Network Anomaly Detection Using AI – Implementation Report

Team 8

December 12, 2025

1 Executive Summary

This project implements an AI-based network anomaly detection system to identify suspicious traffic patterns indicative of cyberattacks such as DoS activity and port scanning. The solution was developed in two parallel work streams: (1) *Data Analyst & AI Lead*, responsible for data preparation and model development, and (2) *Visualization & Alerting Lead & QA Lead*, responsible for the user-facing interface, alerting concept, and testing. The second group could work largely autonomously at the beginning by building the visualization layer and app structure while the first group focused on the ML pipeline.

The ML team first cleaned and prepared the dataset, performed feature selection, trained multiple candidate models, and selected the best-performing approach based on evaluation metrics. After model selection, the solution was integrated into the visualization and alerting application to support end-to-end processing of sample network logs and to display detected anomalies.

The results are strong for binary classification (normal vs. attack), which proved reliable and consistent across test runs. Multiclass classification (distinguishing between attack categories) also performed well, but is not yet fully integrated into the current application: the dashboard and alerting logic are still optimized for the binary case, and multiclass-specific visualizations and UI elements need to be added.

A key limitation is that the system currently operates on simulated / offline data rather than true real-time monitoring. The next step is to connect the pipeline to live traffic sources (e.g., flow exporters) and implement streaming inference plus operational alerting.

Overall, the project demonstrates that combining robust data preparation, model selection, and a dedicated visualization/QA track can yield a practical detection tool. With real-time integration and multiclass UI support, the system can evolve into a deployable monitoring component that not only flags attacks, but also provides rapid, actionable context for incident response in production networks.

2 Technical design and architecture

Figure 1 illustrates the overall technical design and architecture of the proposed network anomaly detection system. The architecture follows a modular pipeline approach, enabling a clear separation of concerns between data acquisition, machine learning, and user-facing components.

Network traffic data, currently based on simulated datasets, is first processed by a dedicated preprocessing layer responsible for data cleaning, feature extraction, and normalization. The prepared data is then forwarded to the AI layer, where trained machine learning models perform inference. An SVM is used for binary classification (normal vs. attack), while a Random Forest model handles multiclass attack classification.

The inference results are passed to the visualization and alerting layer, which provides dashboard-based insights into network behavior and highlights detected anomalies (refer to Figure 2). This design allows the visualization and alerting components to operate independently of the underlying model implementation. The architecture is extensible by design and can be augmented with a real-time monitoring component in future work to support live network traffic analysis.

