packets sent in the forward direction (from source to destination).

Relevance: Helps distinguish between normal and abnormal traffic. For example, DDoS attacks often show a high number of forward packets with little or no response.

#### **4. Total Length of Fwd Packets**

Definition: Sum of the sizes of all packets sent in the forward direction.

Relevance: High cumulative length in a short time may indicate flooding-type attacks.

#### **5. Flow Bytes/s**

Definition: The number of bytes transferred per second during the flow.

Relevance: High throughput might suggest data exfiltration or DoS attacks. Low values could indicate reconnaissance.

#### **6. Flow Packets/s**

Definition: Number of packets transmitted per second.

Relevance: High packet rates may indicate scanning or flooding behavior, particularly with uniform packet sizes.

#### **7. Flow IAT Mean**

Definition: Mean time between packets in a flow (Inter Arrival Time).

Relevance: Low values (packets arriving very fast) could indicate automated scripts or bot activity.

#### **8. Flow IAT Std**

Definition: Standard deviation of the inter-arrival time.

Relevance: Consistently regular timing may indicate scripted or bot activity, whereas erratic patterns may suggest user interaction or dynamic applications.

#### **9. Fwd Packet Length Max**

Definition: Maximum length of a forward direction packet.

Relevance: Very large packets could suggest attempts to overflow buffers or send large payloads.

#### **10. Fwd Packet Length Min**

Definition: Minimum length of a forward packet.

Relevance: Repeated small packet sizes may indicate scanning or brute-force attempts.

### **11. Fwd Packet Length Mean**

Definition: Average length of all packets in the forward direction.

Relevance: Useful in profiling application behavior; anomalies could signify malicious traffic.

### **12. Fwd Packet Length Std**

Definition: Standard deviation of forward packet lengths.

Relevance: High variability may indicate multiple stages or mixed behaviors in a session.

### **13. Min Packet Length**

Definition: Smallest packet length in the flow.

Relevance: Very small packets can be associated with SYN scans or ICMP ping floods.

### **14. Max Packet Length**

Definition: Largest packet length in the flow.

Relevance: Large packets may indicate file transfers or potentially malicious payloads.

### **15. Packet Length Mean**

Definition: Mean packet size over the flow.

Relevance: Average size helps differentiate between protocols (e.g., HTTP vs. ICMP) and usage types.

### **16. Packet Length Std**

Definition: Standard deviation of packet lengths.

Relevance: Identifies variability in packet sizes, which is often low for automated attacks.

### **17. Packet Length Variance**

Definition: Variance in the lengths of packets.

Relevance: High variance may suggest multipurpose sessions or attacks masking as normal traffic.

### **18. FIN Flag Count**

Definition: Number of FIN (Finish) flags in packets within the flow.

Relevance: Used to gracefully terminate TCP connections. Anomalous use can be a sign of FIN scan or stealth

attacks.

### **19. SYN Flag Count**

Definition: Number of SYN (Synchronize) flags in the flow.

Relevance: High SYN count may indicate SYN flood attacks, which are used to exhaust server resources.

### **20. ACK Flag Count**

Definition: Number of ACK (Acknowledgment) flags in the flow.

Relevance: ACK flags confirm receipt of packets. Abnormal patterns may reveal session hijacking or spoofed traffic.

### **3.3.5. Model Selection and Comparison**

Several machine learning models were evaluated to identify the most effective one for the classification task:

- Logistic Regression: A linear model often used as a baseline in classification problems. It is efficient, interpretable, and suitable for linearly separable data.
- Decision Tree: A non-linear model that splits the data based on feature thresholds. While interpretable and fast, it can overfit on small or noisy datasets.
- Random Forest: An ensemble method that builds multiple decision trees and aggregates their outputs to improve generalization and reduce overfitting.
- XGBoost (Extreme Gradient Boosting): A powerful boosting algorithm that builds trees sequentially, focusing on minimizing errors from previous iterations. Known for its high performance and scalability, it often outperforms other models in structured data tasks.

### **3.3.6. Data Splitting with Stratified Sampling**

The dataset was split into training and testing subsets while maintaining the original class distribution through stratified sampling. This ensured that both sets had a representative proportion of each class, which is critical in cases of class imbalance.

### **3.3.7. Cross-Validation**

To ensure robustness and reduce variance in performance evaluation, k-fold cross-validation was employed during training. This involved dividing the training data into k subsets, training the model on k-1 folds, and validating it on the remaining fold, repeating this process k times. The average performance across all folds was considered for

model comparison.

### **3.3.8. Evaluation Metrics**

A comprehensive set of metrics was used to assess and compare model performance:

- Accuracy: Proportion of correct predictions over total predictions
- Precision: Proportion of true positive predictions out of all positive predictions made.
- Recall (Sensitivity): Proportion of true positives identified out of all actual positives.
- F1-Score: Harmonic means of precision and recall, especially useful in cases of class imbalance.
- ROC AUC Score: Measures the model's ability to distinguish between classes, providing insight into performance across different thresholds.

These metrics ensured a balanced evaluation, especially when handling imbalanced datasets.

