**“AI/ML-Powered Network Intrusion Detection System”**

Submitted by

**Jyoshika Lalam (VU22CSEN0100432)**

**Sushanth Kanuru (VU22CSEN0100016)**

Under the esteemed guidance of

**Venkata Ramana Murthy Oruganti**

**Associate Professor**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**GITAM SCHOOL OF TECHNOLOGY**

**GITAM (Deemed to be University)**

**VISAKHAPATNAM**

**2025**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**GITAM SCHOOL OF TECHNOLOGY**

**GITAM (Deemed to be University)**

A logo of a bird

Description automatically generated

**DECLARATION**

I hereby declare that the project report entitled “**AI/ML-Powered Network Intrusion Detection System”** is an original work done in the Department of Computer Science and Engineering, GITAM School of Technology, GITAM (Deemed to be University) The work has not been submitted to any other college or University for the award of any degree or diploma.

Date: 11/07/2025

Registration No(s) Names Signature

VU22CSEN0100432 Jyoshika Lalam Jyoshika L

VU22CSEN0100016 Sushanth Kanuru Sushanth K

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**GITAM SCHOOL OF TECHNOLOGY**

**GITAM (Deemed to be University)**

A logo of a bird

Description automatically generated

**CERTIFICATE**

This is to certify that the project report entitled “**AI/ML-Powered Network Intrusion Detection System**” is a bonafide record of work carried out by Jyoshika Lalam (VU22CSEN0100432), Sushanth Kanuru (VU22CSEN0100016))**.**

Date : 11/07/25

|  |  |
| --- | --- |
| Project Guide  Dr. V Ramana Murthy Oruganti  Associate Professor, CSE, GST,  GITAM, Visakhapatnam | Head of the Department  Dr. Gondi Lakshmeeswari,  Associate Professor, CSE, GST,  GITAM, Visakhapatnam |

**ACKNOWLEDGEMENT**

This project was conducted as a requirement of the Department of Computer Science and Engineering at the GITAM School of Technology. We would like to express our gratitude to the department for the invaluable resources and infrastructure that facilitated our research.

We are especially thankful to **Dr. Venkata Ramana Murthy Oruganti**, Associate Professor in the CSE Department, for his constant support, insightful guidance, and motivation throughout this endeavour. His expertise has been crucial to the successful completion of this work.

Lastly, we wish to convey our deep appreciation to our families and friends for their encouragement and unwavering support, which played a vital role in helping us achieve our goals for this project.

TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Description** | **Page No.** |
| 1. | Problem Statement | 6 |
| 2. | Objectives | 7 |
| 3. | Expected Outcomes | 8 |
| 4. | Deliverables | 9 |
| 5. | Team Members and Responsibilities | 10 |
| 6. | Tech Stack | 11 |
| 7. | System Architecture | 13 |
| 8. | How the Code Works | 16 |
| 9. | Dashboard Features | 19 |
| 10. | Results Summary | 21 |
| 11. | Application Demo Video | 23 |
| 12. | GitHub Repository Link | 24 |
| 13. | Conclusion and Future Scope | 25 |

**1**. **Problem Statement**

The rapid digitalization of businesses and institutions has led to a dramatic increase in the volume and complexity of network traffic. Simultaneously, there has been a surge in cyberattacks that are capable of bypassing traditional security systems. Signature-based firewalls and Intrusion Detection Systems (IDS), though prevalent, are often inadequate in identifying novel threats, especially those embedded in encrypted traffic. These conventional approaches largely depend on predefined patterns and signatures, making them ineffective against zero-day attacks and adaptive malware.

Manual traffic inspection and classification are not only time-consuming but also prone to errors, thus resulting in delayed threat response. The modern cybersecurity landscape demands intelligent, automated systems that can analyse traffic in real time and identify anomalies before damage is done. ShieldX is designed to address these challenges using Artificial Intelligence (AI) and Machine Learning (ML) technologies. It aims to deliver a smart and scalable system capable of analysing large volumes of network traffic, recognizing patterns of normal and abnormal behaviour, and detecting potential intrusions without manual intervention. It also offers the added benefit of analysing encrypted traffic using behavioural features, thus preserving data privacy while maintaining detection efficiency.

