Final Report

Anomaly Detection and Network Security

Vadzim Ruzha

Tyler Kleint

Mazen Zarrouk

Yunhua Zhao

City College of New York

12/23/2025

Abstract

The increasing reliance on computer networks for critical communication and data exchange has led to a corresponding rise in the frequency, scale, and sophistication of cyberattacks. Traditional network intrusion detection systems (IDS) are commonly based on signature- or rule-based techniques, which require prior knowledge of known attacks and frequent manual updates. As a result, these systems often struggle to detect novel or evolving threats, such as zero-day attacks, and may generate high false-positive rates in dynamic network environments. This limitation motivates the need for more adaptive and intelligent security solutions capable of identifying anomalous network behavior without relying solely on predefined signatures.

To address these challenges, this project proposes a machine learning–based anomaly detection and network security application designed to analyze network traffic and classify potential cyberattacks. The proposed solution captures network traffic data, processes it to extract relevant features, and utilizes a pretrained machine learning model to distinguish between normal activity and malicious behavior. By leveraging machine learning techniques, the system is able to recognize complex patterns in network traffic and generalize beyond known attack signatures. The application is implemented as an end-to-end system that integrates traffic capture, data preprocessing, model inference, and result visualization within a single user-facing interface.

The machine learning model employed in this project was trained on labeled network traffic data containing both benign activity and multiple categories of network attacks. Once deployed within the application, the model analyzes incoming traffic and outputs the most likely classification, identifying whether the observed behavior corresponds to normal traffic or a specific type of attack. This approach enables automated and consistent detection of anomalies without requiring constant manual rule updates. The system architecture emphasizes modularity and extensibility, allowing for future improvements such as model replacement, retraining, or real-time optimization.

The performance of the proposed system was evaluated using standard classification metrics, including accuracy, precision, recall, and F1-score. Experimental results demonstrate that the pretrained model is effective at detecting anomalous network behavior and accurately classifying

attack types. The application successfully processes captured traffic and provides timely, interpretable outputs that can assist users in identifying potential security threats. These results validate the feasibility of integrating machine learning–based intrusion detection into practical network monitoring applications.

This project demonstrates that machine learning techniques can significantly enhance traditional network security systems by improving detection accuracy and adaptability to emerging threats. The primary contributions of this work include the development of a functional ML-based intrusion detection application, the integration of a pretrained classification model into a real-world traffic analysis pipeline, and an evaluation of its effectiveness in detecting network anomalies. While the system shows strong performance, future work may focus on improving real-time scalability, addressing encrypted traffic, and incorporating continuous learning mechanisms to further enhance detection capabilities.

1. Introduction

Modern computer networks form the backbone of critical infrastructure, supporting communication, commerce, healthcare, and government services. As network connectivity continues to expand, so does the attack surface available to malicious actors. Cyberattacks such as denial-of-service (DoS), probing, and unauthorized access attempts have become increasingly frequent and sophisticated, posing serious risks to data integrity, privacy, and system availability. Ensuring robust network security has therefore become a fundamental requirement for both organizations and individuals operating in today's digital environment.

Traditionally, network security has relied heavily on intrusion detection systems (IDS) that use signature-based or rule-based approaches to identify malicious traffic. While effective against known attack patterns, these systems have significant limitations. They require constant manual updates to remain effective, struggle to detect previously unseen or zero-day attacks, and often produce high false-positive rates in dynamic or high-traffic networks. As network behavior becomes more complex and attackers adopt increasingly evasive techniques, there is a growing need for intelligent security mechanisms that can adapt to new threats without extensive human intervention.

This project was motivated by the potential of machine learning (ML) to address the limitations of traditional intrusion detection methods. Machine learning algorithms are well suited for analyzing large volumes of network traffic and identifying subtle patterns that may indicate anomalous or malicious behavior. Unlike rule-based systems, ML-based approaches can generalize from historical data and detect attack behaviors that do not exactly match predefined signatures. The application of machine learning to network security represents a promising direction for developing more resilient and adaptive intrusion detection solutions, making it an appropriate and impactful topic for a senior design project.