## **3.4 DistilBert Training**

### **3.4.1. Data Preparation**

The process began with a balanced pandas DataFrame containing two columns: text, which held raw network traffic records, and binary\_label, where 0 indicated benign and 1 indicated malicious traffic. This DataFrame was then converted into a Hugging Face Dataset object, and the binary\_label column was renamed to labels to maintain compatibility with the Transformers API.

### **3.4.2. Tokenizer and Tokenization**

The DistilBERT tokenizer was loaded using `AutoTokenizer.from_pretrained("distilbert-base-uncased")`. A `tokenize_function` was defined to tokenize each text record, applying padding and truncation to ensure a maximum sequence length of 128 tokens. This function was mapped over the entire dataset in batches. The original text column was removed, and the dataset was reformatted to PyTorch tensors.

### **3.4.3. Train/Eval Split**

The tokenized dataset was divided into training and evaluation sets using a 90% to 10% ratio. A fixed random seed (42) was used during this split to ensure that the results are reproducible across different runs.

### **3.4.4. Model Initialization**

A pre-trained DistilBERT model was loaded using `AutoModelForSequenceClassification` and configured for binary classification by setting `num_labels=2`. This allowed the model to predict whether a given network traffic sample

was benign or malicious.

#### **3.4.5. Training Configuration**

Training arguments were defined using the `TrainingArguments` class. Key settings included the output directory (`./distilbert_binary`), evaluation and saving strategy set to epoch, learning rate of  $2e-5$ , a batch size of 16 for both training and evaluation, 3 epochs, and 0.01 weight decay. Logging was configured every 50 steps, and the model was set to reload the best version at the end based on evaluation accuracy.

#### **3.4.6. Trainer Setup and Metric**

The accuracy metric was loaded using `evaluate.load("accuracy")`, and a `Trainer` object was created with the model, training arguments, training and evaluation datasets, and a `compute_metrics` function that returned accuracy during training. This setup ensured real-time evaluation of model performance during training.

#### **3.4.7. Training and Evaluation**

The model was trained over three epochs, with evaluation and model checkpointing happening at the end of each epoch. After training, the model with the highest evaluation accuracy was automatically reloaded for final use.

#### **3.4.8. Model Saving**

Upon completion, both the fine-tuned model and tokenizer were saved to the `./distilbert_binary_model` directory. This folder was then compressed into a ZIP archive to facilitate download and future deployment.



## 4. Results and Discussions

This chapter presents the evaluation outcomes of the AI NIDS system, based on experiments conducted using the CICIDS2017 dataset. Both classical machine learning models and the deep learning-based DistilBERT model were assessed in terms of accuracy, precision, recall, F1-score, and interpretability. The following are the tables displaying our results.

### 4.1 RandomForest Classification Report

Class	Support	Precision	Recall	F1-Score
BENIGN	454779	1.00	1.00	1.00
Bot	366	0.60	0.59	0.59
DDoS	25643	1.00	1.00	1.00
DoS-GoldenEye	2022	0.97	0.98	0.98
DoS-Hulk	46189	0.99	0.99	0.99
DoS-Slowhttp test	1086	0.99	0.98	0.99
DoS-slowloris	1139	0.99	0.99	0.99
FTP-Patator	1596	1.00	1.00	1.00
Infiltration	6	1.00	0.83	0.91

PortScan	31740	0.99	1.00	1.00
SSH-Patator	1149	0.98	0.98	0.98
Web-Attack-B ruteForce	310	0.71	0.67	0.69
Web-Attack-S qlInjection	3	0.00	0.00	0.00
Web-Attack-X SS	121	0.37	0.40	0.38

**Table 1. RandomForest Classification Report**

The Random Forest model shows excellent performance with near-perfect scores across most classes, especially for dominant ones like BENIGN, DDoS, and DoS-Hulk. Minor classes such as Web-Attack-SqlInjection and Web-Attack-XSS exhibit poor recall and precision, likely due to class imbalance. Overall, this model demonstrates high reliability in detecting major attack types with a macro F1-score of 0.82.

## 4.2 DecisionTree Classification Report

Class	Support	Precision	Recall	F1-Score
BENIGN	454779	1.00	1.00	1.00
Bot	366	0.58	0.61	0.59
DDoS	25643	1.00	1.00	1.00

DoS-GoldenEye	2022	0.97	0.98	0.97
DoS-Hulk	46189	0.99	0.99	0.99
DoS-Slowhttp test	1086	0.98	0.99	0.99
DoS-slowloris	1139	0.99	0.99	0.99
FTP-Patator	1596	1.00	1.00	1.00
Heartbleed	0	0.00	0.00	0.00
Infiltration	6	1.00	0.67	0.80
PortScan	31740	0.99	0.99	0.99
SSH-Patator	1149	0.98	0.98	0.98
Web-Attack-BruteForce	310	0.70	0.63	0.66
Web-Attack-SqlInjection	3	0.17	0.33	0.22
Web-Attack-XSS	121	0.35	0.40	0.37

**Table 2. DecisionTree Classification Report**

The Decision Tree performs similarly to Random Forest on major classes but slightly lags in handling minority attack types. For instance, Heartbleed was never detected due to the absence of samples in the test set. While precision and recall are high for most classes, the model struggles with rare categories, lowering its macro F1-score to **0.77**. The weighted average remains perfect due to dominance by benign traffic.