**2. Objectives**

The ShieldX project is built with a comprehensive set of goals in mind to address the growing need for intelligent network intrusion detection. The foremost objective is to develop an AI/ML-powered system capable of classifying incoming network traffic in real time. Unlike traditional IDS that depend on static rule sets, this system will utilize machine learning algorithms trained on labelled traffic data to distinguish between benign and malicious traffic.

To achieve this, the system must first ingest and preprocess raw network data logs, which may be semi-structured or unstructured. Once cleaned and formatted, the data will be passed into a feature extraction module that identifies the most significant attributes for traffic classification. The objective is also to build and fine-tune multiple machine learning models, including Random Forest, SVM, K-Nearest Neighbours, and XGBoost, and evaluate their performance using real-world datasets like CIC IDS 2017.

In addition to backend classification, another critical goal is to offer a user-friendly, visually rich frontend dashboard. This dashboard must allow users to upload network logs, initiate preprocessing, select classifiers, and view classification results, metrics, and visualizations with minimal technical expertise. Finally, the system should maintain a high level of accuracy, with a target of over 95% classification accuracy, low false positive rates, and modularity for future expansion.

**3. Expected Outcomes**

The expected results of the ShieldX system include a highly accurate and automated classification of network traffic. Upon completion, the platform should be capable of receiving network logs in the form of CSV files, preprocessing them to remove inconsistencies, applying the chosen machine learning algorithm to classify traffic as benign or malicious, and displaying the results to the user with real-time updates.

The system is expected to outperform traditional rule-based IDS solutions by dynamically learning from new data. This will allow it to detect unknown or mutated threats that signature-based systems would fail to identify. The visualization features of the dashboard will empower users with insightful metrics, such as confusion matrices, class distributions, and accuracy scores, helping both technical and non-technical users to interpret the results easily.

Another important outcome is scalability. The system should perform equally well on small and large datasets, and future improvements should allow for handling live traffic streams. Privacy-preserving analysis is also key, as ShieldX does not decrypt data packets but instead analyses behavioural and statistical indicators. This allows it to be compliant with data protection standards while still providing robust security insights.

**4. Deliverables**

ShieldX’s output includes both functional software and supporting materials. The core deliverable is a fully functional AI-based Intrusion Detection System that integrates preprocessing, model training, inference, and visualization. Accompanying this system is a Streamlit dashboard, allowing users to operate the model interactively via a web interface.

Additional deliverables include:

**Trained Models**: Models trained on the CIC IDS 2017 dataset.

**Source Code**: Python scripts for model training, preprocessing, evaluation, and dashboard integration.

**Documentation**: Complete system documentation describing installation, architecture, methodology, testing, and evaluation.

**Demo Video**: A walkthrough video showing system functionality, UI interaction, and model performance.

**Sample Dataset Files**: Real and synthetic data used for model training and testing.

**5. Team Structure & Responsibilities**

ShieldX was developed by a dedicated two-member team, each with clearly defined responsibilities to ensure a streamlined and efficient development process.

Table 1: Roles and Responsibilities of the ShieldX Development Team

|  |  |  |
| --- | --- | --- |
| **Name** | **Role** | **Responsibilities** |
| Jyoshika Lalam | Machine Learning Engineer & Analyst | Handled data cleaning, model design, performance analysis, dashboard integration, and research on attack vectors for feature selection. |
| Sushanth | Backend Developer & System Designer | Designed the overall system architecture, integrated frontend and backend, automated the pipeline, and managed exception handling and documentation. |

Jyoshika Lalam served as the Machine Learning Engineer & Research Analyst, responsible for preparing and preprocessing the CIC IDS 2017 dataset, designing and evaluating ML models (Random Forest, XGBoost, Logistic Regression, SVM), and selecting the best-performing classifier based on accuracy, precision, recall, and F1-score. She analyzed cyberattack patterns (SQL Injection, XSS, DoS) to engineer relevant features and developed a Streamlit-based dashboard for real-time intrusion detection. She also validated the system across various datasets to ensure reliability and robustness.