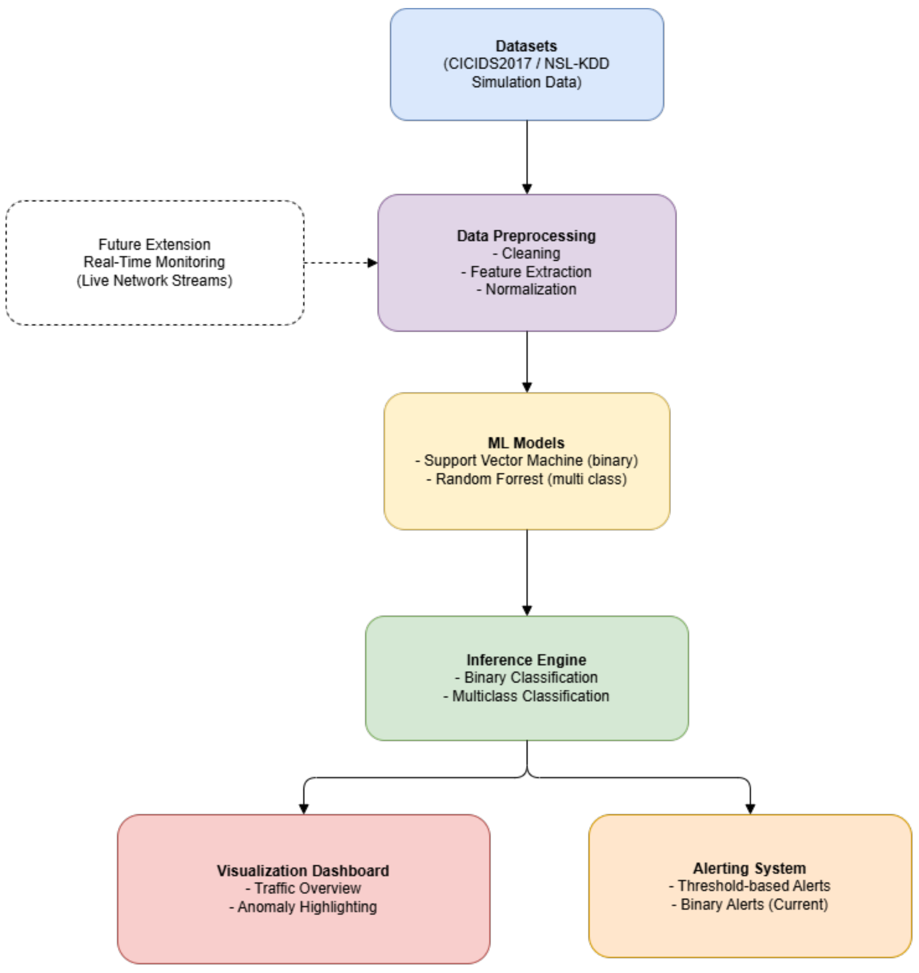
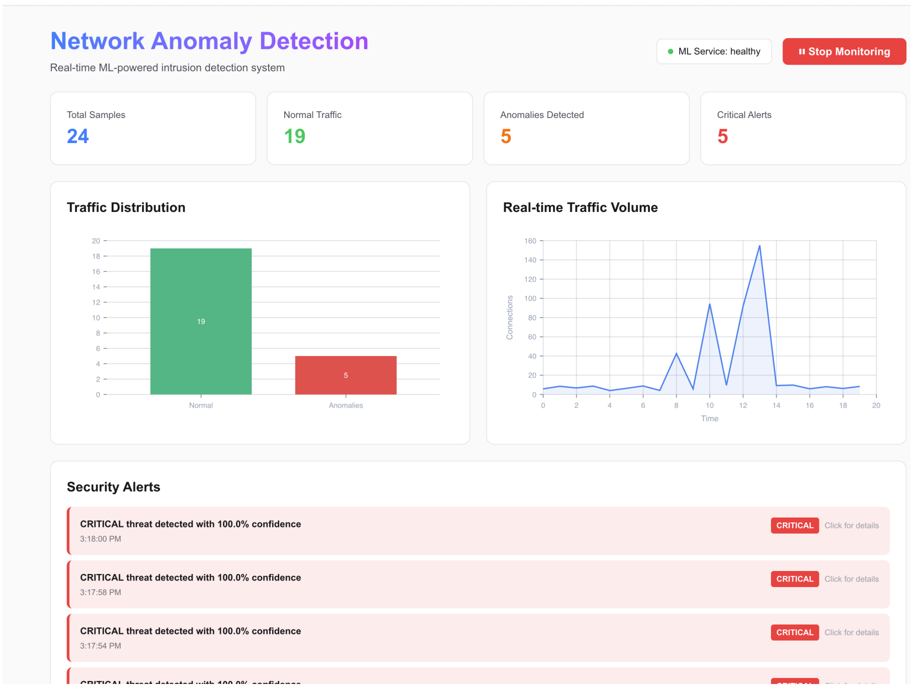


Figure 1: System Architecture and Component Integration Chart


Network Anomaly Detection

Real-time ML-powered intrusion detection system

● ML Service: healthy

Total Samples	Normal Traffic	Anomalies Detected	Critical Alerts
24	19	5	5

Security Alerts

CRITICAL threat detected with 100.0% confidence
3:18:00 PM
CRITICAL — Click for details

CRITICAL threat detected with 100.0% confidence
3:17:58 PM
CRITICAL — Click for details

CRITICAL threat detected with 100.0% confidence
3:17:54 PM
CRITICAL — Click for details

Figure 2: Application Dashboard signaling the user for network anomalies.