### 4.3 LogisticRegression Classification Report

Class	Support	Precision	Recall	F1-Score
BENIGN	454779	0.89	0.95	0.92
Bot	366	0.00	0.00	0.00
DDoS	25643	0.51	0.50	0.51
DoS-GoldenEye	2022	0.00	0.00	0.00
DoS-Hulk	46189	0.70	0.82	0.76
DoS-Slowhttp test	1086	0.00	0.00	0.00
DoS-slowloris	1139	0.00	0.00	0.00
FTP-Patator	1596	0.00	0.00	0.00
Infiltration	6	0.00	0.00	0.00
PortScan	31740	0.00	0.00	0.00

SSH-Patator	1149	0.00	0.00	0.00
Web-Attack-B ruteForce	310	0.00	0.00	0.00
Web-Attack-S qlInjection	3	0.00	0.00	0.00
Web-Attack-X SS	121	0.00	0.00	0.00

**Table 3. LogisticRegression Classification Report**

Logistic Regression performs poorly overall, especially for almost all attack types except BENIGN and DoS-Hulk. Many classes show zero precision and recall, indicating that the model fails to identify these correctly. The macro F1-score is just 0.16, making it unsuitable for multi-class intrusion detection without significant class balancing or model enhancement.

#### 4.4 XGBoost Classification Report

Class	Support	Precision	Recall	F1-Score
BENIGN	454779	1.00	1.00	1.00
Bot	366	1.00	0.40	0.57
DDoS	25643	1.00	1.00	1.00
DoS-GoldenE ye	2022	0.96	0.97	0.97

DoS-Hulk	46189	0.99	0.99	0.99
DoS-Slowhttp test	1086	0.99	0.99	0.99
DoS-slowloris	1139	0.99	0.99	0.99
FTP-Patator	1596	1.00	1.00	1.00
Infiltration	6	0.83	0.83	0.83
PortScan	31740	0.99	1.00	1.00
SSH-Patator	1149	0.98	0.99	0.98
Web-Attack-B ruteForce	310	0.71	0.19	0.30
Web-Attack-S qlInjection	3	0.50	0.67	0.57
Web-Attack-X SS	121	0.67	0.02	0.03

**Table 4. XGBoost Classification Report**

XGBoost shows strong performance, closely rivaling Random Forest, with perfect or near-perfect metrics for major classes. However, it performs inconsistently for minority classes like Web-Attack-XSS, BruteForce, and SqlInjection, where precision and recall fluctuate. With a macro F1-score of 0.80, it's a powerful and scalable model, but it still faces issues with underrepresented classes.

## 4.5 DistilBERT Model Evaluation

DistilBERT achieved a **high accuracy of 96%**, showing strong overall performance in classifying network intrusion types. Its **F1-score of 0.96** indicates a good balance between precision (**94.02%**) and recall (**99.09%**), suggesting the model rarely misses attacks and keeps false positives low. The evaluation was efficient, with low latency (**~0.79s**) and high throughput (**254 samples/sec**), making it suitable for near real-time intrusion detection.

Here are the DistilBERT model evaluation metrics :

- Evaluation Loss: 0.1357
- Evaluation Accuracy: 96%
- Evaluation Precision: 94.02%
- Evaluation Recall: 99.09%
- Evaluation F1-Score: 96.49%
- Model Preparation Time: 0.0033 seconds
- Evaluation Runtime: 0.7862 seconds
- Evaluation Samples per Second: 254.40
- Evaluation Steps per Second: 16.54

From an operational standpoint, the **evaluation loss of 0.1357** suggests that the model has been well-trained and is not overfitting to the training data. The evaluation was conducted efficiently, with a **runtime of just 0.7862 seconds**, and a **model preparation time of 0.0033 seconds**, indicating that the model is lightweight and suitable for real-time or near real-time applications.

In terms of throughput, DistilBERT processed data at a rate of **254.40 samples per second**, with an evaluation speed of **16.54 steps per second**, demonstrating its capacity to handle high-volume traffic data in a timely manner. These characteristics make it a strong candidate for deployment in production-level **Intrusion Detection Systems (IDS)**, particularly in environments where rapid threat detection and minimal latency are essential, such as enterprise networks or critical infrastructure.