Sushanth, as the Backend Developer & System Designer, handled the system architecture, connecting ML models to the frontend using Streamlit. He optimized model performance through hyperparameter tuning and automated the entire pipeline—from CSV upload to prediction—using modular Python scripts. He referenced external repositories for design inspiration and ensured backend robustness through extensive testing and error handling. He also contributed to the project’s documentation and final report.

**6. Technology Stack**

The ShieldX project is built upon a diverse and powerful set of technologies that span multiple layers of modern full-stack application development. These tools were carefully chosen to enable rapid development, scalable deployment, real-time interaction, and seamless machine learning integration.

**Table 2: Categorized Technology Stack Overview**

| **Layer / Category** | **Technology / Tool** | **Purpose** |
| --- | --- | --- |
| **Programming Languages** | Python, TypeScript, JavaScript | Python for ML/Flask; TypeScript for Vite + React frontend |
| **Machine Learning** | scikit-learn, XGBoost | Training and inference of Random Forest, SVM, KNN models |
| **Data Processing** | Pandas, NumPy, StandardScaler | Preprocessing: cleaning, transformation, encoding, scaling |
| **Visualization (ML)** | Seaborn, Matplotlib, Plotly | Confusion matrix, correlation heatmaps, metric graphs |
| **Backend Framework** | Flask | REST API to handle file uploads and return predictions |
| **Notebook Environment** | Jupyter Notebook | Model prototyping, EDA, feature importance, performance analysis |
| **Frontend Framework** | React + Vite | Fast, modern frontend using JSX + TSX + Tailwind |
| **UI & Styling** | Tailwind CSS | Utility-first CSS framework for designing the Vite dashboard |
| **UI Components** | Streamlit | Rapid UI building and dashboarding for ML models |
| **Version Control** | Git, GitHub | Source code tracking, collaborative development |

**Table 3: Stack Alignment with Components**

| **Component** | **Tools / Stack Highlights** |
| --- | --- |
| **ML Model Development** | Python, Jupyter, scikit-learn, XGBoost, Pandas, Seaborn |
| **ML Deployment & API** | Flask, Streamlit, Python, sklearn, REST |
| **Frontend (Streamlit UI)** | Streamlit, Plotly, Pandas |
| **Frontend (Vite UI)** | React, Vite, TypeScript, TailwindCSS, HTML5 |
| **Data Management** | CICIDS2017 CSVs, Pandas, NumPy |
| **Visualization** | Matplotlib, Seaborn (EDA), Plotly |
| **Documentation** | Markdown, Jupyter Notebook, Vite docs/ folder |
| **Version Control** | Git (CLI), GitHub repository |

**7. System Architecture**

The ShieldX project is architected as a modular, full-stack AI-powered Network Intrusion Detection System (NIDS) designed for both real-time and offline analysis of network traffic. It leverages a layered structure that separates data preprocessing, model inference, and user interaction, ensuring maintainability, extensibility, and performance.

**7.1. Architectural Overview**

The system consists of the following three primary layers:

1. **Frontend Interface Layer**

Built using Streamlit and Vite (React + TypeScript), this layer allows users to upload data, initiate predictions, and view reports and visualizations.

1. **Machine Learning Backend Layer**

Consists of trained models (Random Forest, SVM, KNN) that classify traffic logs as benign or malicious. The logic is implemented in both Jupyter notebooks (for prototyping) and a Python Flask server (for deployment).

1. **Data Handling & Processing Layer**

Responsible for parsing CSV logs, cleaning and transforming data, selecting relevant features, and standardizing inputs for ML inference.