3 Security analysis

3.1 Threat Model

The system is designed to detect anomalous network traffic using machine learning techniques. The primary threat actor is assumed to be an external attacker with network access but without privileged access to the monitored system or the detection infrastructure.

The attacker's objectives include:

- Launching network-based attacks such as Denial of Service (DoS), port scans, or brute-force attempts.
- Evading detection by crafting traffic patterns that resemble normal behavior (e.g., low-and-slow attacks).
- Degrading the detection capability by injecting misleading or malicious data into the traffic stream.

The attacker is assumed to have knowledge of common intrusion detection techniques but no direct knowledge of the internal model parameters or feature selection.

3.2 Assumptions

The security analysis is based on the following assumptions:

- The training datasets (e.g., CICIDS2017, NSL-KDD) are correctly labeled and representative of real-world network traffic.

- Network logs are collected reliably and have not been tampered with before entering the anomaly detection pipeline.
- The detection system operates in a monitored environment where baseline “normal” traffic behavior is relatively stable.
- The system has sufficient computational resources to process traffic data and perform inference in near real-time.

3.3 Limitations

Despite its effectiveness, the proposed anomaly detection system has several limitations:

- Machine learning models may produce false positives when legitimate traffic deviates from previously observed patterns.
- Previously unseen (zero-day) attacks may not be accurately classified, especially in supervised learning approaches.
- Model performance is highly dependent on feature selection and data quality; noisy or biased data can significantly reduce detection accuracy.
- Adversarial adaptation over time may reduce detection effectiveness if the model is not regularly retrained and updated.

Overall, the system improves network visibility and attack detection but should be considered a complementary security mechanism rather than a standalone defense solution.

4 Testing methodology and results

4.1 Testing Methodology

The system was evaluated using supervised machine learning models tailored to the respective classification tasks. For binary classification (normal vs. attack), a Support Vector Machine (SVM) was employed due to its strong performance in high-dimensional feature spaces. For multiclass classification (attack type identification), a Random Forest classifier was selected because of its robustness, interpretability, and ability to handle non-linear feature interactions.
The dataset was split into training and testing subsets using a fixed ratio (e.g. *train_ratio* and *test_ratio*) of 20%, ensuring that no data leakage occurred between training and evaluation. In addition, cross-validation was applied on the training set to improve model generalization and reduce overfitting by evaluating performance across multiple data folds. Standard preprocessing steps (normalization, feature selection) were applied consistently to both subsets.

Model performance was assessed using the following metrics:

- Accuracy
- Precision and Recall
- F1-score
- False Positive Rate

4.2 Binary Classification Results (SVM)

The SVM achieved reliable performance in distinguishing normal traffic from malicious activity. False positives were limited, indicating that legitimate traffic was rarely misclassified as an attack.

Confusion Matrix:

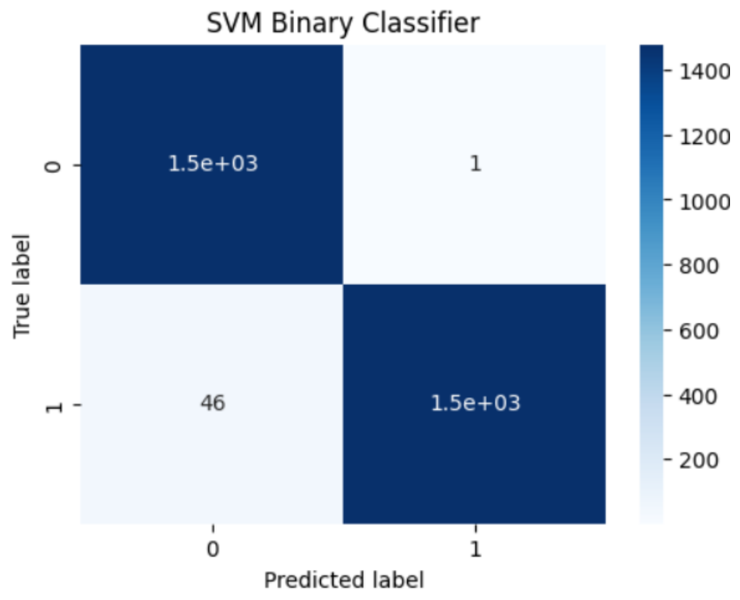


Figure 3: Confusion matrix for binary classification using SVM

ROC Curve:

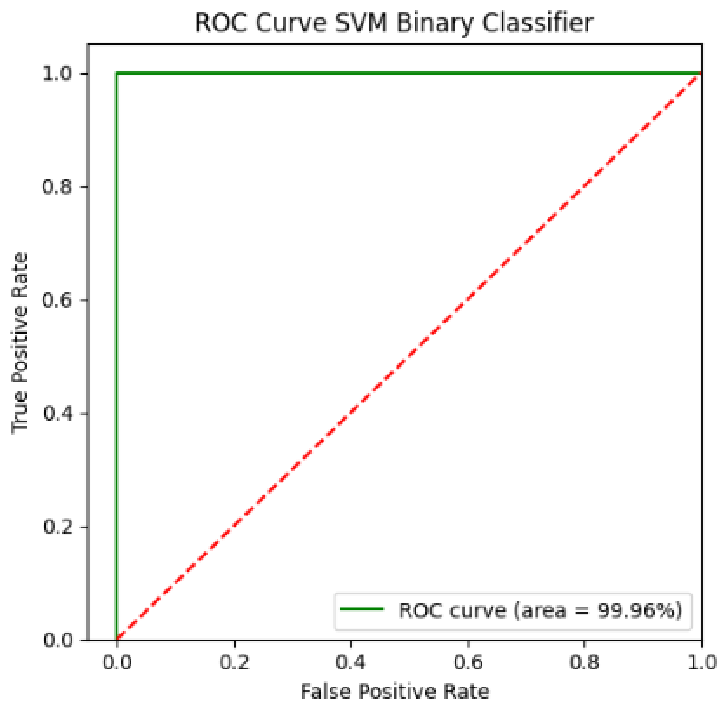


Figure 4: ROC curve for binary classification (SVM)

F1-Score and Accuracy:

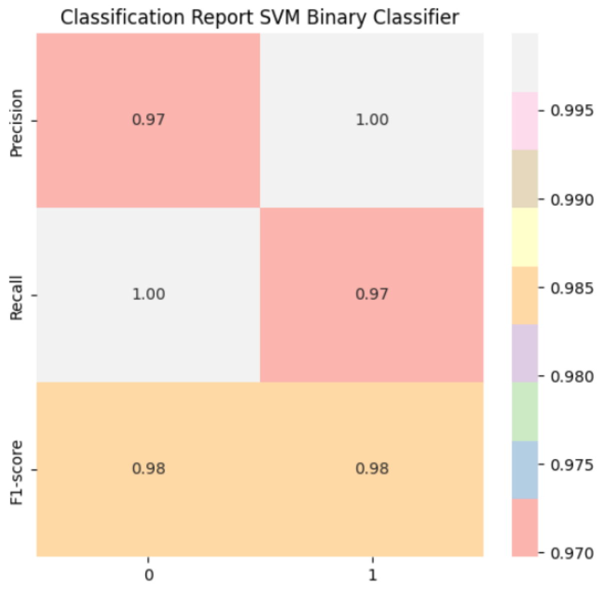


Figure 5: Metric report for binary classification (SVM); $Accuracy = 0.984\bar{3}$

Overall, the binary classifier demonstrates strong detection capability and is suitable for deployment in the current application setup.