## 4.6 WebApp Interface:

### 4.6.1. Home

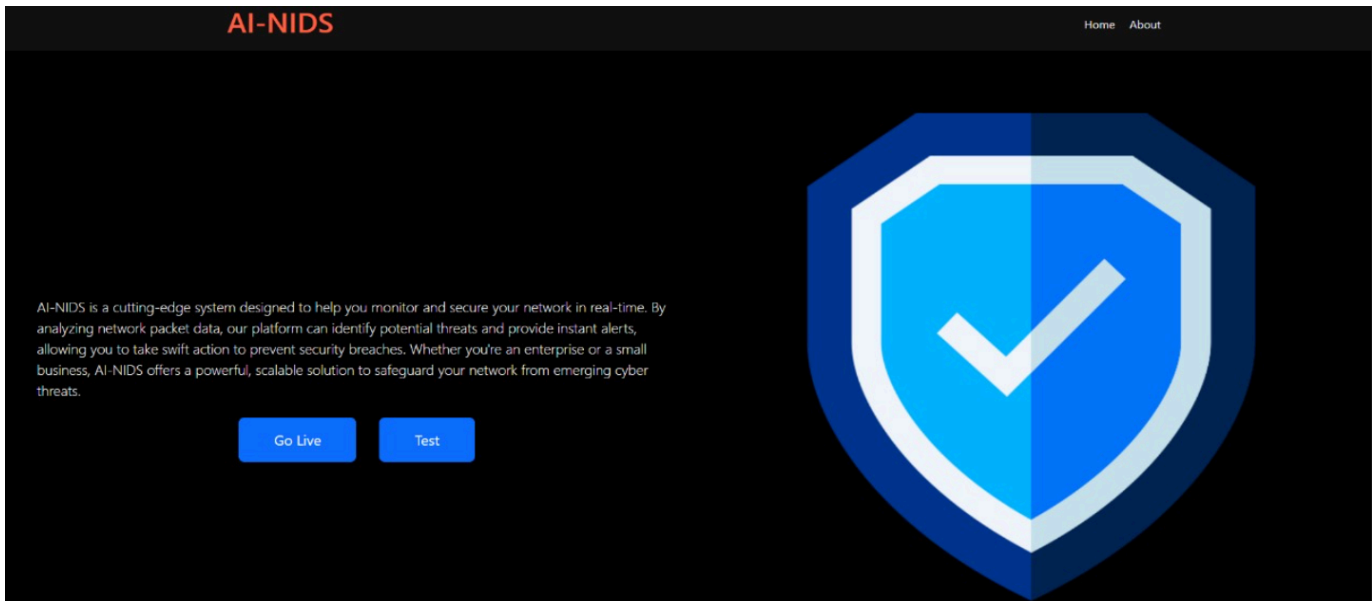


Fig 2. Home

### 4.6.2. Live Model Predictions

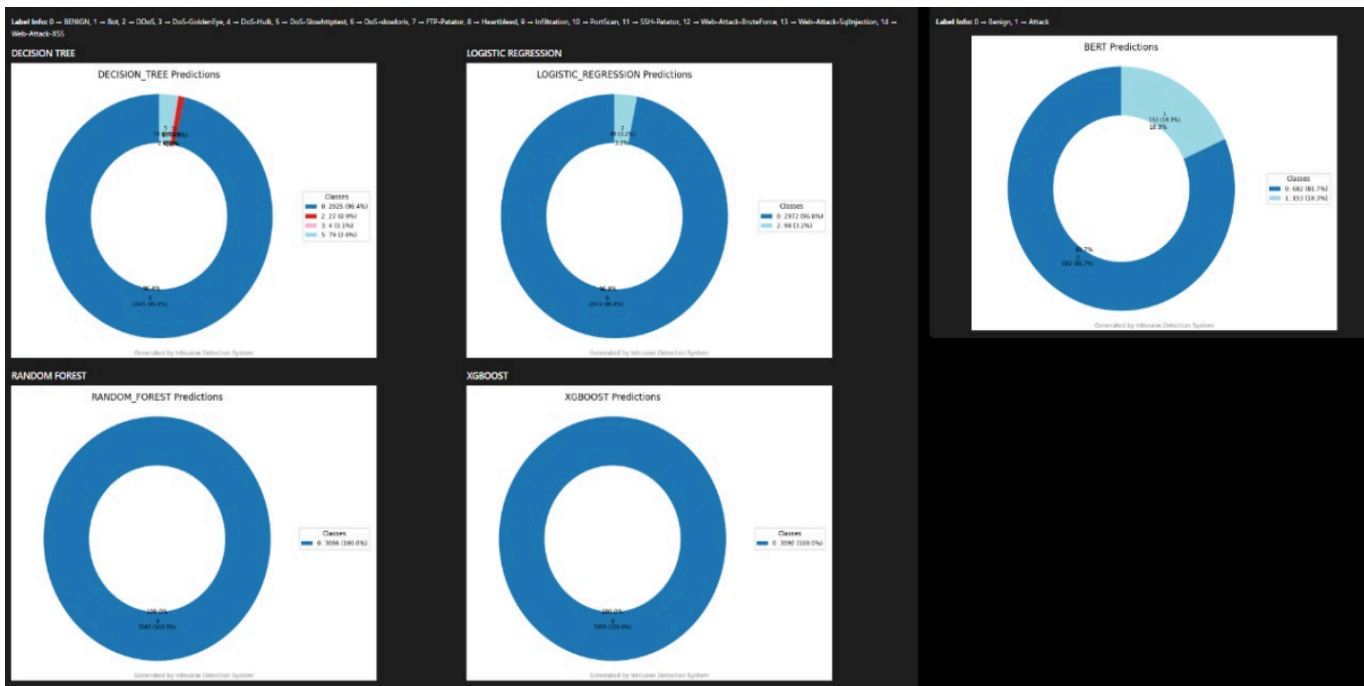