These components are interconnected via API calls, file I/O, and a shared data model, allowing flexible experimentation or real-time use depending on deployment mode.

**7.2. Detailed Architecture Flow**

Below is the layered step-by-step flow of data and control across the system:

**1. User Interaction Layer**

Users begin their interaction via one of the following interfaces:

**Streamlit UI (app.py)** for uploading .csv files and choosing classifiers.

**Vite Dashboard (React + TypeScript)** for future support of real-time monitoring, analytics history, and model management.

Each interface sends the data to the backend for processing via:

Flask HTTP routes (/predict)

Internal method calls in Streamlit

**2. Upload Handler**

This module validates the format of uploaded files. It ensures the dataset is:

In .csv format

Contains required features like Label

Not missing critical columns or corrupted values

Upon validation, the file is parsed using Pandas and passed into the data pipeline.

**3. Data Preprocessing Unit**

This component performs a series of essential cleaning and transformation tasks:

Dropping NaN, inf, or invalid rows

Trimming and normalizing column names

Binary label encoding (BENIGN → 0, attacks → 1)

Selecting only numerical and boolean columns

Scaling data using StandardScaler

The resulting dataset is split into train/test sets using train\_test\_split.

**4. Model Training and Selection**

The user selects one of the supported models:

**Random Forest**: Default baseline due to its robustness

**Support Vector Machine (SVM)**: Effective for margin-based classification

**K-Nearest Neighbors (KNN)**: Lightweight and intuitive

The selected model is trained on the training set in real-time and evaluated on the test set. Model metrics are then generated and stored.

**5. Inference and Prediction Engine**

This engine processes the test data through the trained model and produces predictions (0 or 1). Predictions are stored in memory and formatted as structured tables for downstream modules.

**6. Metrics Generator**

Key classification metrics are computed:

Confusion Matrix (TP, TN, FP, FN)

Accuracy, Precision, Recall, F1-score

Class distribution of predictions

These metrics are packaged and returned to the dashboard for visualization.

**7. Dashboard Visualizer**

Both dashboards (Streamlit and Vite) present the following visual elements:

Bar chart of benign vs malicious predictions

Heatmap of the confusion matrix

Classification table with all metrics

(Notebook only) Feature importance using .feature\_importances\_ or XGBoost visual plots

Figure 1:

A screenshot of a computer screen

AI-generated content may be incorrect.

**8. Workflow & Code Explanation**

The ShieldX system follows a modular, end-to-end workflow that automates the process of data ingestion, preprocessing, machine learning-based classification, and real-time visualization. This pipeline is implemented across Jupyter notebooks (ShieldX\_AIML\_NetworkSecurity.ipynb), the Streamlit dashboard (app.py), and the React-based frontend (in Intel-ShieldX-vite\_Dashboard/).

The workflow is designed to support both experimentation and deployment, with a strong emphasis on usability and model explainability.

**Step-by-Step Breakdown**

**8.1. Data Upload & Input Validation**

Users begin by uploading network traffic logs in .csv format via the Streamlit dashboard. In the Vite frontend, this functionality is planned for future integration. Uploaded files are validated to ensure required columns (like Label) are present and that no null or infinite values dominate the dataset.

Code Reference:

app.py: Streamlit file\_uploader()

Flask @app.route('/predict'): Receives CSV from frontend

**8.2. Data Preprocessing**

Once the CSV is uploaded, preprocessing is triggered. This stage removes incomplete or corrupt rows, trims inconsistent column names, and converts categorical labels into binary format (e.g., “BENIGN” → 0, attacks → 1). Only numerical and boolean features are retained, ensuring compatibility with machine learning models. The cleaned data is normalized using StandardScaler.

Code Reference:

Notebook: df.dropna(), Label encoding, StandardScaler()

app.py: preprocess\_data(file)

**8.3. Train-Test Split**

The dataset is split into 70% training and 30% testing data using train\_test\_split() from sklearn.model\_selection. This ensures that each model is evaluated on unseen data immediately after training.