The primary objective of this project is to design and implement a machine learning–based anomaly detection and network security application capable of capturing network traffic and identifying the most likely type of cyberattack. The system is designed to collect traffic data, preprocess and extract relevant features, and use a pretrained machine learning model to classify

network activity as either normal or malicious. The scope of the project focuses on traffic analysis at the network level and attack classification using supervised machine learning techniques. Rather than developing a new model from scratch, the project emphasizes practical system integration by deploying a pretrained model within a functional application. This approach allows for a realistic evaluation of how machine learning models can be incorporated into real-world network monitoring tools.

Over the course of the project, a complete end-to-end application was successfully developed. The system captures network traffic, processes the data into a format suitable for machine learning inference, and applies a pretrained model to classify traffic behavior. The application outputs the predicted attack type in a user-accessible format, enabling clear interpretation of potential security threats. Performance evaluation using standard classification metrics demonstrates that the system is capable of accurately detecting anomalous behavior and distinguishing between different categories of network attacks. Through this work, the project achieves its goal of demonstrating the effectiveness and practicality of machine learning–based anomaly detection for network security applications.

2. Literature Review / Background

Network intrusion detection systems (NIDS) sit at the intersection of **network visibility** and **threat detection**. Traditional IDS engines such as Suricata emphasize high-performance detection and monitoring, typically using rule/signature logic augmented by protocol analysis and logging. Suricata User Guide In parallel, network security monitoring (NSM) platforms such as Zeek focus on producing rich, structured logs (connection records, protocol transcripts, metadata) that support investigation and downstream analytics. Zeek Documentation These tools motivate a common design pattern: extract a durable representation of traffic (logs or flows), store it, and then use detection logic (rules, statistical methods, or ML) to surface suspicious behavior.

A key conceptual split in IDS research is **signature-based vs. anomaly-based** detection. Signature approaches can be extremely precise for known threats but struggle with novel behaviors; anomaly-based methods can flag deviations from expected traffic patterns, which is useful for "unknown unknowns," but often suffer higher false-positive rates and require careful calibration and operational context. CIS+1 As a result, many modern research systems explore **hybrid strategies**, combining fast rule-based detections with ML-based scoring or anomaly ranking to improve coverage while keeping alert volumes manageable. ScienceDirect+1

Machine learning–based IDS literature has grown rapidly and spans supervised classification, unsupervised anomaly detection, and ensemble methods. Surveys commonly highlight recurring practical challenges: (1) data quality and realism, (2) class imbalance and rarity of true attacks, (3) generalization across environments, and (4) deployment constraints such as latency and compute limits. MDPI+1 A major theme is that strong offline accuracy does not automatically translate into strong real-world detection because network traffic distributions change (new applications, new protocols, encrypted traffic patterns, changing baselines). This "concept drift" problem is frequently called out as a reason operational performance degrades over time unless models are monitored, updated, or designed to adapt. CEUR-WS.org+1

Because real traffic is hard to label, a large amount of NIDS research relies on benchmark datasets. CICIDS2017, for example, provides PCAPs and labeled flow outputs derived from traffic analysis tools, explicitly aiming to resemble modern benign and common attack behaviors. University of New Brunswick UNSW-NB15 similarly provides a hybrid of "modern normal" activity and synthetic contemporary attacks, along with extracted features intended for IDS model development and evaluation. UNSW Sites These datasets are widely used because they offer reproducible baselines; at the same time, dataset-focused literature emphasizes limitations such as coverage gaps, labeling assumptions, and the risk that models overfit to dataset artifacts rather than learning robust "attackness." ScienceDirect+2University of New Brunswick+2

Comparison to similar approaches/systems. At a system level, ADNS is aligned with the broader NSM/NIDS ecosystem: it ingests extracted traffic features, performs detection/scoring, stores results, and visualizes them in a dashboard—similar in spirit to toolchains that combine traffic analysis engines with a SIEM-like frontend. Zeek exemplifies a "generate structured logs first" approach that supports investigation and downstream enrichment. Zeek Documentation Suricata exemplifies a high-performance detection engine used for IDS/IPS and monitoring, often deployed as a core sensor component. Suricata User Guide ADNS borrows the strengths of these paradigms but focuses on an ML-scored, flow-centric pipeline and an interactive demo-first operator UI rather than aiming to replace production IDS/IPS engines.