Fig 3. Live Model Prediction



### 4.6.3. Model Testing

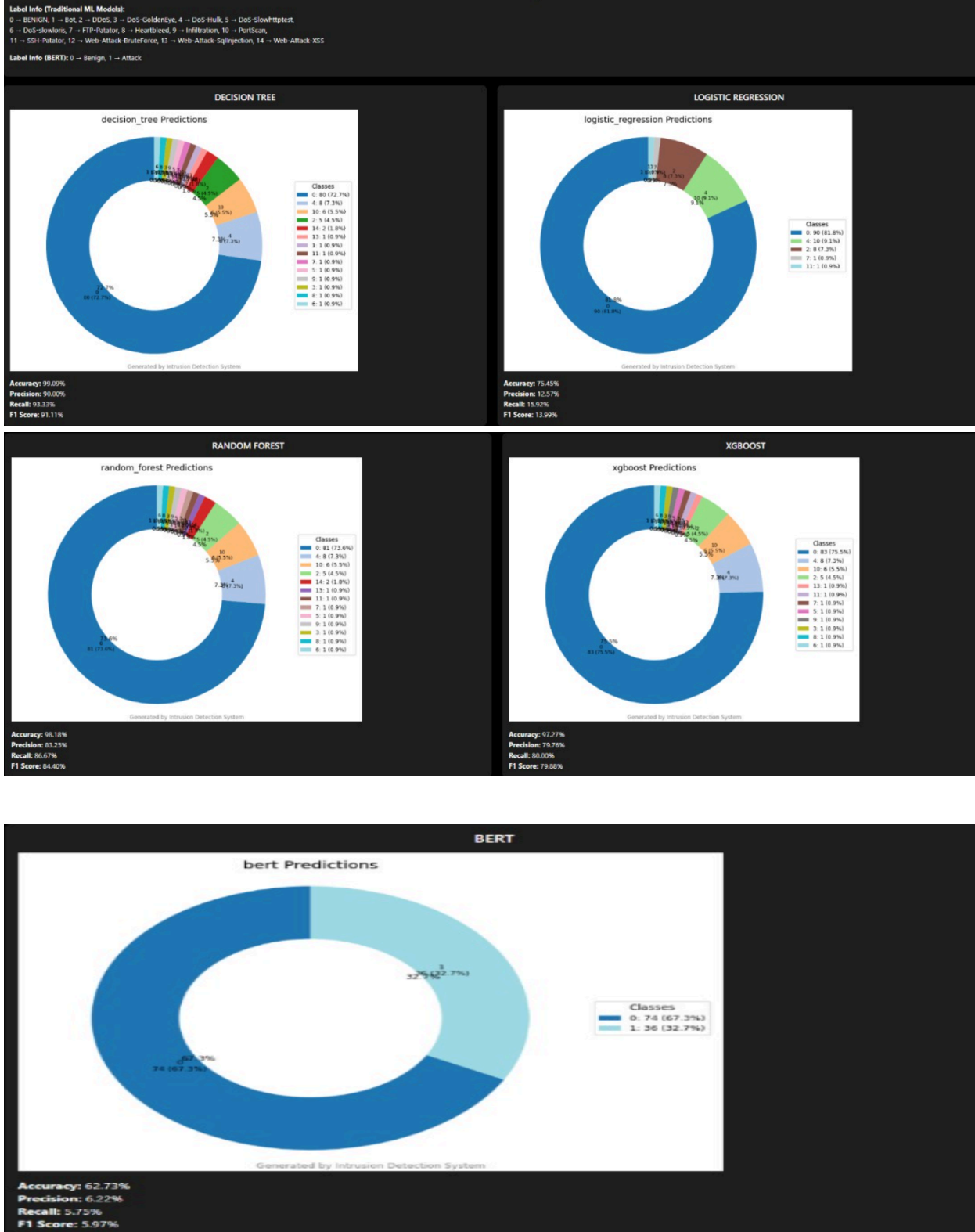


Fig 4 . Model Testing