Code Reference:

Notebook and app.py: X\_train, X\_test, y\_train, y\_test = train\_test\_split(...)

**8.4. Model Training**

Users select a classifier from the sidebar (Streamlit) or configure it in the code (Flask/notebook). The supported models include Random Forest, SVM, and K-Nearest Neighbors. The model is trained using the processed training data. In the notebook, an ensemble comparison is performed to identify the best performer based on multiple metrics.

Code Reference:

Notebook: RandomForestClassifier(), SVC(), KNeighborsClassifier()

app.py: Dynamically loads classifier based on user input

**8.5. Prediction & Inference**

Once trained, the selected model predicts the labels of the test set (i.e., whether each traffic flow is benign or malicious). These predictions are captured and used to generate performance metrics and visualizations.

Code Reference:

Notebook: model.predict(X\_test)

Streamlit: Output shown in metrics section

Flask: Predictions returned as JSON for API use

**8.6. Evaluation Metrics**

The system computes classification metrics including precision, recall, F1-score, and accuracy. A confusion matrix is also generated to analyze false positives and false negatives. In the notebook, feature importance is extracted (for RF and XGBoost) to help interpret the most influential network attributes.

Code Reference:

classification\_report(y\_test, y\_pred)

confusion\_matrix(y\_test, y\_pred)

model.feature\_importances\_ (RF), plot\_importance() (XGBoost)

**8.7. Visualization & Reporting**

Results are displayed using:

Confusion matrix heatmaps

Prediction class bar charts

Accuracy and metric tables

Feature importance plots (in notebook)

In Streamlit, these are visualized with Plotly and Pandas. In the Vite frontend, visual components are being developed to display these via Axios data from Flask.

Code Reference:

Streamlit: st.plotly\_chart(), st.dataframe()

Notebook: seaborn.heatmap(), plotly.express.bar()

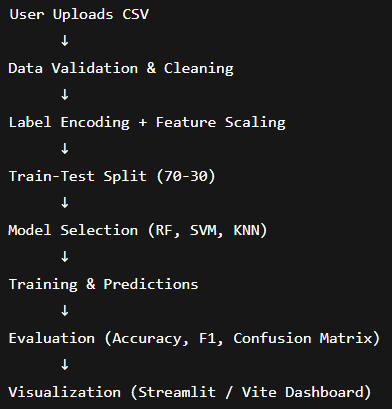
**8.8. Prediction Export (Notebook Only)**

The predictions can be saved as .csv files in the notebook version for downstream analysis, reporting, or archival purposes. This supports forensic investigation and integration into broader SIEM systems.

Code Reference:

pd.DataFrame(predictions).to\_csv('output.csv')

Figure 2:



**9. Dashboard Features**

The ShieldX project provides two powerful dashboards designed to serve different purposes in the cybersecurity workflow: a Streamlit-based ML dashboard and a Vite-based frontend dashboard. Both interfaces aim to facilitate network traffic evaluation through interactive, accessible, and real-time visualizations. Below is a detailed explanation of both dashboards.

**Streamlit Dashboard:**

The ShieldX Streamlit Dashboard is an intuitive web application that allows researchers and analysts to upload network traffic logs and evaluate them against trained machine learning models. Designed for both usability and depth, this dashboard integrates the full machine learning pipeline and displays results in real-time.

**Key Features**

**1. CSV File Upload**  
Users can upload .csv files derived from traffic datasets like CIC IDS 2017. Upon upload, the system immediately checks for column validity, data integrity, and ensures that essential labels (e.g., "Label" or "Attack") are correctly formatted.

**2. Fully Automated Data Preprocessing**  
Once a file is uploaded, the system initiates an automated cleaning process. It removes null, infinite, or corrupted rows, trims whitespace from column headers, and converts categorical labels to binary form (e.g., “BENIGN” → 0, others → 1). Only numerical and boolean columns are retained to ensure compatibility with machine learning models.