Note: Initially, the unexpectedly high performance of the model was considered potentially suspicious. However, after detailed evaluation and consistently strong metrics across all tests, we confirm that the model performance is valid and sufficient for the intended use case.

4.3 Multiclass Classification Results (Random Forest)

The Random Forest model showd good performance in differentiating between multiple attack types. Misclassifications mainly occurred between attack classes with similar traffic patterns, which is expected in multiclass intrusion detection scenarios.

Confusion Matrix:

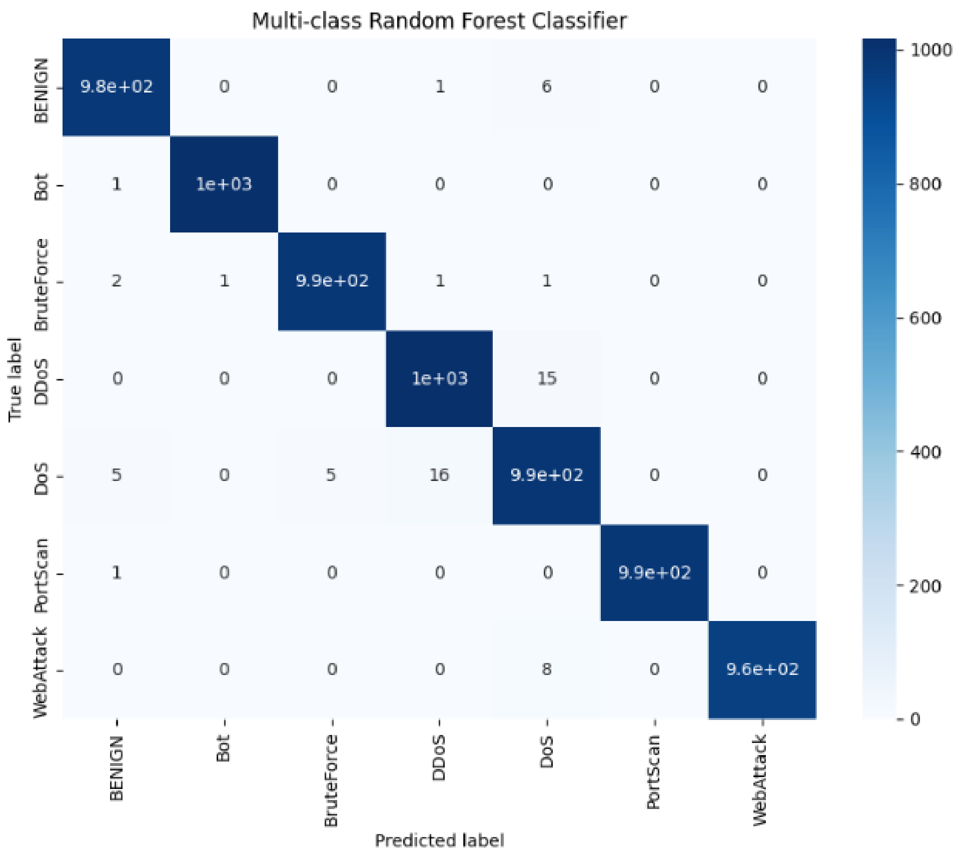


Figure 6: Confusion matrix for multiclass classification using Random Forest

Per-Class F1-Scores:

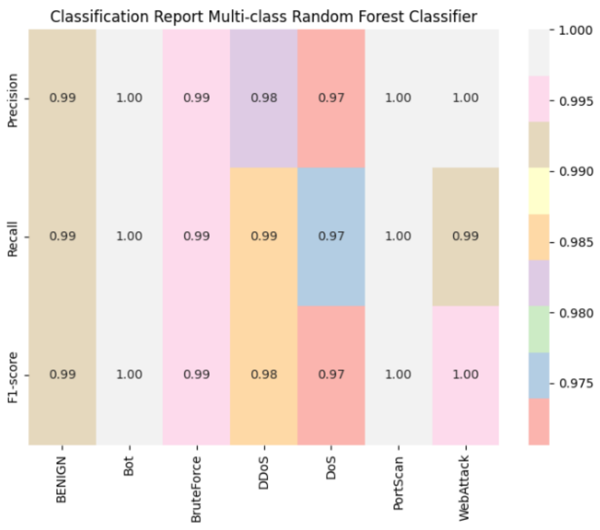


Figure 7: Metric report for multiclass classification (RF); $Accuracy = 0.991$

Although the multiclass model performs well, its results are not yet fully reflected in the current visualization and alerting interface, which is primarily optimized for binary classification.