4.6.4 Live Components Logs

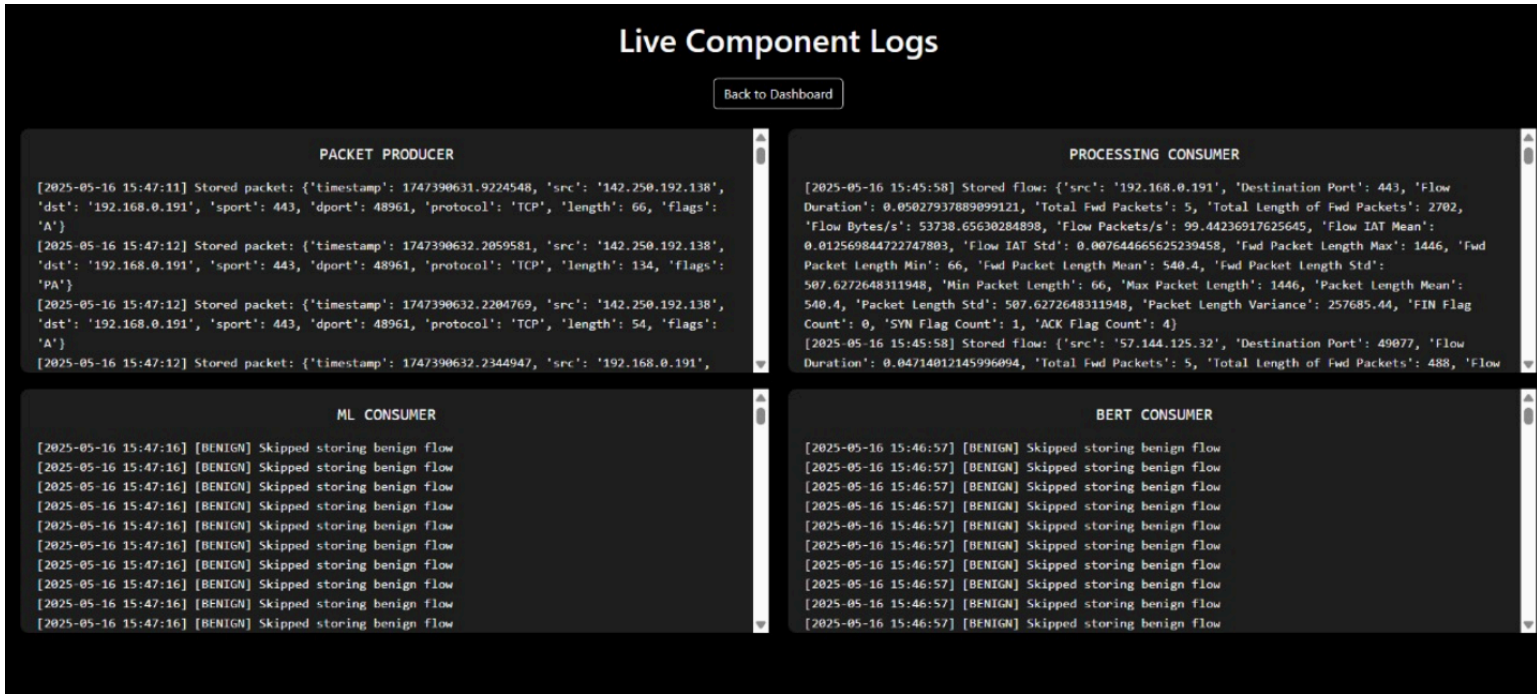


Fig 5. Live Component logs

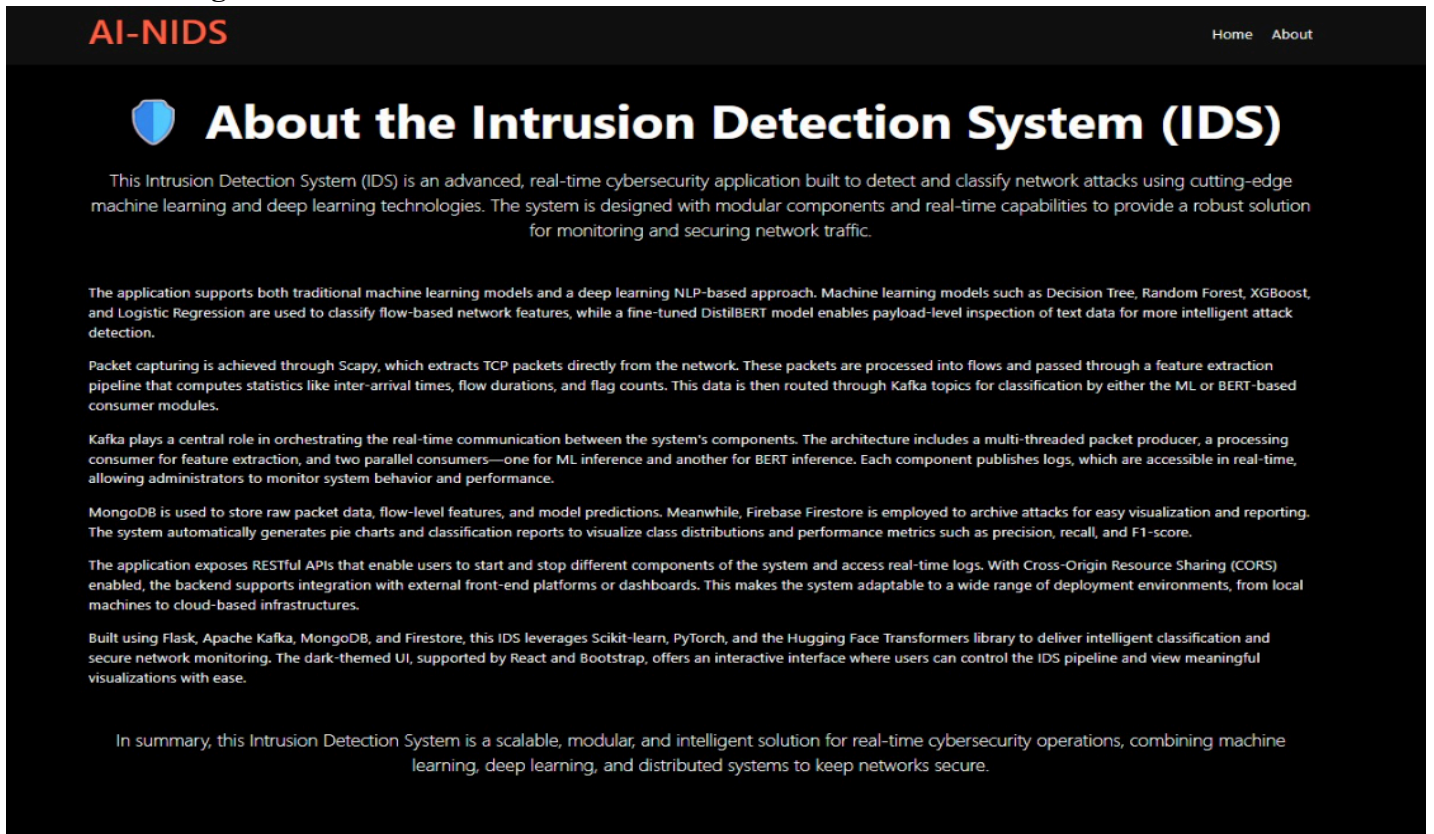
4.6.5 NIDS Control API:

NIDS Control API

- [test](#)
- [Logs - Packet Producer](#)
- [Logs - Processing Consumer](#)
- [Logs - ML Consumer](#)
- [Logs - BERT Consumer](#)
- [insert](#)
- [API Stats](#)
- [Upload CSV](#)

Fig 6. NIDS Control API

#### 4.6.6. About Page:



The screenshot displays the 'About' page of the AI-NIDS application. The page has a dark theme with a black background and white text. At the top, there is a navigation bar with 'AI-NIDS' in orange and 'Home About' in white. The main heading is 'About the Intrusion Detection System (IDS)' in large white font, preceded by a blue shield icon. Below the heading, there are several paragraphs of text describing the system's capabilities, architecture, and technology stack. The text is organized into sections with varying indentations. The overall layout is clean and professional, typical of a technical documentation page.

**AI-NIDS** Home About

## About the Intrusion Detection System (IDS)

This Intrusion Detection System (IDS) is an advanced, real-time cybersecurity application built to detect and classify network attacks using cutting-edge machine learning and deep learning technologies. The system is designed with modular components and real-time capabilities to provide a robust solution for monitoring and securing network traffic.

The application supports both traditional machine learning models and a deep learning NLP-based approach. Machine learning models such as Decision Tree, Random Forest, XGBoost, and Logistic Regression are used to classify flow-based network features, while a fine-tuned DistilBERT model enables payload-level inspection of text data for more intelligent attack detection.

Packet capturing is achieved through Scapy, which extracts TCP packets directly from the network. These packets are processed into flows and passed through a feature extraction pipeline that computes statistics like inter-arrival times, flow durations, and flag counts. This data is then routed through Kafka topics for classification by either the ML or BERT-based consumer modules.

Kafka plays a central role in orchestrating the real-time communication between the system's components. The architecture includes a multi-threaded packet producer, a processing consumer for feature extraction, and two parallel consumers—one for ML inference and another for BERT inference. Each component publishes logs, which are accessible in real-time, allowing administrators to monitor system behavior and performance.

MongoDB is used to store raw packet data, flow-level features, and model predictions. Meanwhile, Firebase Firestore is employed to archive attacks for easy visualization and reporting. The system automatically generates pie charts and classification reports to visualize class distributions and performance metrics such as precision, recall, and F1-score.

The application exposes RESTful APIs that enable users to start and stop different components of the system and access real-time logs. With Cross-Origin Resource Sharing (CORS) enabled, the backend supports integration with external front-end platforms or dashboards. This makes the system adaptable to a wide range of deployment environments, from local machines to cloud-based infrastructures.

Built using Flask, Apache Kafka, MongoDB, and Firestore, this IDS leverages Scikit-learn, PyTorch, and the Hugging Face Transformers library to deliver intelligent classification and secure network monitoring. The dark-themed UI, supported by React and Bootstrap, offers an interactive interface where users can control the IDS pipeline and view meaningful visualizations with ease.

In summary, this Intrusion Detection System is a scalable, modular, and intelligent solution for real-time cybersecurity operations, combining machine learning, deep learning, and distributed systems to keep networks secure.

Fig 7. About Page

## **5. Summary and conclusions**

### **5.1 Summary**

The AI-NIDS web application, developed using a React frontend and Flask backend, integrates multiple machine learning models to detect various network intrusion types. Random Forest and XGBoost models delivered strong performance, especially in identifying dominant attack classes such as BENIGN, DDoS, and DoS-Hulk, with macro F1-scores above 0.80. The Decision Tree model showed reasonable accuracy but had difficulty detecting rare attack types. Logistic Regression was less effective, showing poor precision and recall for most attacks. The DistilBERT model, a lightweight transformer-based approach, demonstrated excellent accuracy (96%) and a balanced precision-recall trade-off, making it suitable for near real-time intrusion detection.

### **5.2 Conclusion**

Overall, the AI-NIDS system offers a robust and scalable solution for network security monitoring by leveraging a combination of traditional machine learning and advanced NLP models. Its high accuracy and efficient runtime performance make it well-suited for practical deployment. However, the models currently face challenges with class imbalance, particularly in detecting rare or minority attack types. Future work should focus on improving minority class detection through data augmentation or specialized algorithms and optimizing model latency for faster inference. This will further enhance the system's reliability and applicability in real-world cybersecurity environments.

## 6. References

1. Rathore, S., et al., "Enhancing Cyber Threat Detection with an Improved Artificial Neural Network Model," *Computers & Security*, 2023.
2. Raval, M., et al., "AI-Enabled Threat Detection: Leveraging Artificial Intelligence for Advanced Security and Cyber Threat Mitigation," *ACM Computing Surveys*, 2024.
3. Dhanabal, L., & Shantharajah, S. P., "Artificial Intelligence-Based Intrusion Detection System – A Detailed Survey," *Journal of Network and Computer Applications*, 2024.
4. Hussain, A., et al., "A Comprehensive Review of AI-Based Intrusion Detection Systems," *Future Generation Computer Systems*, 2023.
5. Singh, R., & Bedi, P., "AI-Driven Intrusion Detection Systems Leveraging Deep Learning for Network Security," *IEEE Access*, 2024.
6. Zhang, Y., et al., "Intrusion Detection Based on Federated Learning: A Systematic Review," *Journal of Systems Architecture*, 2023.
7. Li, J., et al., "Intrusion Detection at Scale with the Assistance of a Command-line Language Model," *arXiv preprint arXiv:2401.02541*, 2024.
8. Verma, A., et al., "Explainable Intrusion Detection Systems (X-IDS): A Survey of Methods and Trends," *IEEE Transactions on Dependable and Secure Computing*, 2022.
9. Wang, T., et al., "Deep Reinforcement Learning for Intrusion Detection in IoT: A Survey," *Sensors*, 2024.
10. O'Shea, D., et al., "AI-Powered Intrusion Detection Systems: Real-World Performance Analysis," *Journal of Cybersecurity Research*, 2024.
11. Patel, R., et al., "Developing an IDS for Network Security Using Machine Learning," *International Journal of Network Security*, 2024.
12. Sharma, H., & Arora, A., "Machine Learning for Network Intrusion Detection—A Comparative Study," *Computers & Electrical Engineering*, 2023.
13. Agarwal, K., et al., "Advancements in Training and Deployment Strategies for AI-Based IDS in IoT," *IEEE Transactions on Industrial Informatics*, 2025.
14. Ali, Z., et al., "Advancing Cybersecurity: A Comprehensive Review of AI-Driven Detection Techniques," *Information Systems Frontiers*, 2024.

# Appendices

## Appendix A: Technology Stack and Environment Setup

- **Backend Technologies:**
  - Python 3.x
  - Flask
  - Apache Kafka
  - MongoDB
  - Firebase Firestore
  - Scapy
  - PyTorch
  - HuggingFace Transformers
  - Scikit-learn, XGBoost, Joblib
- **Frontend Technologies:**
  - React.js
  - HTML5, CSS3, JavaScript
- **Libraries Used:**
  - Matplotlib (Chart generation)
  - Flask-CORS (Cross-Origin Resource Sharing)

## Appendix B: Dataset Details

- **Dataset Used:** CICIDS2017
- **Source:** Kaggle
- **Attack Types Covered:**
  - DDoS
  - DoS-Hulk
  - DoS-GoldenEye
  - DoS-Slowhttptest
  - DoS-slowloris

- Bot
- PortScan
- SSH-Patator
- FTP-Patator
- Web-Attack-BruteForce
- Web-Attack-XSS
- Web-Attack-SqlInjection
- Infiltration
- Heartbleed

## **Appendix C: Model Hyperparameters**

- **Random Forest:**
  - Number of trees: 100
  - Criterion: Gini
- **XGBoost:**
  - Learning Rate: 0.1
  - Max Depth: 6
  - Estimators: 100
- **Logistic Regression:**
  - Solver: liblinear
  - Regularization: L2
- **Decision Tree:**
  - Criterion: Gini
  - Max Depth: None (default)
- **DistilBERT Fine-Tuning:**
  - Model: distilbert-base-uncased
  - Max Sequence Length: 128 tokens
  - Learning Rate: 2e-5
  - Batch Size: 16
  - Epochs: 3
  - Evaluation Metric: Accuracy

## Appendix D: Evaluation Metrics Used

- Accuracy
- Precision
- Recall
- F1-Score
- ROC AUC Score
- Model Preparation Time
- Evaluation Runtime
- Throughput (Samples per Second)

## Appendix E: System Architecture Overview

- **Data Capture:** Scapy-based packet sniffer.
- **Streaming:** Kafka message broker with separate topics for packets, flow features, and model inference.
- **Inference:**
  - Multiple pre-trained ML models.
  - DistilBERT for textual payload analysis.
- **Data Storage:**
  - MongoDB for raw and processed data.
  - Firestore for real-time dashboard syncing.
- **Dashboard:**
  - React-based UI.
  - APIs exposed via Flask backend.