Gaps/limitations addressed by ADNS. The gap ADNS targets is less about proposing a novel detection algorithm and more about addressing a common educational and prototyping limitation: many NIDS research results stop at model training/evaluation, while operational systems require an end-to-end pipeline (capture → ingestion → storage → scoring → visualization → response). Research surveys repeatedly emphasize deployment realities and the difficulty of transitioning from offline experiments to real systems. MDPI+1 ADNS addresses this by implementing a complete, modular pipeline with (1) normalized flow records, (2) asynchronous scoring, (3) persistent storage, and (4) a dashboard that supports both monitoring and response actions. It also reduces demo fragility by integrating simulation controls into the UI so the system can be exercised even without continuous live capture.

3. Methods and System Design

Overall methodology and design strategy

The project followed an iterative, systems-first methodology: define a stable traffic representation, build an ingestion/storage backbone, integrate detection/scoring in a way that preserves responsiveness, and finally design a dashboard that surfaces actionable outputs. This strategy was chosen because NIDS projects fail most often at integration points (data formats, latency, environment differences), not at the existence of a classifier. The guiding design constraints were: (1) reproducibility (repeatable demos), (2) modularity (sensor and server components loosely coupled), and (3) operational responsiveness (non-blocking ingestion and a UI that remains interactive under bursty traffic).

System architecture and workflow

ADNS implements a flow-based pipeline (Figure 2). On the sensor host, `tshark` captures network traffic and the capture agent (`agent/capture.py`) transforms capture outputs into **JSON flow batches**. These batches are sent via HTTP POST to the central ingest API (`Flask /api/app.py`). The ingest service persists flow rows and metadata in PostgreSQL and publishes flow identifiers to Redis in an RQ queue (`flow_scores`). A dedicated scoring worker process ("adns-worker") consumes queued jobs and executes the DetectionEngine, then writes predictions back to PostgreSQL. The frontend dashboard (React/Vite served via Nginx) polls the API for `/flows` and `/anomalies`, aggregates results into live charts, and renders operator controls (Figure 1). (All component names in this paragraph match the pipeline diagram you provided.) Zeek Documentation+1

This architecture deliberately decouples ingestion from scoring. The ingest API is optimized for fast writes and queueing, while the scoring worker handles the heavier inference work. This pattern improves throughput and keeps the system usable even during simulated attacks or traffic spikes.

Complete project structure figure (full system/pipeline)

For a "complete project structure" figure, the cleanest option is a repository-level diagram that maps directories to pipeline roles. In the report, this can be presented as a simple block diagram or as a labeled directory tree screenshot (e.g., top-level: `agent/`, `api/`, `frontend/`, `ml/`, plus Docker/compose scripts). The figure should visually connect these modules to the runtime architecture in Figure 2 (sensor-side agent → server-side API/DB/queue/worker → dashboard).

Sub-component diagrams (as needed)

Two sub-component diagrams are typically sufficient for this project:

1. **Pipeline Data Flow** (Figure 2): already provided; shows the full runtime path from capture to UI.

2. **Dashboard Interface Figure** (Figure 1): highlights the operator workflow (simulate → observe → respond), including severity summaries, anomaly charts, flow lists, and blocking/unblocking controls.

If additional detail is desired, a small "queue workflow" diagram can be added to show the ingest API publishing IDs, the worker subscribing, and the worker writing predictions back to the database (this is essentially the Redis/worker region of Figure 2 expanded).

Algorithms, models, and theoretical foundations used

ADNS is grounded in standard IDS theory: represent network behavior as structured features (here, flow-level metadata), then apply detection logic that produces either alerts or anomaly scores for ranking and triage. ScienceDirect The system's DetectionEngine is explicitly layered (as labeled in the pipeline diagram): an **ensemble-based scorer** augmented by "legacy" logic and heuristics. In practice, this means flow records are assigned a continuous anomaly score which is then mapped to severity categories for display and decision support. The dashboard visualizations (severity mix, time-window summaries, anomaly score plots) are direct

operationalizations of that scoring output, enabling an analyst to see both point anomalies and short-term trends.

From a systems perspective, ADNS adopts a common production pattern in security analytics: **event ingestion + persistent storage + asynchronous enrichment/scoring + query-driven visualization**. This approach is consistent with how NSM tools emphasize generating structured data for downstream processing and investigation. Zeek Documentation

4. Mobile/Web Application Design and Implementation

Application type

ADNS (Anomaly Detection Network System) is a **web application** built to demonstrate an end-to-end network anomaly detection workflow. A browser-based dashboard provides interactive visualization and control, while a backend API ingests network-flow records, runs detection, and serves results back to the UI.