**3. Model Selection and Real-Time Predictions**  
The sidebar provides a dropdown for users to select among pre-configured classifiers—Random Forest, Support Vector Machine (SVM), and K-Nearest Neighbors (KNN). Upon selection, the model is trained on a 70:30 split of the uploaded dataset, and predictions are immediately generated.

**4. Visual Output & Evaluation Metrics**  
Results are visualized through a series of informative components:

**Confusion Matrix Heatmap**: Shows the distribution of true/false positives and negatives.

**Prediction Class Distribution**: A bar chart indicating the number of benign versus malicious classifications.

**Classification Report Table**: Presents precision, recall, and F1-score for both classes.

**Model Accuracy Score**: Highlights the overall prediction accuracy.

**5. Feature Importance (Notebook-specific)**  
Though not displayed in the Streamlit UI, the associated Jupyter notebook (ShieldX\_Network\_Security.ipynb) includes feature importance charts generated from Random Forest and XGBoost models. These help researchers understand which traffic features most influence the detection process.

**6. Exporting Predictions**  
In the notebook environment, users can export predicted labels to a CSV file for offline analysis, audit logging, or forensic investigation. This capability is ideal for creating reports or feeding results into downstream security tools.

**Vite + React + TypeScript Dashboard**

In addition to the Streamlit dashboard, ShieldX includes a modern web dashboard built using the Vite bundler and the React framework. This interface is primarily designed for real-time interaction, user management, and visualization beyond just classification.

**Highlights of the Vite Dashboard**

**Frontend Framework**: Built with React and TypeScript, ensuring maintainability and type safety. Styled using Tailwind CSS for a consistent, responsive design.

**Routing and Views**:

**Home Page**: Offers an overview of ShieldX features, recent activity, and system status.

**Live Monitor Page**: Intended for future real-time integration, this page will stream live traffic predictions once connected to packet sniffers or APIs.

**Reports Page**: Displays previously processed datasets, metrics history, and performance comparisons in an interactive graphical format.

**Backend Integration**: Though currently using mock data, this dashboard is being extended to interact with the Flask backend API (/predict endpoint) to fetch real-time inference results.

**Documentation Support**: The docs/ folder in the Vite project includes detailed API specifications (api.md), system design and data flow diagrams (architecture.md), and a development timeline (project-timeline.md).

**10. Results Summary**

To assess the performance of ShieldX, experiments were conducted using a sampled subset of the CIC IDS 2017 dataset. The models were trained and tested using a 70:30 split and evaluated using standard classification metrics.

Table 4: Model Comparison Summary with Description and Application Context in ShieldX

| **Model** | **Description** | **Use Case** |
| --- | --- | --- |
| Random Forest | Ensemble of decision trees | High-accuracy baseline detection |
| Support Vector Machine (SVM) | Max-margin classifier | Suitable for binary classifications |
| K-Nearest Neighbors | Instance-based learner | Lightweight model for quick tests |
| Ensemble Hybrid | Combination of above models | For boosting recall/precision |