4.4 Summary of Results

In summary, the SVM-based binary classifier provides reliable and low-noise detection of anomalous traffic, while the Random Forest multiclass classifier offers promising attack differentiation capabilities. Future work includes tighter integration of multiclass results into the user interface and extension of the testing pipeline to real-time traffic data.

5 Key Takeaways

This project demonstrates that machine-learning-based anomaly detection can reliably identify malicious network traffic when supported by proper data preprocessing and model selection. For future development, special attention must be given to keeping the models up to date, as changing network behavior and emerging attack patterns can significantly impact detection performance.

When integrating the system into a real-time monitoring environment, challengs such as stream processing, latency constraints, and stable feature extraction must be addressed. Additionally, any preprocessing techniques used during training, such as PCA (Principal Component Analysis), must be applied consistently during inference to avoid performance degradation.

Overall, maintaining data quality, ensuring pipeline consistency, and regularly validating model performance are critical for transitioning the system from a simulated environment to a production-ready deployment.

6 Team member roles and contributions

- **Member:** Lukas W. Blochmann, **Position:** Data Analyst
  - Analyzed, cleaned, and preprocessed the datasets
  - Defined key feature characteristics for model training
  - Prepared and documented the data preprocessing pipeline and report
  - Supported data preparation interfaces for model integration with DataPrepper API
  - Final report and coordination
- **Member:** Frederik Lind, **Position:** QA Lead
  - Defined quality strategy and acceptance criteria for the ML service and dashboard
  - Set up automated CI pipelines for linting, builds, and basic regression checks
  - Integrated static code analysis and enforced quality gates with SonarQube
  - Ensured dependency, security, and build stability across the project
  - Documented QA procedures and supported release readiness
- **Member:** Mohammed Jbilou, **Position:** AI Lead

  - Designed and implemented the ML inference service (Flask API)
  - Trained and optimized ML models for binary and multiclass anomaly detection
  - Defined threat level classification based on model confidence
  - Managed model persistence and integration with the frontend

- **Member:** Nicolas Gillard, **Position:** Visualization & Alerting Lead
  - Designed and implemented the visualization dashboard
  - Integrated model outputs into visual analytics and alerts
  - Developed views for normal vs. anomalous traffic
  - Supported usability testing and visual consistency

7 References and datasets used

Datasets in use

References

[1] Canadian Institute for Cybersecurity. CICIDS2017 Dataset.
`https://www.unb.ca/cic/datasets/ids-2017.html`

[2] University of New Brunswick. NSL-KDD Dataset.
`https://github.com/Jehuty4949/NSL_KDD`

Used tools

Frontend

- Language: TypeScript
- Framework & Libraries:
  - Next.js 16.0.8 (React framework)
  - React 19.2.0 (UI library)
  - Tailwind CSS 4 (styling)
  - Nivo 0.99.0 (data visualization - bar, line, and core charts)

Dev Tools

- ESLint (code linting)
- TypeScript 5 (type checking)
- ML-Service (machine learning folder)
- Language: Python

Web Framework

- Flask 3.0.0 (REST API server)
- Flask-CORS 4.0.0 (cross-origin support)

Coding Libraries

- NumPy 1.24.0 (numerical computing)
- Pandas 2.0.0 (data manipulation)
- Scikit-learn 1.3.0 (machine learning algorithms)
- Joblib 1.3.0 (model persistence/serialization)