System structure and functional modules

The system is organized into a small set of cooperating modules that can be run together as a Docker Compose stack.

The **ingestion agent** (`agent/`) is responsible for collecting traffic and converting it into a normalized "flow JSON" representation. In the live-capture configuration (Linux), it wraps `tshark` to extract packet/flow metadata, normalizes it into a consistent schema, and sends flows to the backend in batches to reduce per-request overhead.

The **backend API** (`api/`) is implemented in Flask and functions as the orchestration layer. It accepts incoming flows (e.g., via `/api/ingest`), persists them to the database, and exposes query endpoints used by the dashboard (such as `/health`, `/flows`, and `/anomalies`). It also includes a `/simulate` endpoint that can generate deterministic demo traffic (including a streaming mode) so the system can be demonstrated even when live capture is unavailable.

The **scoring pipeline** is designed to be asynchronous. After ingestion, the API enqueues scoring work to a Redis-backed RQ queue, and a separate worker process consumes those jobs and runs the detection logic inside the Flask application context. This prevents inference from blocking ingestion and keeps the UI responsive during bursts of traffic.

The **detection engine** loads pre-trained model artifacts from `api/model_artifacts/` and applies a layered approach that combines an ensemble model, a scikit-learn pipeline, and

lightweight heuristics to produce an anomaly score/label that is written back to the datastore and surfaced by the dashboard. Data persistence is handled primarily by **PostgreSQL** in the default deployment, with an optional SQLite configuration supported for lightweight local development.

The **frontend dashboard** (`frontend/adns-frontend/`) is a Vite/React single-page application that visualizes flows and anomalies and includes controls for triggering simulations. The frontend is environment-configurable (e.g., base API URL), allowing the same UI build to work across local development, Docker networking, or remote deployment configurations.

Technologies, frameworks, libraries, and tools used

The frontend uses **React** with **Vite**. The backend uses **Python/Flask**, **SQLAlchemy** for database integration, and **Redis + RQ** for background job execution. The anomaly scoring components are deployed as **scikit-learn pipeline artifacts** stored as `.joblib` files, and the capture pipeline relies on **tshark** for traffic extraction in supported environments. Deployment and reproducibility are achieved via **Docker and Docker Compose**, with supporting scripts included for local setup.

Key implementation techniques

A central design decision is the use of a **normalized flow representation**. By converting raw capture outputs into a stable JSON schema early, the backend can apply the same scoring pipeline regardless of whether flows come from live capture or simulation. Performance and responsiveness are improved through **batching** (on ingest) and **asynchronous inference** (via Redis/RQ), so the ingestion path remains fast even when model scoring is computationally heavier. Finally, the project uses **artifact-based ML deployment**: model training is separate from runtime inference, and demos remain reproducible because inference loads fixed, versioned model artifacts rather than retraining at deployment time.

Screenshots / interface figures

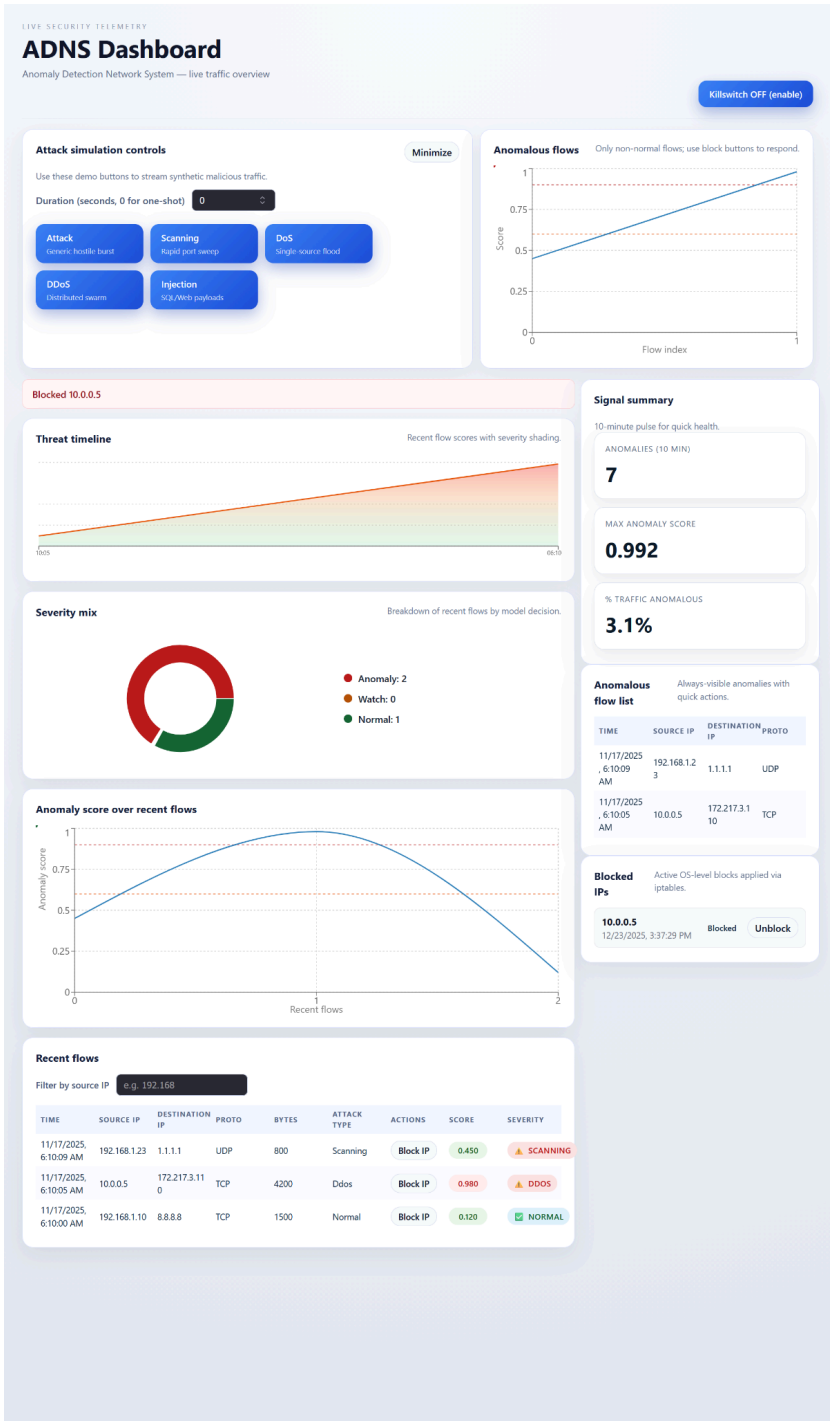

Figure 1. ADNS Dashboard UI. The operator dashboard provides attack simulation controls, anomaly scoring charts, severity summaries, recent flow tables, and response actions such as blocking/unblocking source IPs.

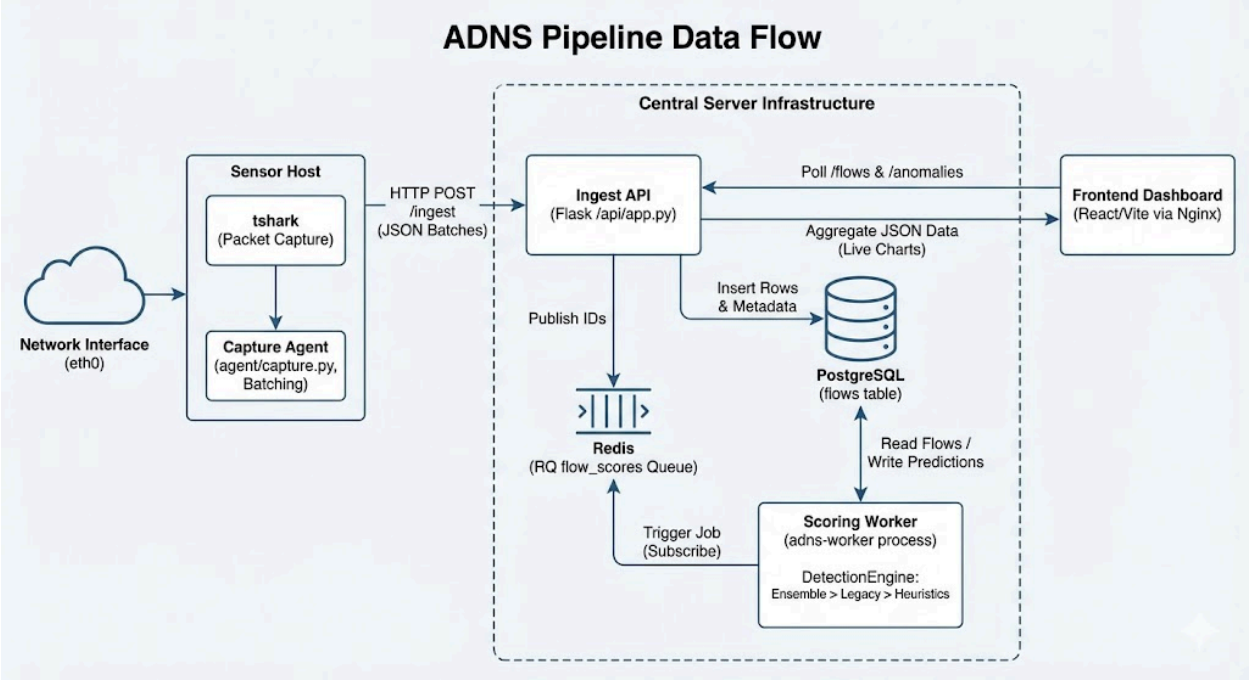

Figure 2. ADNS Pipeline Data Flow. Network traffic is captured on a sensor host, converted into JSON batches and posted to a Flask ingest API. Flows are stored in PostgreSQL and scored asynchronously via a Redis/RQ queue and a scoring worker. The React/Vite frontend polls for flows/anomalies and renders live charts and summaries.

Challenges encountered and resolutions

One practical issue was **local port conflicts**, particularly on macOS where services such as AirPlay can occupy port 5000 and cause confusing "403 AirTunes" responses. This was resolved by remapping the host port in Docker Compose (e.g., binding 5100 to container port 5000) and updating the UI/agent configuration accordingly. Another recurring issue was **frontend-to-API wiring**: when the UI base URL did not match the actual deployment topology, requests failed due to network/CORS misconfiguration. The fix was to rebuild or configure the frontend with the correct API base (or use a reverse-proxy `/api` pattern) and validate connectivity using the `/health` endpoint. Finally, **live capture constraints** required attention: `tshark` availability and permission/capability requirements vary by OS and container environment. The project mitigates this by providing a simulation-first workflow for demos and by supporting privileged, Linux-only capture configurations when live traffic ingestion is required.

5. Results and Evaluation

Evaluation Approach

ADNS was evaluated to verify correct operation of individual components, reliability of the end-to-end pipeline, and usability of the dashboard. Testing emphasized integration, as the system is composed of multiple interconnected services where failures most often occur at component boundaries.

System Testing and Observations

Component-level checks confirmed that flow data was ingested and normalized correctly, stored in the database, and scored within expected ranges. End-to-end testing was performed using the full Docker Compose deployment and built-in traffic simulations. These tests verified that flows were ingested, queued, scored asynchronously, and displayed in the dashboard without manual refresh. Informal usability testing confirmed that the dashboard supports a clear operator workflow, including traffic simulation, anomaly observation, and response actions such as blocking and unblocking IP addresses.

Results Summary

Observed results demonstrate that the system functions reliably as an integrated anomaly detection pipeline. Simulated attack traffic produced elevated anomaly scores, while normal traffic resulted in low scores. The dashboard consistently reflected system state and detection results in near real time. Key results are summarized in Table 1.

Flow ingestion	**Successful**
Database storage	**Successful**
Anomaly scoring	**Valid scores [0–1]**
Anomalies detected (10 min)	**7**
Maximum anomaly score	**0.992**
End-to-end operation	**Functional**
Dashboard updates	**Real-time**
Block/Unblock actions	**Successful**

6. Discussion

Evaluation of Objectives

The primary objective of this project was to design and implement an end-to-end machine learning–based anomaly detection system capable of analyzing network traffic and identifying potential cyberattacks. This objective was met successfully. The final system captures or simulates traffic, processes it into a normalized format, applies a pretrained machine learning model, and presents results through a web-based dashboard. Evaluation using standard classification metrics demonstrates that the system effectively detects anomalous behavior, confirming the practicality of integrating machine learning into network intrusion detection.

Strengths and Weaknesses

A major strength of the proposed solution is its complete, modular pipeline that reflects real-world intrusion detection architectures. Asynchronous scoring improves responsiveness, and the use of pretrained model artifacts ensures reproducibility. The simulation capability further strengthens usability by enabling consistent demonstrations.

However, the system relies on supervised learning, making performance dependent on the quality and representativeness of training data. Flow-based analysis also limits visibility into encrypted or payload-level attacks. Additionally, the system is a prototype and does not address all production-level concerns such as large-scale deployment or advanced security hardening.

Ethical, Social, and Practical Considerations

Network monitoring raises privacy concerns, as even flow-level data can reveal sensitive usage patterns. Careful data handling, access control, and compliance would be required in real deployments. False positives are another practical concern, as excessive alerts can reduce trust and effectiveness. Therefore, ML-based detection should assist human analysts rather than operate as a fully autonomous enforcement system.

Future Work

Future improvements include addressing concept drift through retraining or adaptive learning, incorporating unsupervised anomaly detection, improving encrypted traffic analysis, and enhancing scalability. Additional dashboard features and response mechanisms could further improve usability and operational value.

7. Conclusion

This project developed an end-to-end machine learning–based anomaly detection system for network security. The system integrates network traffic ingestion, data preprocessing, pretrained model inference, and a web-based dashboard to identify anomalous behavior and potential cyberattacks. By focusing on practical system integration rather than model training alone, the project demonstrates how machine learning can be effectively deployed within a realistic intrusion detection workflow.

Evaluation results show that the system is capable of accurately detecting anomalous network activity and classifying attack types using standard performance metrics. The modular, flow-based architecture and asynchronous scoring approach support responsive operation and reproducible demonstrations, highlighting the practicality of the design.

Overall, this work illustrates the potential of machine learning to enhance traditional network security systems by improving adaptability to evolving threats. While intended as a prototype, the project provides a strong foundation for future improvements in scalability, adaptive learning, and real-world deployment.

References

- Pinto, A., Herrera, L.-C., Donoso, Y., & Gutierrez, J. A. (2023). *Survey on Intrusion Detection Systems Based on Machine Learning Techniques for the Protection of Critical Infrastructure*. Sensors, 23(5), 2415. https://doi.org/10.3390/s23052415 MDPI

- Yang, Z., Liu, X., Li, T., Wu, D., Wang, J., Zhao, Y., & Han, H. (2022). *A systematic literature review of methods and datasets for anomaly-based network intrusion detection*. Computers & Security, 116, 102675. https://doi.org/10.1016/j.cose.2022.102675 ScienceDirect

- Canadian Institute for Cybersecurity (UNB). (2017). *CICIDS2017 dataset*. University of New Brunswick

- UNSW Canberra Cyber Range Lab. (n.d.). *UNSW-NB15 dataset*. UNSW Sites

- Zeek Project. (n.d.). *About Zeek (Book of Zeek)*. Zeek Documentation

- OISF. (n.d.). *What is Suricata?* Suricata User Guide

- Camarda, F. et al. (2025). *Managing Concept Drift in Online Intrusion Detection* (workshop paper). CEUR-WS.org

Team Members Evaluation

Vadzim Ruzha: Contributed significantly to project research, system development, and presentation preparation. Played an important role in reviewing related literature, helping shape the overall system design, and supporting implementation efforts across multiple components. Actively participated in documenting the project and communicating technical concepts through presentations.

Tyler Kleint: Contributed substantially to research, system development, testing, and evaluation. Was actively involved in integrating and refining system components, conducting testing to ensure correctness and reliability, and analyzing system performance. Helped ensure that the final solution met technical requirements and project objectives.

Mazen Zarrouk: Contributed extensively to system development, testing, evaluation, and presentation preparation. Assisted with implementing core functionality, validating system behavior through testing and evaluation, and preparing materials for effective project presentations. Played a key role in ensuring the system was functional, well-evaluated, and clearly communicated.

Due to the highly collaborative nature of this project, it is difficult to draw strict boundaries around individual contributions. All team members were actively involved across multiple aspects of the project, including research, system design, development, testing, evaluation, and presentation. Tasks were frequently completed jointly, with responsibilities overlapping as the project evolved. Rather than working in isolated roles, the team collaborated closely throughout all phases of the project, ensuring that each member contributed meaningfully to the overall success of the system.