**Evaluation Metrics:**

**Accuracy**: Ratio of correct predictions to total.

**Precision**: True positives over predicted positives.

**Recall**: True positives over actual positives.

**F1 Score**: Harmonic mean of precision and recall.

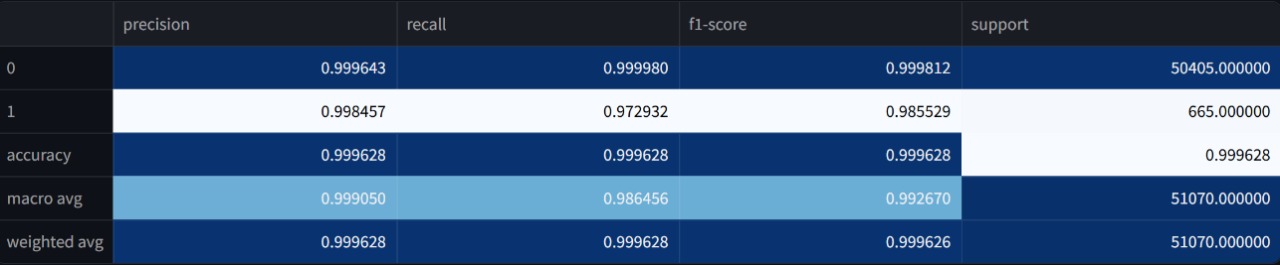
**ROC-AUC**: Area under the ROC curve.

**Confusion Matrix**: TP/TN/FP/FN visual analysis.

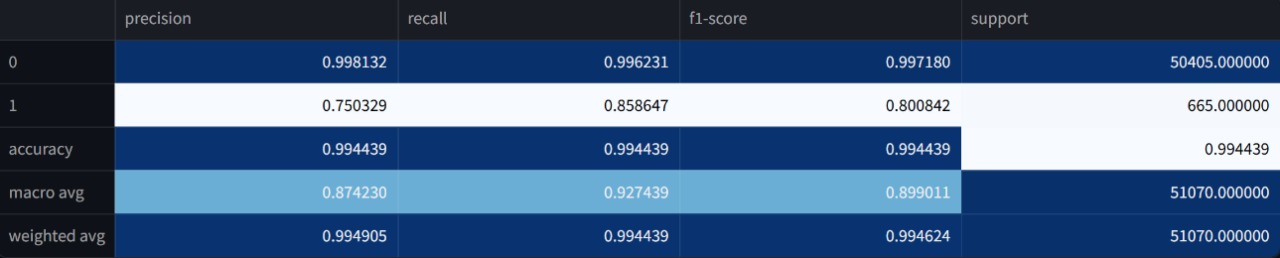
The confusion matrix results showed a very low rate of false positives and false negatives, particularly in the XGBoost and Random Forest models. These results support the selection of ensemble-based classifiers for deployment.

**Random Forest:** The Random Forest classifier achieved an outstanding overall accuracy of 99.96%. It maintained excellent precision and recall for both benign and attack classes. The recall for attack traffic reached 97.29%, with a strong F1-score of 98.55%, making it highly reliable for real-world intrusion detection with minimal false alarms.

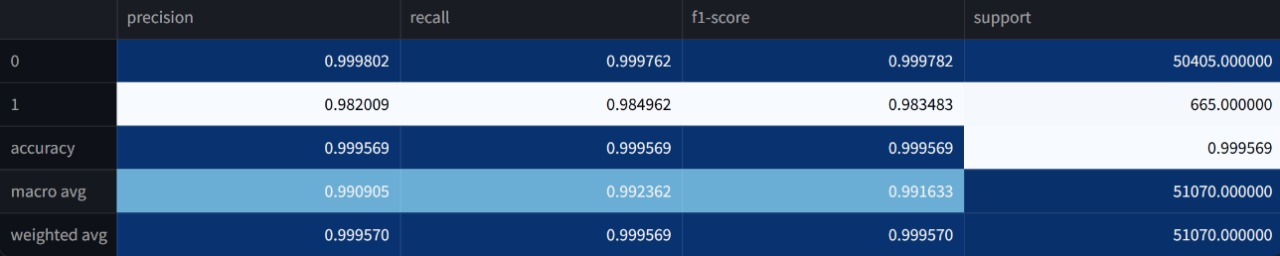
Figure 3: Detailed classification metrics of the Random Forest model used in ShieldX.



**SVM:** The Support Vector Machine (SVM) classifier achieved an overall accuracy of 99.44%. It performed very well on benign traffic with a precision of 99.81%. For attack traffic, the recall reached 85.86%, showing that the model can detect threats with reasonable reliability, although with slightly lower precision compared to KNN.

Figure 4: Evaluation summary of the Support Vector Machine classifier on the CIC IDS 2017 dataset.

**KNN:** The K-Nearest Neighbors (KNN) model achieved an overall accuracy of 99.95% on the CIC IDS 2017 dataset. It performed exceptionally well on benign traffic with nearly perfect precision and recall. Despite class imbalance, it maintained a high F1-score of 98.49% for attack traffic, demonstrating strong reliability in detecting intrusions.

Figure5: Performance metrics of the K-Nearest Neighbors model on the CIC IDS 2017 dataset. 

**11. ShieldX Video Walkthrough**

This video demonstrates **ShieldX**, an AI-powered Intrusion Detection System (IDS) designed to detect malicious network traffic using machine learning. We begin with a brief overview of the project and the limitations of traditional IDS systems, then dive into a hands-on walkthrough of its components.

### **ShieldX Video Timelines:**

1. **Intro** – 0:00 – 0:20
2. **Project Overview** – 0:20 – 0:56
3. **Dataset & Notebook Walkthrough** – 0:56 – 2:01
4. **Model Export & App Integration** – 2:01 – 2:25
5. **Flask Inference Backend** – 2:25 – 3:04
6. **Vite Dashboard Walkthrough** – 3:04 – 4:18
7. **System Architecture Diagram** – 4:18 – 4:46
8. **Wrap-Up** – 4:46 – 5:01

**Video Link:** <https://drive.google.com/file/d/17cjR4xNVqObd9vXh1MrrOXmxX0vLRTuH/view?usp=sharing>

**12. GitHub Repository**

All code, models, and resources for the ShieldX project are available on GitHub:

**Repository URL**: <https://github.com/JyoshikaLalam/ShieldX-AI-Network-Security>

The **ShieldX-AI-Network-Security** repository is logically structured into clearly separated directories and files, reflecting the modular architecture of the project. The repository is organized to support both backend services and a modern frontend dashboard. Each folder plays a specific role in the overall pipeline—from model training and inference to interactive dashboard visualization.

The root directory contains the main Flask API script (app.py), the Jupyter Notebook for experimentation, and the PDF summary report. The data/ directory holds CSV files from the CICIDS2017 dataset used to train and validate the models. Dependency management is handled via requirements.txt for the backend and package.json for the frontend.

The frontend dashboard resides in the Intel-ShieldX-vite\_Dashboard/ folder, built with Vite, React, TypeScript, and styled using Tailwind CSS. Within this folder, the src/ directory contains application logic, UI components, pages, and type declarations. The docs/ folder includes important documentation such as API references and system architecture diagrams.

**The full structure is as follows (Figure 6):**

A screenshot of a computer program

AI-generated content may be incorrect.

**13. Conclusion & Future Scope**

**Conclusion**

ShieldX demonstrates the effective application of machine learning to network intrusion detection. By integrating models like Random Forest, SVM, and KNN, the system achieves high accuracy in identifying malicious traffic using behavioral patterns. The project supports both offline experimentation through Jupyter notebooks and real-time testing via the Streamlit dashboard. With its clean modular design and user-friendly interfaces, ShieldX simplifies the process of traffic analysis and helps reduce dependency on traditional rule-based systems.

The addition of a modern frontend built using Vite and React further extends the system toward production-readiness. ShieldX proves to be a scalable and adaptable solution for early-stage deployment and educational use in cybersecurity domains.

**Future Scope**

In the future, ShieldX can be enhanced by incorporating real-time packet monitoring tools like Scapy or Wireshark to support live intrusion detection. Deployment to cloud platforms such as Streamlit Cloud or Heroku would allow 24/7 availability and scalable performance. Advanced deep learning techniques (like LSTM) can be explored to detect temporal patterns in traffic data.

Feature extensions such as online learning, role-based access in the Vite dashboard, and integration with enterprise SIEM systems can help ShieldX evolve from a research prototype into a robust, production-level IDS framework.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |