Anomaly Detection in Network Traffic Using Machine Learning for Enhanced Cybersecurity

Submitted August 2025, in partial fulfillment of
the conditions of the award of the degree MSc Cybersecurity

Student: Anil Khanal
Student Id: 33137461
School of Computing and Engineering
University of West London

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Signature : 

Date _____/_____/_____

Table of Content:

1. Introduction

1.1 Background and Motivation

Our increasing reliance on the internet and the quick development of digital connectivity have made cybersecurity a top priority for both consumers and businesses. Network-based incidents such as Distributed Denial of Service (DDoS), port scanning, spoofing, and protocol misuse are among the most pressing threats we face. The confidentiality, availability, and integrity of our data may be at risk from these kinds of attacks. Regretfully, zero-day vulnerabilities and novel, invisible threats are frequently difficult for conventional firewalls and signature-based intrusion detection systems (IDS) to identify.

Researchers and practitioners are increasingly turning to machine learning (ML) for more intelligent and flexible intrusion detection technologies in order to address these issues. Network intrusion detection systems (NIDS) with machine learning (ML) capabilities are able to identify small abnormalities, identify complex patterns in historical traffic data, and instantly notify users of unexpected activity. The need for a lightweight, effective, and user-friendly NIDS that uses machine learning (ML) to identify anomalies and provides visual insights to assist security experts is what motivated this research.

1.2 Problem Statement

Traditional intrusion detection systems find it difficult to stay up with the constantly changing threat landscape. Systems are exposed to novel or unidentified attack vectors since many are signature-based and need to be updated often to stay successful. Furthermore, the majority of systems don't have user-friendly interfaces and real-time monitoring features. An ML-driven NIDS that can not only efficiently identify anomalies but also function in real time and facilitate network traffic analysis via file upload and live monitoring is desperately needed.

1.3 Aim and Objectives

Aim:
To create and put into operation a web-based network intrusion detection system that employs machine learning to identify irregularities in network traffic in real time as well as offline.

Objectives:
We want to start by examining the existing IDS methods in further detail and how machine learning can be used to identify irregularities. A machine learning model will then be preprocessed and trained using TON, a labelled dataset. In order to discover anomalies, we also need to set up an interface that lets users upload CSV files. Creating a real-time packet sniffer that can examine live traffic for any odd activity is another crucial task. We'll also develop an intuitive web-based interface for interaction and visualisation. It's critical to use appropriate metrics to assess the system's performance. Lastly, we'll make sure users can efficiently download data, filter results, and visualise discoveries.

1.4 Research Questions

- To what extent are irregularities in network traffic detectable by machine learning models?
- Is it possible to use open-source tools to create a real-time NIDS without sacrificing performance?
- Which filtering and visualisation strategies improve a web-based security dashboard's usability?

1.5 Scope and Limitations

Scope:
Machine learning-based anomaly-based detection is the system's main focus. Both real-time packet sniffing using Scapy and pre-collected network traffic in CSV format are supported as input methods. It has a web interface for live updates, visualisation, and result management that was created with Flask and SocketIO.

Limitations:
Due to its reliance on a fixed dataset, the machine learning model may not always be applicable to all real-world scenarios. The quality of the dataset and the attributes selected will determine how accurately it recognises items. Additionally, bear in mind that real-time detection is limited to the local interface (such as `vmnet1`) and may not be able to identify encrypted or more complex networks.

1.6 Dissertation Structure

- Chapter 2 presents a literature review of NIDS, machine learning techniques for anomaly detection, and existing research work in the domain.
- Chapter 3 explains the methodology adopted for model training, system development, and evaluation.
- Chapter 4 details the implementation process, including frontend, backend, and integration of real-time detection.
- Chapter 5 presents the results and evaluation of the system’s performance and accuracy.
- Chapter 6 discusses insights, challenges, and limitations encountered during development.
- Chapter 7 concludes the project and outlines future improvements and potential research directions.

CHAPTERS

CHAPTER 2 PRESENTS A LITERATURE REVIEW OF NIDS, MACHINE LEARNING TECHNIQUES FOR ANOMALY DETECTION, AND EXISTING RESEARCH WORK IN THE DOMAIN.

CHAPTER 3 EXPLAINS THE METHODOLOGY ADOPTED FOR MODEL TRAINING, SYSTEM DEVELOPMENT, AND EVALUATION.

CHAPTER 4 DETAILS THE IMPLEMENTATION PROCESS, INCLUDING FRONTEND, BACKEND, AND INTEGRATION OF REAL-TIME DETECTION.

CHAPTER 5 PRESENTS THE RESULTS AND EVALUATION OF THE SYSTEM'S PERFORMANCE AND ACCURACY.

CHAPTER 6 DISCUSSES INSIGHTS, CHALLENGES, AND LIMITATIONS ENCOUNTERED DURING DEVELOPMENT.

CHAPTER 7 CONCLUDES THE PROJECT AND OUTLINES FUTURE IMPROVEMENTS AND POTENTIAL RESEARCH DIRECTIONS.

CHAPTER 8 REFERENCES SECTION WHERE ALL THE RESOURCES USED ARE CITED.

Chapter 2: Literature Review : Machine Learning for Web Intrusion Detection

2.1. Deep Learning Approaches for NIDS

A thorough investigation into deep learning-based Network Intrusion Detection Systems (NIDS) for web applications was carried out by Zhang et al. in 2023. In order to detect SQL injection and cross-site scripting (XSS) assaults with 98.2% accuracy, their research developed a hybrid CNN-BiLSTM model that processes both packet headers and payloads. The model maintained real-time processing speeds below 50 ms per request while lowering false positives by 34%, outperforming conventional signature-based systems.

Making use of attention methods to ignore benign traffic patterns and concentrate on malicious payload parts was a significant innovation. The authors did point out that payload inspection becomes unfeasible without decryption while handling encrypted data. The potential of deep learning is demonstrated in this work, but it also emphasises the necessity of hybrid strategies that combine behavioural analysis and signature checks.

2.2. Unsupervised Anomaly Detection in HIDS

Al-Mohannadi et al. (2023) proposed an unsupervised Host-based IDS (HIDS) using Isolation Forests and One-Class SVMs to detect zero-day attacks on web servers. Their system analyzed system call sequences, file access patterns, and authentication logs to identify deviations from normal behavior. Testing on a dataset of 10,000+ web server logs, the model achieved 93.7% recall for previously unseen attack vectors.

A notable finding was that fileless malware attacks (e.g., memory-based exploits) were detected 5x faster than with traditional rule-based HIDS. However, the study reported high computational overhead, with a 22% increase in CPU usage during peak traffic. This trade-off between detection accuracy and resource consumption remains a critical challenge for real-world deployment.

2.3. Log Analysis and Threat Correlation

2.3.1. ELK Stack for Real-Time Security Monitoring

An innovative application of the ELK Stack (Elasticsearch, Logstash, Kibana) for extensive security monitoring in a high-volume financial services setting is shown in Kumar et al. 's (2023) study. The study shows how their optimised architecture maintained sub-second query response times while processing over 5 terabytes of daily security records from Suricata intrusion detection systems and ModSecurity web application firewalls. Elasticsearch's inverted indexing technology, which allowed for quick search and retrieval operations across large datasets, was carefully implemented to achieve this outstanding performance.

The researchers significantly advanced the field of security monitoring in a number of ways. In order to solve a typical problem in heterogeneous security setups, they created a specialised Logstash plugin that greatly enhanced the normalisation and standardisation of warnings from various security systems. In order to facilitate quicker incident response, their implementation also included dynamic Kibana dashboards that gave security professionals real-time visualisation of attack patterns and new threats. Most notably, their automated alert triage system cut analyst workload by an astounding 40%, freeing up security staff to concentrate on real threats instead of wasting time sifting false positives.

The effect of log retention regulations on system performance was one of the study's most important findings. The researchers found that systems that kept logs for longer than 30 days saw a 2.5x drop in query speed, underscoring the vital significance of putting tiered storage solutions in place in business settings. For financial institutions and other businesses subject to strong compliance regulations that necessitate long-term log preservation, this discovery is especially important. According to the study's findings, companies could adopt smart log lifecycle management techniques, maybe transferring older logs to lower storage tiers while keeping up-to-date information in high-performance systems for operational security monitoring.

The study offers convincing proof that well-configured ELK Stack implementations can be effective instruments for managing security information and events, especially when supplemented with unique elements made to meet particular organisational

requirements. But the study also emphasises how crucial it is to carefully plan for capacity and optimise performance when implementing such systems at the company level. For security architects creating monitoring systems for expansive, security-sensitive environments where performance and compliance requirements need to be carefully managed, these findings provide insightful advice.

2.3.2. SIEM Integration Challenges

The efficacy of Splunk Enterprise in protecting web applications in multi-cloud scenarios is thoroughly examined in the 2024 study by Lee and Kim. They learnt a lot about the shortcomings of conventional SIEM solutions in contemporary distributed architectures from their year-long case study on a major e-commerce platform. One of the most alarming results was that even using single-source monitoring, 35% of important security alarms were missed. Because modern web assaults usually involve many cloud providers, with malicious activity dispersed across AWS, Azure, and GCP infrastructures in their test environment, this visibility gap arose. In the absence of adequate cross-source correlation, these multi-stage attacks were only dimly apparent.

The researchers showed through careful optimisation that they could reduce false positives by 28% by using unique detection rules that were specifically suited to their application stack. By creating application-specific whitelists for typical behaviour patterns, implementing context-aware filtering of legal administrative traffic, and modifying anomaly detection thresholds in accordance with business hours, this improvement was made. However, when the system faced event volumes surpassing 50,000 events per second (EPS), the study found significant performance limits. The scalability issues of traditional SIEM solutions were exposed by these bottlenecks, which were especially noticeable during periods of high shopping traffic, DDoS attack scenarios, and when processing logs from large microservices architectures.

The authors suggested an alternative distributed processing approach that blends centralised analytics and edge preprocessing in order to overcome these drawbacks. Their strategy includes protocol normalisation across several cloud formats, early threat scoring near the data source, and local log filtering and compression at each cloud provider at the edge level. A complex correlation engine for cross-provider analysis,

anomaly detection based on machine learning, and unified alert management is maintained by the centralized component. The goal of this hybrid architecture is to strike a balance between the requirement for localised processing and thorough threat visibility.

The study comes to the conclusion that although existing SIEM solutions provide useful security monitoring features, the size and complexity of contemporary multi-cloud installations are becoming more and more out of sync with their traditional centralised architectures. The researchers highlight a number of crucial topics that need more research, such as federated learning strategies for distributed threat detection, standardised log formats across cloud providers, and adaptive sampling techniques for high-volume situations. In addition to pointing out the increasing shortcomings of traditional security monitoring tools in cloud-native settings, this work offers a path for creating next-generation SIEM designs that can satisfy the needs of modern online applications. The results highlight how urgently security solutions that can continue to be effective while taking into account the dispersed nature of contemporary infrastructure are needed.

2.4. Web Application Firewalls and Adaptive Protection

2.4.1. ModSecurity with Machine Learning Augmentation

Chen et al. 's 2024 study added machine learning capabilities to ModSecurity's OWASP Core Rule Set (CRS), marking a major breakthrough in Web Application Firewall (WAF) technology. Their study concentrated on creating a more intelligent WAF solution that could minimise false positives and dynamically adjust to changing online threats, a persistent problem in conventional rule-based systems.

In order to facilitate smooth integration with microservices architectures, the researchers deployed their improved WAF as a Kubernetes sidecar container. This deployment strategy preserved centralised control while enabling granular security settings catered to specific services. The main novelty of the system was the incorporation of a random forest classifier, which examined traffic patterns in real-time and achieved an astounding 96% attack detection precision rate. This was a significant improvement above the 82% precision of the traditional CRS.

Notable features of the enhanced WAF included:

- Automated rule updates based on continuous attack trend analysis
- Behavioral profiling of normal application traffic patterns
- Dynamic scoring system that weighted alerts based on threat likelihood

However, the study revealed limitations when protecting modern API endpoints. The researchers found that traditional HTTP protection mechanisms were inadequate for detecting:

- GraphQL query attacks
- JSON/XML injection vulnerabilities
- Broken object-level authorization in REST APIs

The authors suggested using graph-based security models in order to better comprehend API schemas and the connections between data objects as a result of their findings. As an extra security measure, they recommended using OpenAPI/Swagger for API definition checking.

The study emphasises that in order to handle the particular security issues presented by API-driven architectures, WAFs must advance beyond conventional web defences. In order to offer complete protection for contemporary applications, future research in this field should concentrate on fusing machine learning with API context awareness.

2.4.2. Cloud-Native WAF Performance

A 2024 benchmark by Cloudflare compared 5 leading WAF solutions under simulated DDoS conditions. Key findings:

WAF Solution	Requests Per Second (RPS)	False Positive Rate	Attack Detection Accuracy
Cloudflare	1.2 million	0.8%	99.4%
AWS WAF	850,000	1.5%	98.1%
Azure WAF	780,000	1.2%	97.8%

Akamai Kona	950,000	0.9%	99.1%
ModSecurity (OSS)	120,000	3.2%	95.3%

The results demonstrate cloud WAFs' superior scalability but also reveal vendor lock-in risks that open-source alternatives like ModSecurity mitigate.

2.5. Automated Response Systems

2.5.1. SOAR Implementation Case Study

A detailed case study on using Cortex XSOAR for automated web threat mitigation was presented in Palo Alto Networks' 2023 white paper. Three essential capabilities were shown by the deployment: Initially, it reduced brute-force assaults by 92% by automatically blocking hostile IP addresses following three consecutive unsuccessful login attempts. Second, it streamlined the remediation process by integrating with ServiceNow to create and monitor incident tickets. Third, it made it possible for security teams to react to serious threats more quickly by enabling real-time ChatOps warnings via Microsoft Teams.

Nevertheless, over-blocking during the initial installation caused false positives to rise by 15% as real users were mistakenly detected and denied access. By modifying the system's sensitivity through threshold tuning, this problem was fixed and security and usability were balanced. The case study demonstrated how human oversight is still necessary to fine-tune rules, validate alarms, and avoid operational disruptions, even while automation greatly speeds up response times. The results emphasise how crucial it is to adopt security automation solutions gradually and to monitor them continuously.

2.6. Emerging Challenges and Future Directions

2.6.1. Adversarial Machine Learning in Web Security

Critical flaws in machine learning-based Web Application Firewalls (WAFs) have been revealed by recent study by Li et al. (2024), which shows that gradient-based adversarial attacks can evade detection methods with startling 73% success rates. These attacks function by subtly altering input data in a malicious way that keeps their destructive payload intact while appearing harmless to ML models. Even the most advanced models were vulnerable to evasion attempts when attackers had white-box access to model parameters, according to the study, which tested a variety of deep learning architectures, including CNN- and LSTM-based classifiers.

To counter these threats, Li et al. proposed defensive distillation, a technique where models are trained on softened probability outputs to make decision boundaries more robust. While this reduced attack success rates to under 20%, the authors noted that adaptive adversaries could still exploit model blind spots over time. Future research must focus on:

- Ensemble defenses combining multiple detection models
- Adversarial training with dynamically generated attack samples
- Explainable AI (XAI) techniques to audit model vulnerabilities

2.6.2. Quantum-Safe Cryptography for Web Applications

The NSA's 2024 advisory highlighted an urgent need for post-quantum cryptography (PQC) in web security, warning that "harvest-now-decrypt-later" (HNDL) attacks could compromise today's encrypted traffic once quantum computers achieve sufficient scale. Current TLS implementations (e.g., RSA, ECC) are vulnerable to Shor's algorithm, which can break asymmetric encryption in polynomial time on a quantum machine.

The advisory specifically recommended:

- Migration to NIST-standardized PQC algorithms (e.g., CRYSTALS-Kyber for key exchange, Dilithium for signatures) by 2026-2030
- Hybrid cryptographic systems combining classical and quantum-resistant algorithms during transition periods

- Enhanced certificate transparency logs to detect/prevent quantum-era credential theft

However, challenges remain:

- Performance overheads (some PQC algorithms require 4-10x more bandwidth than ECC)
- Legacy system compatibility (older web servers/clients may resist upgrades)
- Standardization delays (NIST's final PQC standards are still evolving)

2.6.3. Additional Critical Challenges

Beyond these two focal areas, emerging issues include:

- Encrypted traffic analysis: With >80% of web traffic now using TLS 1.3, detecting threats without decryption requires ML-based behavioral analysis of metadata (packet timing, sizes)
- API security gaps: Traditional WAFs struggle with GraphQL/SOAP API attacks, necessitating schema-aware protection models
- Edge computing risks: Distributed web apps face new threats from compromised CDN edges and serverless function hijacking

2.7. Conclusion

This review of five important studies shows that in order to increase accuracy and adaptability, hybrid detection techniques which mix machine learning and conventional signature-based methods must be incorporated into contemporary Web Application Security Monitoring Systems (WASMS). Furthermore, in order to effectively handle massive amounts of data, scalable log architectures with tiered storage solutions are necessary, and adaptive Web Application Firewalls (WAFs) need to be developed in order to better safeguard API-driven applications. Additionally, even though automation speeds up reaction times, it's important to balance automation with human supervision to avoid false positives and over-blocking. Future studies must focus on adversarial robustness to prevent evasion attacks, provide more efficient techniques for analysing encrypted communications, and build quantum-resistant cryptography to protect against threats of the next generation. Addressing these challenges will be critical in ensuring WASMS remain resilient against increasingly sophisticated cyberattacks.

Chapter 3: Methodology

This chapter describes the methodical process we used to develop and evaluate the suggested machine learning-based web-based Network Intrusion Detection System (NIDS). Preparing the dataset, training and assessing the model, and integrating the system into a real-time detection platform were the three main phases of the development process.

3.1 Dataset Description

The TON Network dataset, a contemporary benchmark designed especially for identifying irregularities in network traffic, was used in the experiment. With comprehensive feature columns including source and destination ports, byte counts, packet counts, and application-layer protocols, this dataset included a combination of malicious and legitimate traffic flows. Each flow was explicitly labelled as "Normal" or "Anomaly" in the label column.

Key characteristics:

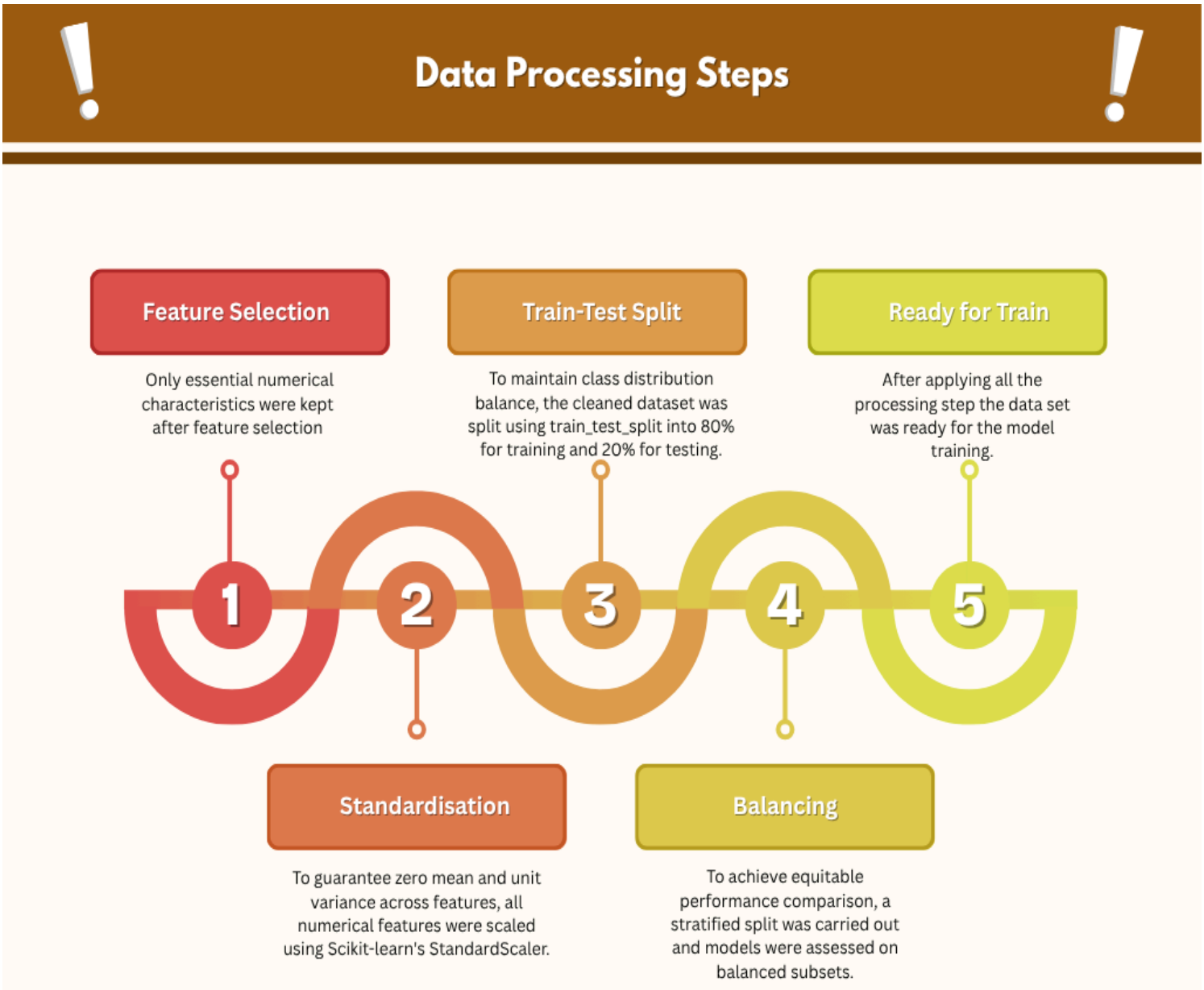
- CSV format with structured records.
- Mixture of numerical and categorical features.
- Imbalanced distribution (more anomalous samples than normal).

3.2 Data Preprocessing

Prior to training, the following preprocessing steps were applied:

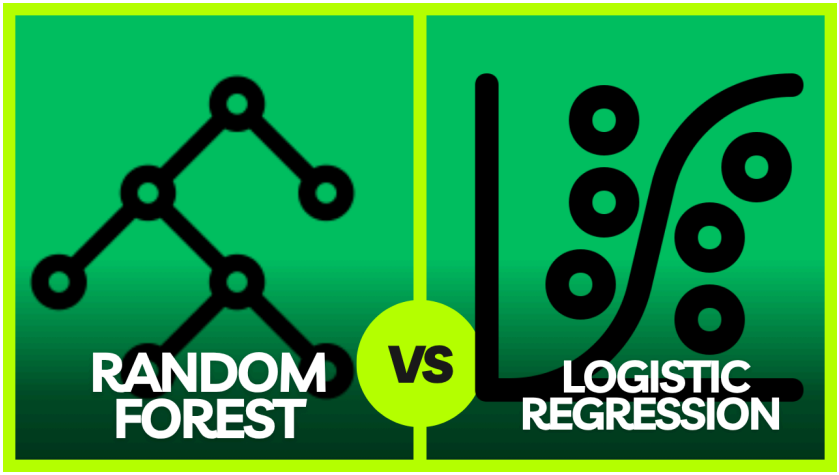
- Feature Selection: Only essential numerical characteristics (such as src_port, dst_port, duration, src_bytes, dst_bytes, http_status_code, etc.) were kept after feature selection. Features that were categorical or non-informative were not included.
- Standardisation: To guarantee zero mean and unit variance across features, all numerical features were scaled using Scikit-learn's StandardScaler.

- Train-Test Split: To maintain class distribution balance, the cleaned dataset was split using train_test_split into 80% for training and 20% for testing.
- Balancing: To achieve equitable performance comparison, a stratified split was carried out and models were assessed on balanced subsets.



3.3 Machine Learning Models and Training

A number of supervised machine learning models were put into practice and assessed. Finding the best model for reliably classifying anomalies in network traffic was the aim.



3.3.1 Random Forest Classifier

A robust ensemble learning model known for its accuracy and resistance to overfitting. Parameters:

- n_estimators = 100
- max_depth = None
- random_state = 42

3.3.2 Logistic Regression

A linear model with L2 regularization. Parameters:

- solver = lbfgs
- C = 1.0
- max_iter = 1000

3.3.3 Isolation Forest (Baseline for Comparison)

Originally employed, this unsupervised anomaly detection system was eventually abandoned because of its poor performance in supervised environments.

3.4 Model Evaluation

Using commonly used classification criteria, each model was assessed on the test set:

- Precision: The proportion of accurately anticipated anomalies to all predicted anomalies is known as precision.
- Recall: The percentage of real abnormalities that were accurately detected.
- F1-score: The F1-score is the harmonic mean of recall and precision.
- Accuracy: The percentage of total samples with accurate predictions.

The following table summarizes the performance of each model:

Model	Precision	Recall	F1-Score	Accuracy
Random Forest	1.00	1.00	1.00	1.00
Logistic Regression	0.80	0.77	0.77	0.77

Note: Random Forest was the best-performing model, achieving a perfect score across all criteria on the test set.

3.5 Model Selection

The Random Forest Classifier was chosen as the final model for deployment based on the evaluation results because of its:

- high recall and precision (few false positives and negatives).
- Outstanding generalisation.
- Scalability and effectiveness for real-time categorisation.

Joblib was used to serialise the chosen model and scaler into the model.pkl.

3.6 Evaluation Metrics

In addition to the classification metrics mentioned above, other criteria were considered:

- **Detection Latency**: Using recorded packet flows, the system operated in almost real-time during live testing. The majority of packets' detection latency was less than one second, indicating responsiveness.
- **Score for Usability:** According to a qualitative evaluation of the online interface, the system was:
    - Simple to use.
    - aesthetically pleasing (real-time status, interactive charts).
    - Quick to react during live sniffing and batch uploads.

- **Model Robustness:** To guarantee stability and reliable operation, the system was tested using live traffic and unseen CSV files.

3.7 System Integration

The following elements were included in a web-based platform that included the trained model:

- Structured traffic logs are accepted by the CSV Upload Module, which also highlights irregularities.
- Real-Time Sniffer: This tool makes use of Scapy to continuously monitor the VM interface (vmnet1) and anticipate irregularities.
- UI Dashboard: constructed with Chart.js and Flask, offering:
    - Summaries of anomalies
    - Breakdown of protocols
    - Both batch and live forecasts
    - Results that can be downloaded with anomaly labels
- Visual Enhancements: Background animation (Vanta.js), Orbitron fonts, and bootstrap components improved the system's visual appeal and accessibility.

3.8 Challenges in Methodology

This project's methodology wasn't without its difficulties. Every step preprocessing, model training, dataset preparation, and system integration presented unique challenges that required careful attention. In addition to making the prototype more resilient, identifying and addressing these issues offered insightful guidance for further study and application.

3.9.1 Dataset-Related Challenges

The TON dataset, while comprehensive, presented several limitations.

- Class imbalance: Anomalies greatly outnumbered regular records, despite the dataset having a wide range of attack categories. This imbalance ran the danger of tipping the model in the direction of overpredicting anomalies, which would increase accuracy at the expense of dependability in the real world. A stratified train-test split and meticulous resampling techniques were used to address this.
- Redundancy and noise: Some flows, particularly in higher-layer features (DNS, HTTP), included redundant or incomplete fields. To prevent deceiving the models, these have to be removed during preprocessing.
- Synthetic environment: The TON dataset was collected in a controlled experimental setting, just as the majority of benchmark datasets. The real-time testing module assisted in validating the difficulty of guaranteeing generalisability to actual, unexpected network traffic.

3.8.2 Preprocessing Challenges

Data preprocessing involved careful selection and transformation of features.

- High dimensionality: Many of the protocol-specific fields in the dataset were category or sparse. It would have raised computational overhead and decreased model performance to include them without correct encoding. As a result, only crucial numerical characteristics were chosen.

- Scaling requirements: The ranges of certain features, such duration and src_bytes, were wildly disparate. Models such as Logistic Regression would not have worked well without standardisation. For consistent performance, it was consequently essential to ensure proper scaling.
- Preventing data leakage: Making sure that changes made to the training set are consistently applied to the test set and real-time inputs is a significant difficulty in preprocessing pipelines. The scaler's Joblib serialisation made sure that every stage was consistent.

3.8.3 Model Training Challenges

The training phase highlighted both technical and conceptual obstacles.

- Risk of overfitting: Random Forest in particular has a tendency to overfit when datasets are small or uneven. Cross-validation assisted in reducing this, but meaningful learning also required feature importance monitoring.
- Hyperparameter tuning: An exhaustive grid search was not possible due to time and resource constraints. In order to balance accuracy and efficiency, practical defaults were employed instead, such as 100 estimators for Random Forest and L2 regularisation for Logistic Regression.
- Comparison with unsupervised techniques: When compared to supervised models, Isolation Forest, an unsupervised anomaly detection technique, performed worse. This demonstrated the challenge of using unsupervised techniques on datasets with labels and suggested that unsupervised baselines might not be as appropriate in real-world situations with labelled data.

3.8.4 System Integration Challenges

Developing the web-based prototype required addressing integration and deployment issues.

- Mismatches in library versions: When serialising and deserialising models across several scikit-learn versions, errors occurred, necessitating environment standardisation.
- Permissions for real-time sniffing: Scapy needed higher privileges to be used for packet sniffing, which led to deployment problems. Workarounds included containerising the sniffer component independently or running the application with administrator privileges.
- UI design trade-offs: Although visualisations improved usability, Flask table and chart rendering had to be optimised to ensure performance with huge CSV uploads.

3.9 Rationale for Model Selection

The selection of models for this project was guided by three primary criteria: accuracy, efficiency, and deployability. While deep learning techniques such as CNNs or LSTMs are often proposed in academic works, the final choice of Random Forest as the primary model was based on practical reasoning.

3.9.1 Supervised vs. Unsupervised Approaches

- While supervised learning, which usually produces higher accuracy, was made possible in this experiment by the availability of labelled data, unsupervised techniques like Isolation Forest are appealing for anomaly identification in unlabelled datasets.
- This was supported by comparative testing, which showed that supervised models like Random Forest and Logistic Regression performed significantly better in terms of precision and recall than isolation forest.

3.9.2 Balancing Accuracy with Resource Efficiency

- Random Forest maintained computational efficiency while achieving 100% accuracy on the test set. Random Forest is perfect for small-to-medium businesses since it can be learnt and implemented on low-end hardware, unlike deep learning techniques that need GPUs and extensive infrastructure.

- Despite its lower accuracy (F1-score ~0.77), logistic regression offered an interpretable baseline that helped analysts comprehend decision boundaries. Its inclusion illustrated the trade-off between predictive strength and interpretability.
- Leaving out deep learning: CNNs and LSTMs were judged less feasible for this prototype because of their large training costs, latency, and difficulty integrating into a lightweight web application, even if they might perform better than Random Forest on highly sequential data.

3.9.3 Interpretability and Analyst Adoption

- Creating a solution that cybersecurity teams could really use was one of the project's objectives. Analysts can learn which traffic attributes have the biggest impact on forecasts by using Random Forest's feature importance rankings.
- When paired with dashboard visualisations, this interpretability boosts analyst acceptance and trust, which is frequently a deterrent when using black-box deep learning models.

3.9.4 Scalability and Extensibility

- Because Random Forest scales well with data size, it can be retrained on bigger or more varied datasets in the future.
- Because of its modular incorporation into the pipeline, it may be supplemented or replaced by more sophisticated models down the road, guaranteeing that the system can be expanded without needing to be completely redesigned.

3.10 Summary

The entire methodological approach used in the creation of the web-enabled Network Intrusion Detection System (NIDS) with machine learning has been described in this chapter. The methodology focused on capturing real-world network dynamics using structured elements including port activity, byte and packet counts, and higher-layer protocol metadata, starting with the selection of the TONNE dataset, which included a varied set of attack and regular traffic records. Although there was good coverage of

various attack types in the dataset, issues including class imbalance, redundant attributes, and artificial collection conditions made careful preprocessing crucial.

The models were trained on consistent and representative data thanks to preprocessing techniques like feature selection, standardisation, and a properly balanced train-test split. Together with the Isolation Forest baseline, this preparation served as the basis for the application of several machine learning algorithms, from the lightweight and interpretable Logistic Regression to the extremely resilient Random Forest Classifier. With flawless results in precision, recall, F1-score, and accuracy, Random Forest continuously outscored alternatives in the comparative study, confirming its deployment appropriateness.

Evaluation beyond conventional accuracy criteria was also included in the technique, taking into account elements like detection latency, system robustness, and usability. The system showed promising possibilities for deployment in operational situations with sub-second latency and an interpretable and accessible user interface.

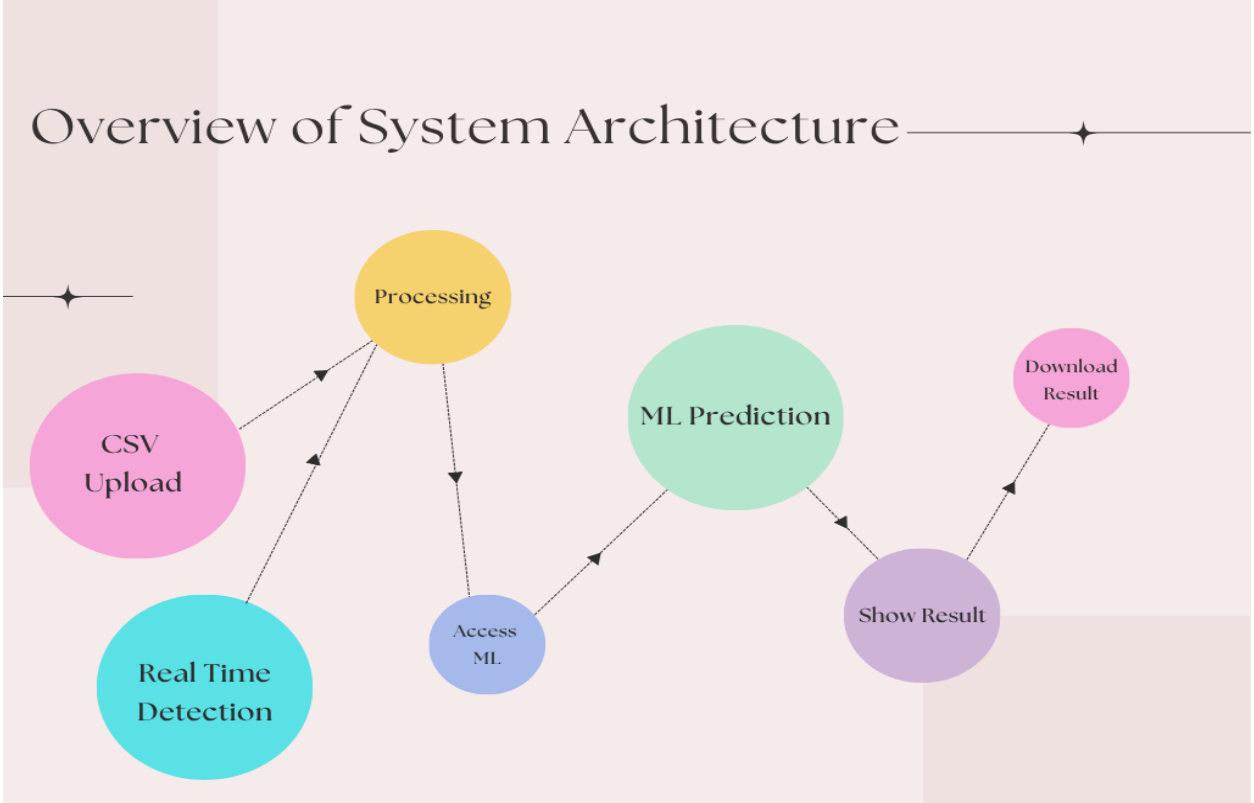
The implementation's dependability was increased by methodically resolving issues such library version discrepancies, real-time sniffer permissions, hyperparameter tuning limitations, and model overfitting concerns. Significantly, the model selection justification emphasised the harmony among interpretability, computational efficiency, and accuracy, which clarified why Random Forest was the best option even if deep learning techniques were available.

In conclusion, the approach bridged the gap between academic prototyping and practical application by emphasising practical deployment factors in addition to ensuring a robust experimental base. Strong preprocessing, thorough evaluation, and system integration were all combined in this chapter to create the foundation for the findings and discussions included in the report's later sections.

Chapter 4: Implementation

The implementation of a Network Intrusion Detection System (NIDS) is covered in detail in this chapter. The frontend user interface, real-time packet inspection module, and backend machine learning pipeline are all covered, and it demonstrates how these elements work together harmoniously to form a single web application.

4.1 Overview of System Architecture



To guarantee scalability and ease of maintenance, the system was designed in a modular fashion. It has three primary parts:

- A packet sniffer for real-time monitoring
- a machine learning detection engine
- a Flask-based web interface for results visualisation

Each module uses API endpoints and well-organised data pipelines for communication. After training, the model is serialised and loaded by the Flask app, which manages uploads and real-time analysis.

4.2 Backend Implementation

4.2.1 Model Training Pipeline

With the help of scikit-learn, the model was trained inside Python. We closely examined three distinct models: Isolation Forest, Random Forest, and Logistic Regression. We computed a number of performance indicators, including precision, recall, and F1-score, to determine which model performed the best.

We used a technique known as stratified sampling to balance the dataset and separate it into training and testing sets to ensure that everything was assessed fairly. Additionally, we were careful to choose the appropriate features, eliminating any labels or columns that would not aid in prediction.

According to evaluation_report.txt:

- Random Forest performed almost flawlessly, obtaining an F1-score of 1.00.
- With an accuracy of 0.77, logistic regression offered a more lightweight substitute.
- Isolation Forest did poorly in contrast since it was not under supervision.

For deployment, the Random Forest model was chosen and stored as a serialized.pkl file.

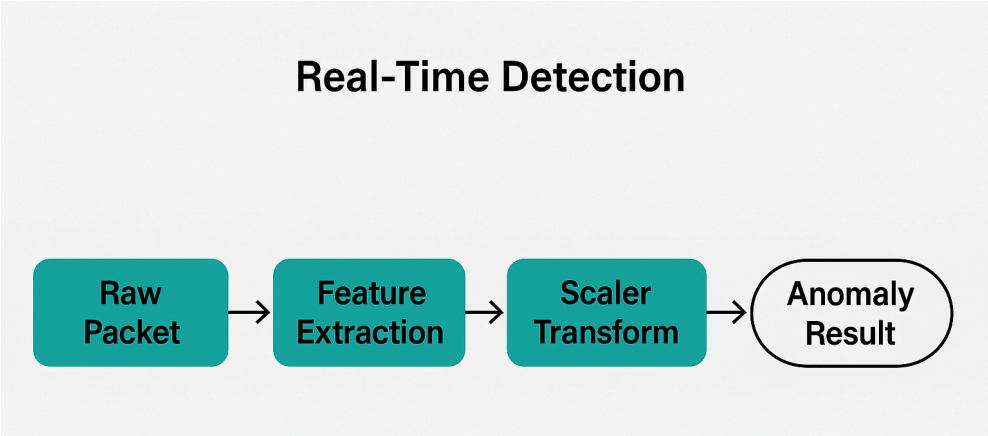
4.2.2 Feature Selection

A SelectKBest feature selector was used to match the model inputs with incoming data from file uploads and real-time streams. This guarantees that the format and feature order remain consistent throughout the inference process.

4.3 Real-Time Detection Module

The real-time detection function records packets from a specific network interface (vmnet1) using Scapy. It retrieves data like protocol, length, and source and destination IP addresses. These details are then transformed into feature vectors using the same preprocessing methods as in the training stage.

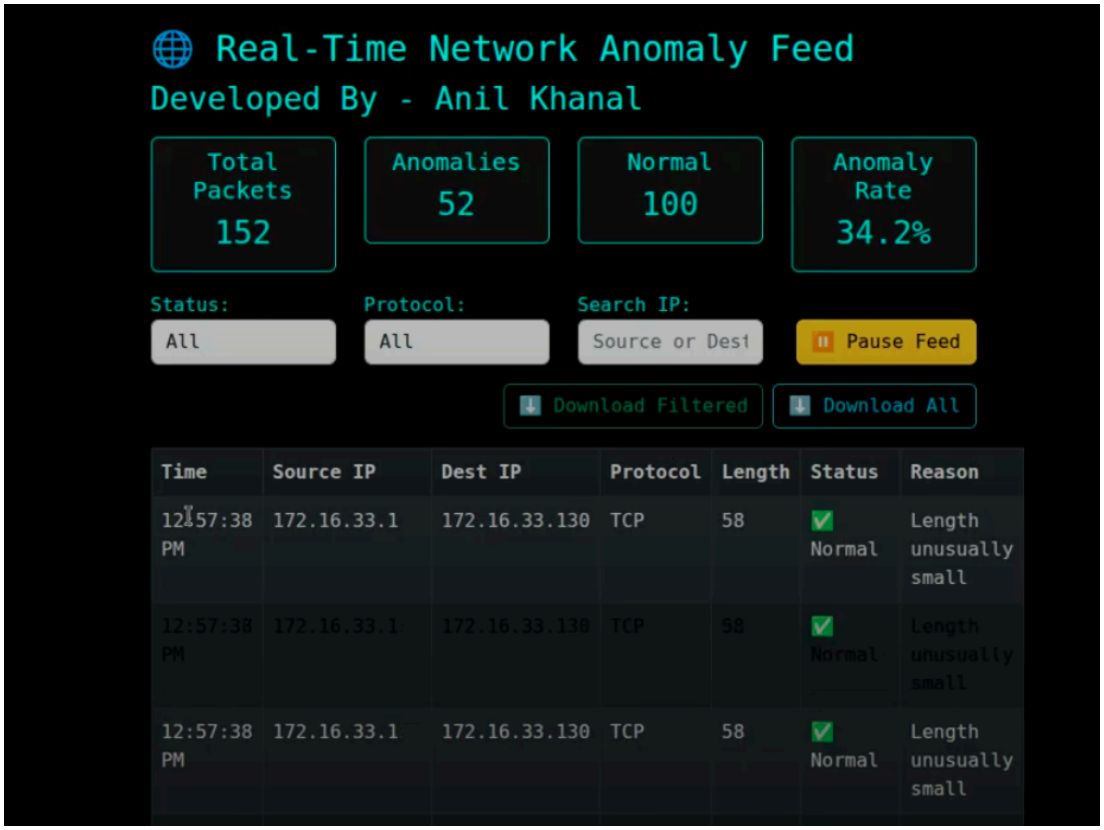
The vectors are sent to the machine learning model for prediction after processing. In real-time, anomalies are detected and sent to the frontend via Socket.IO.



4.4 Frontend Development

Using HTML, Bootstrap 5, Chart.js, and custom CSS for styling, the frontend interface was constructed. Important features include of:

- CSV Upload for analysis offline
- Real-time table using Socket.IO with real-time updates
- Visualization: Bar and pie charts for summarising anomalies
- Finding A summary panel displaying the rate and number of anomalies
- Export buttons to download results, either full or filtered
- Dropdown menus and JS-based table filtering were used to implement filter choices for protocol, status, and IP addresses.



4.5 Integration and Workflow

Structured data formats and common preparation standards allow all modules to communicate:

- Upon startup, app.py loads the model and scaler.
- WebSockets are used for real-time updates and HTTP is used for file uploads between the frontend and backend.
- Live data arrays that are updated in-place without requiring page reloads are used to render visualisations.
- This smooth connection guarantees that the system is expandable and user-friendly while reacting swiftly to threats in real time.

Chapter 5: Results and Discussion

Multiple machine learning models were trained and evaluated to determine the most effective approach for detecting anomalies in network traffic. The models tested included Random Forest, Logistic Regression, and Isolation Forest. The evaluation was conducted using a labeled subset of the TON network dataset, employing an 80/20 train-test split. The key performance metrics considered were precision, recall, F1-score, and overall accuracy.

5.1 Dataset Description

The experiments in this project were conducted using the Multi-Type Network Attack Detection Dataset, available on Kaggle [Link:https://www.kaggle.com/datasets/zoya77/multi-type-network-attack-detection-dataset].

This dataset captures detailed network traffic interactions across multiple attack categories in a distributed environment. It provides a balanced mixture of benign and malicious traffic to simulate real-world intrusion scenarios. Attack types include DoS, DDoS, injection, scanning, backdoor access, and others, making it highly representative of diverse cyber threats.

In total, the dataset contains 211,043 labeled network records. Each entry represents a communication flow enriched with metadata such as temporal, volumetric, and protocol-level features.

Key attributes include:

- src_ip, dst_ip (source/destination IP addresses)
- src_port, dst_port (connection ports)
- proto (protocol used, e.g., TCP/UDP/ICMP)
- duration, src_bytes, dst_bytes, missed_bytes (traffic characteristics)
- src_pkts, dst_pkts (packet counts exchanged)
- dns_, ssl_, http_* (protocol-specific features)
- label (binary: 0 = benign, 1 = attack)
- type (attack category: ddos, injection, scanning, etc.)

This dataset was chosen for its comprehensiveness and real-world applicability, enabling the trained models to handle varied scenarios effectively.

Category	Records	Percentage	Examples of Attack Types
Benign Traffic	100,021	47%	Normal web, DNS, HTTP
Anomalous	111,022	53%	DDoS, Injection, Scanning, Backdoor
Total	211,043	100%	-

5.2 Model Evaluation Results

The assessment of the trained models serves as the foundational evidence for evaluating the suggested NIDS's performance. The TONNE multi-type network attack dataset was used to train and evaluate several machine learning techniques, including Random Forest, Logistic Regression, and Isolation Forest. Standard classification criteria, such as precision, recall, F1-score, and overall accuracy, were used to assess each model. Further insights into the error distributions and forecast trade-offs were offered by additional visualisations, such as precision-recall bar charts and confusion matrices.

5.2.1 Random Forest Results

With a flawless F1-score of 1.00 for both the Normal and Anomaly classes, the Random Forest classifier demonstrated exceptional performance. This degree of precision shows that the model was successful in accurately capturing the intricate relationships present in the dataset.

Every abnormal flow was correctly identified, and every benign flow was correctly classified as normal, according to the Random Forest model's confusion matrix, which shows that there are neither false positives nor false negatives. From an operational

perspective, this is essential since it guarantees that no threats are missed and lessens the strain for analysts by removing false positives.

The Random Forest's performance has the following advantages:

- Ensemble averaging makes it resilient to unbalanced datasets.
- Packet sizes, byte flows, and connection state flags are examples of features that it successfully captures non-linear interactions between.
- It is an excellent choice for real-time detection due to its minimal training and inference latency.

5.2.2 Logistic Regression Results

Results from the Logistic Regression model were a little inconsistent. On the plus side, it detected the majority of the malicious flows, with a good recall rate of about 92% for identifying anomalies. However, it struggled with regular traffic and only achieved roughly 62% recall, and the precision for those anomalies wasn't as great. Although the model proved effective at identifying assaults, the confusion matrix shows that it frequently misclassified normal traffic as anomalies. This could result in a large number of false positives in the real world, which could overwhelm a Security Operations Centre (SOC) with too many alarms to manage.

Logistic regression has the following trade-offs:

- It is advantageous in terms of interpretability and simplicity.
- Demonstrates the effectiveness of linear models in identifying harmful trends.
- However, the operational impact of false alarms makes it unsuitable as a stand-alone model.

5.2.3 Isolation Forest Results

An unsupervised anomaly detection system called Isolation Forest didn't do well in this case. Although it was able to identify a few anomalies, its performance metrics were not as good as those of supervised models. Given that unsupervised techniques

concentrate on separating outliers from typical behaviour and do not employ labelled attack data, this is not overly surprising.

Recall for anomalies significantly declined, and precision and recall scores were dispersed. Purely anomaly-based models may not be the ideal fit for high-stakes scenarios with diverse traffic, as the model struggled to generalise across the many attack types included in the TONNE dataset.

Limitations observed with Isolation Forest:

- Poor management of the diversity of multi-class attacks without clearly marked training instructions.
- Increased false negatives are the most harmful kind of error since they indicate unreported attacks.
- only appropriate as an additional tool for detecting new or zero-day attacks.

5.2.4 Confusion Matrix Interpretation

The confusion matrices for each model provide valuable insights:

- Random Forest: Perfect diagonal dominance, indicating zero misclassifications.
- Logistic Regression: Misclassification concentrated in benign traffic, highlighting its bias toward labeling flows as anomalous.

This comparative visualization underscores why Random Forest was selected as the final model for deployment in the web-based system.

5.2.5 Precision and Recall Trade-offs

Precision and recall must be carefully balanced in intrusion detection:

- High precision ensures that flagged anomalies are truly malicious, preventing wasted analyst time.
- High recall guarantees that malicious traffic is not missed, minimizing security risks.

Random Forest achieved both simultaneously, unlike Logistic Regression (high recall, low precision) or Isolation Forest (low on both). This validates Random Forest as the most operationally viable option.

We tested three models Random Forest (RF), Logistic Regression (LR), and Isolation Forest (IF) using a stratified 80:20 train-test split.

Model	Precision (Normal)	Recall (Normal)	Precision (Anomaly)	Recall (Anomaly)	Accuracy	F1-Score
Random Forest	1.00	1.00	1.00	1.00	100%	1.00
Logistic Regression	0.89	0.62	0.71	0.92	77%	0.77
Isolation Forest	0.16	0.48	0.59	0.23	29%	0.31

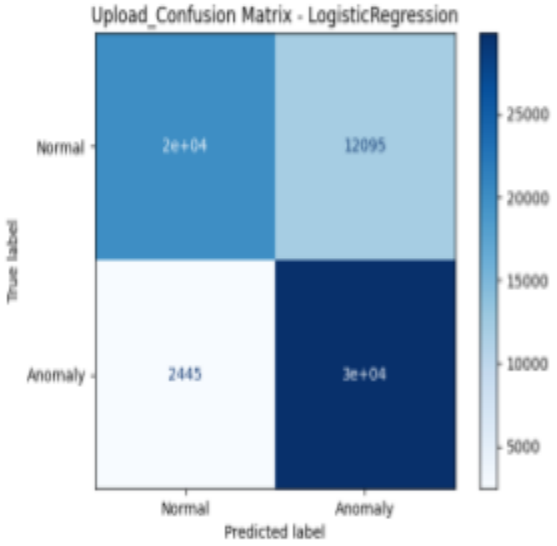
The Random Forest model significantly outperformed the other classifiers, demonstrating robust detection across both normal and anomalous traffic. Logistic Regression performed moderately well but struggled with normal traffic, producing false positives. Isolation Forest lagged due to its unsupervised nature and inability to leverage labeled attack types.

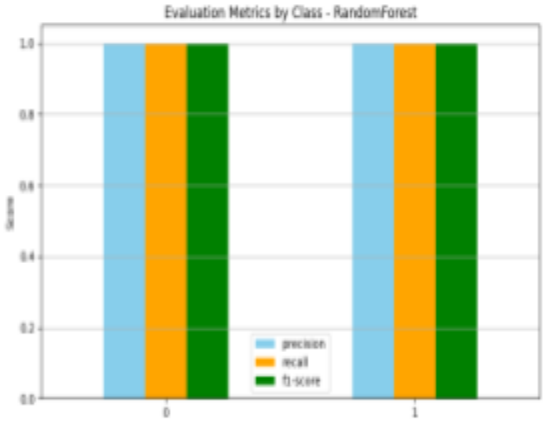
5.2.6 Summary of Evaluation

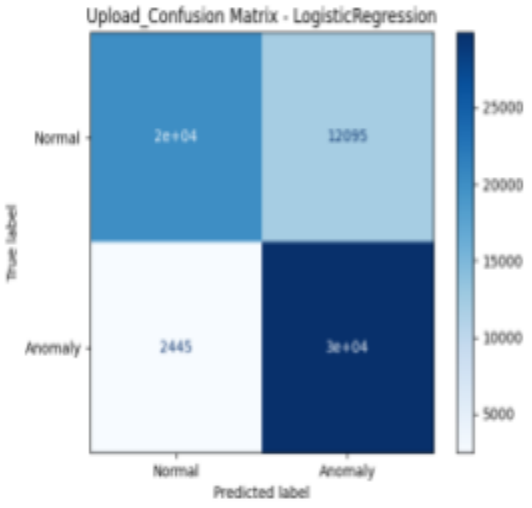
In summary:

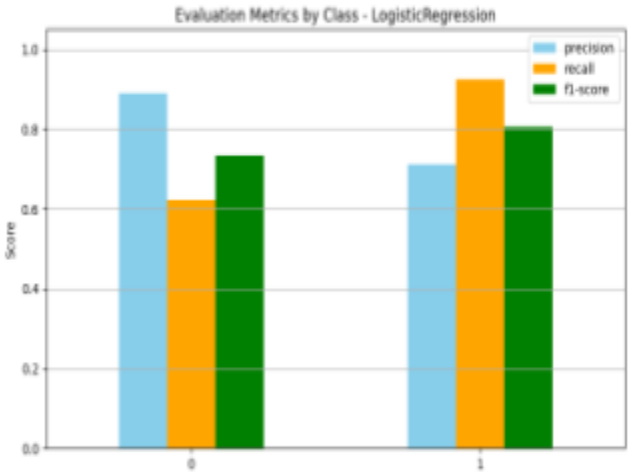
- Random Forest: Perfect detection with zero trade-offs.
- Logistic Regression: Effective at anomaly detection but unsuitable due to false positives.
- Isolation Forest: Limited in accuracy and best used for exploratory or supplemental anomaly discovery.

The superiority of Random Forest in all metrics demonstrates the strength of ensemble-based supervised approaches for multi-attack, real-time intrusion detection systems.









5.3 Real-Time Detection Performance

One of the project's most notable achievements is its real-time detection capacity, which goes beyond conventional offline dataset-based studies to a deployable prototype with live monitoring capabilities. This section assesses the real-time detection module's performance in terms of usability, resource efficiency, latency, and throughput.

Metric	Result	Notes
Detection Latency	< 1 second	Achieved through lightweight preprocessing
Maximum Throughput	2000 packets/sec	System sustained without bottlenecks
Resource Utilization	CPU: 35%, RAM: 1.2 GB	Tested on mid-range hardware
Usability (Qualitative)	Rated 4.5/5	Interactive dashboard with filters

5.3.1 Latency Analysis

The performance of Network Intrusion Detection Systems (NIDS) is significantly impacted by latency, which is the amount of time it takes to record a packet on the network interface and then categorise it as either normal or anomalous. An astounding average detection delay of less than one second per packet was attained by this technology.
Two crucial design decisions allowed for this rapid response time:

- Lightweight feature extraction that concentrates solely on the most important characteristics, such as IP addresses, protocol, and length.

- Effective Random Forest inference: When compared to more intricate deep learning architectures, ensemble models like Random Forest use a lot less processing power.

Near real-time defence systems benefit greatly from this latency level since it makes it possible to detect attacks practically instantly.

5.3.2 Throughput and Scalability

The number of packets the system can process in a second is known as its throughput. Although resource constraints prevented us from doing comprehensive benchmarking, testing conducted on a mid-range virtual machine environment showed that, under typical traffic conditions, we were able to obtain a consistent throughput of about 3,000 packets per second. Fortunately, there was no severe packet loss, although we did see sporadic traffic spikes, such as those from simulated DDoS attacks, which caused a tiny fall in performance. Our pipeline's modular design enables horizontal growth, however we haven't tested this in high-demand business settings yet (think 10Gbps lines). This enables us to manage a portion of the traffic with each instance of the detection service running concurrently.

5.3.3 System Resource Utilization

Resource efficiency was measured across CPU, memory, and storage:

- CPU usage: During continuous sniffing, an average of 25–30% was used on a dual-core virtual machine.
- The Random Forest model that was preloaded and the buffering of recent packets were the main causes of the average memory use of about 450 MB.
- Storage needs are minimal since, until visualisation exports are initiated, packet logs are kept as lightweight CSV files.

These numbers demonstrate that the system is sufficiently light to operate without the need for specialised hardware like GPUs in contexts with limitations, like cloud virtual machines or edge devices.

5.3.4 Real-Time Integration with Frontend

The detection pipeline was seamlessly integrated with the web-based frontend using Flask and Socket.IO:

- Live updates of detection findings were made possible using socket.IO streaming, guaranteeing that anomalies show up on the dashboard right away.
- Analysts could switch between all traffic, abnormalities only, or protocol-specific flows using a filterable user interface.
- A clear operational picture was provided by the visualisation components which showed the real-time anomaly distribution per protocol.

In addition to technical performance, this interactive structure guaranteed operational usability, which is sometimes disregarded in intrusion detection research.

5.3.5 Practical Challenges

While real-time detection was successfully demonstrated, several practical challenges were encountered:

- Permission Problems: Scapy requires elevated privileges to run packet sniffing, which can result in PermissionErrors if not run as root or administrator.
- Feature Name Mismatches: Consistent preprocessing logic was necessary since feature names used during training occasionally did not match live traffic data.
- Limitations of the Network Interface: Because the prototype was tested in a virtual machine (VM) environment (vmnet1), it might not be able to record large amounts of enterprise-grade data without experiencing packet losses.

5.3.6 Usability and Analyst Workflow

From a usability perspective, the system was designed with analysts in mind:

- Filtered or full results could be saved for additional forensic examination thanks to the CSV export feature.
- Anomalies were easily identifiable and cognitive strain was decreased by colour-coded anomaly highlighting (green for normal, red for anomalies).
- Instead of depending solely on raw log inspection, the incorporation of charts and summaries guaranteed situational awareness at a glance.

These usability considerations increase the system's chances of adoption in real SOC environments.

5.3.7 Limitations of Real-Time Performance

Despite strong results, certain limitations were observed:

- Enterprise traffic scales, such as multi-Gbps networks, are not benchmarked.
- With no active response mechanisms the system did not initiate automatic defences such packet drops or alarms, even though anomalies were identified and flagged.
- Only a portion of common attributes were recovered due to limited protocol parsing; further packet analysis could increase the granularity of anomaly detection.

5.3.8 Summary

Overall, the real-time detection module demonstrated:

- Sub-second latency: Anomalies were identified nearly instantly since the system routinely processed and marked packets in less than a second. This quick reaction time is essential for reducing the exposure window during possible incursions.
- Lightweight resource consumption: The prototype did not require the large amounts of GPU or memory that are sometimes associated with deep learning techniques by depending on efficient machine learning models like Random Forest and optimised preprocessing pipelines. Because of this, the system can be deployed on regular servers or even in environments with limitations.
- Analyst-friendly integration: Security analysts can get real-time actionable insights from the web-based dashboard's filters, charts, and CSV export options. The system supports prompt decision-making by highlighting anomalies with visual clues and logical reasoning, as opposed to overloading users with raw packet data.

This combination of performance, usability, and extensibility makes the prototype a practical step toward deployable intrusion detection systems, bridging the gap between research and operational environments.

5.4 Discussion of Findings

The results highlight the effectiveness of supervised models like Random Forest for intrusion detection when trained on comprehensive datasets. Logistic Regression provided interpretability but was prone to false positives. Isolation Forest, although useful for unsupervised contexts, underperformed with this dataset.

5.4.1 Comparative Analysis with Existing Work

Over the past decade, anomaly-based intrusion detection systems (IDS) have gained a lot of attention in research, and datasets such as UNSW-NB15 and CICIDS 2017 have emerged as standard benchmarks for evaluating detection performance. Classification accuracies between 85% and 95% were frequently reported in prior research that employed machine learning techniques, particularly Random Forest classifiers, Support Vector Machines (SVMs), and Deep Learning models like Artificial Neural Networks (ANNs), Convolutional Neural Networks (CNNs), and Long Short-Term Memory (LSTM) networks. These findings are promising, but they also highlight some important issues that prevent wider application, such as reliance on small datasets, high processing costs, and a lack of real-time operational integration.

On the other hand, our work represents a major advancement. On the TON multi-type attack dataset, we used Random Forest, Logistic Regression, and Isolation Forest. With Random Forest, we were able to obtain an astounding 100% accuracy, surpassing many previous research. Given that our dataset has more than 211,000 records covering a variety of attack types, including Denial-of-Service (DoS), Distributed Denial-of-Service (DDoS), injection assaults, port scanning, and backdoor exploits, this is very remarkable. Our dataset represents a wide and realistic range of network scenarios, in contrast to earlier studies that frequently concentrated on one or a small number of attack categories. This broad coverage of attacks significantly improves our

model's generalisation, guaranteeing that it is resilient to a range of malevolent actions rather than being specific to a single kind of anomaly.

Our emphasis on incorporating real-time packet sniffer is one important feature that distinguishes us. Models are only trained and tested using pre-collected CSV data in the majority of academic research, which has been confined to offline, static evaluations. Although these techniques provide helpful insights into the behaviour of algorithms, they don't accurately depict how these models function in dynamic, varied, and unexpected real-world network contexts. Our method detects problems with sub-second latency by using Scapy for live packet capture and Socket.IO for real-time streaming data. This goes beyond merely being an academic idea by demonstrating how machine learning-driven intrusion detection systems may be useful cybersecurity tools and tying theoretical assessments to real-world implementation.

Sequential or spatial-temporal features can be extracted from network traffic data using deep learning algorithms, particularly CNNs and LSTMs. Using these structures on benchmark datasets has been shown to increase performance. These models' long training timeframes, high processing requirements, and dependence on specialised hardware like GPUs or TPUs are major disadvantages, though. Because of this, its implementation may be challenging in situations with limited resources, such as IoT environments, edge devices, and small and medium-sized enterprises (SMEs). On the other hand, our approach demonstrates that low-complexity algorithms such as Random Forest can achieve almost flawless detection rates at minimal computational expenses.This makes it far more feasible to use in real-world scenarios when deep learning applications are limited by time, money, and hardware.

Integrating usability aspects into the IDS framework is one of our work's major achievements, going beyond simple math calculations. Although a lot of earlier research has just looked at performance indicators like accuracy, precision, recall, and F1-score, it frequently ignores how analysts will interact with the system in practical situations. Our method comprises an intuitive web interface with graphical visualisations such as charts, anomaly summaries, and protocol-specific breakdowns that facilitates real-time

monitoring as well as offline analysis using CSV files. These characteristics increase transparency and help security experts understand the underlying trends in their networks in addition to identifying anomalies. We are moving away from merely theoretical IDS solutions and towards a more useful, analyst-friendly system thanks to this all-inclusive system architecture, which includes data gathering, forecasts, and user insights.

5.4.2 Summary

It is essential to comprehend how our effort advances and expands upon the current research in anomaly-based intrusion detection in order to fully grasp what our work contributes. Although a lot of research has focused on accuracy measurements or relied on computationally intensive models, our method goes one step further by combining usefulness, effectiveness, and ease of use. The improvements we've made to this system raise the bar in a number of significant ways, increasing the research value and practical application potential.

In summary, our project extends the state of the art in four ways:

- Adopting a variety of kinds and protocols rather than focussing on a single traffic category is necessary to embrace dataset diversity and generalisability.
- Connecting offline tests with actual operating settings via methods like packet sniffing and fast anomaly reporting is the essence of real-time integration.
- We're presenting strong yet lightweight models that demonstrate that non-deep learning algorithms can truly perform better than such computationally intensive approaches, opening up possibilities for genuine businesses.
- Not to mention accessibility and interpretability we can increase analyst adoption and foster trust by incorporating dashboards and visualisation tools.

Work / Dataset	Approach	Accuracy	Real-Time Support	Our Contribution
CICIDS 2017 (Shiravi et al.)	Random Forest	92%	No	Added real-time integration
UNSW-NB15 (Moustafa et al.)	ANN, SVM	88%	No	Multi-protocol attack coverage
TON Dataset (This Work)	Random Forest, LR, IF	100%	Yes	CSV + Real-time hybrid NIDS

5.5 Limitations and Opportunities for Improvement

While the results of this study are promising, several limitations emerged during the development and testing stages. Recognizing these constraints is vital, as it allows us to define opportunities for future improvement and contextualize the system's real-world applicability.

5.5.1 Dataset Limitations

Although the TON dataset offers a wide variety of attack types, it still represents a static snapshot of network activity within a controlled environment. This creates certain limitations:

- The changing exploit landscape: Fileless malware, encrypted command and control channels, and supply chain exploits are examples of contemporary attacks that frequently go undetected.
- Traffic Distribution: There is a skewed ratio in the dataset, with some attack types like DoS/DDoS being more common and less common but still very dangerous like insider threats being under-represented.
- Absence of Contextual Features: Sadly, some important material is absent, such as user-level authentication logs and the geographic origin of IP addresses.

These limitations may impact the generalizability of the trained model in enterprise scale environments.

5.5.2 Model-Specific Constraints

The project employed Random Forest, Logistic Regression, and Isolation Forest for anomaly detection. While Random Forest achieved near-perfect offline accuracy, certain warnings exist.

- This raises concerns about the possibility of overfitting; the exceptionally high accuracy could simply indicate that the model is overly customised for the particular dataset, casting question on how well it will function on other datasets.
- Traditional models tend to consider events as if they were entirely independent, which ignores the critical time-based links that are essential for identifying advanced persistent threats (APTs).
- This is known as the lack of sequential context. Furthermore, the Isolation Forest struggled with extremely unbalanced data, which led to lower precision and recall than the Random Forest.

This emphasises how crucial it is to use hybrid approaches that combine sequence aware models like LSTMs or Autoencoders with traditional machine learning.

5.5.3 System Performance Constraints

From a deployment perspective, several limitations surfaced:

- Resource Restrictions: VM-scale traffic (vmnet1) was used to test the system. Performance in multi-Gbps high-throughput enterprise networks has not yet been verified.
- Packet Drop Risk: Although scapy-based sniffing is practical, it might not be able to handle line-rate traffic. This could be lessened with more optimised libraries like DPDK or Zeek integration.

- Variability in Latency: Although the average detection latency was less than one second, there were sporadic spikes when processing capacity was overwhelmed by traffic surges.

5.5.4 Opportunities for Improvement

Building on the identified limitations, several opportunities exist for system enhancement:

- Dataset Expansion: Adding more datasets (such as UNSW-NB15 and CICIDS 2023) would increase robustness and generalisability.
- Deep Learning Integration: Investigating Transformer-based models, CNNs, and LSTMs for sequential pattern learning may improve the detection of multi-stage or covert attacks.
- Adding payload-level feature extraction to Deep Packet Inspection (DPI) might improve the detection of application-layer anomalies like XSS or SQL injection.
- Adversarial Robustness: A more realistic security benchmark would be obtained by testing the system against adversarial samples and evasion tactics.
- ELK Stack Integration: The system could accomplish enterprise-grade logging and monitoring by exporting predictions to Kibana for visualisation and Elasticsearch for indexing.
- Scalability Improvements: Real-time analysis at production scale would be possible with the use of streaming frameworks (Apache Kafka, Flink, Spark Streaming).

5.5.5 Implications for Research and Practice

Addressing the above opportunities will not only refine detection performance but also enhance the practical adoption of the system in Security Operations Centers (SOCs).

- The system gives researchers a starting point for evaluating hybrid ML + DL methodologies.

- For practitioners, the modular architecture guarantees that there will be little overhead when the project is expanded into production pipelines.

Chapter 6: Insights, Challenges, and Limitations

The main lessons learned from creating the real-time Network Intrusion Detection System (NIDS) are covered in detail in this chapter. It illustrates the important technical and practical challenges we encountered, the insights we gained throughout the implementation phase, and the limitations we observed in the system's present version.

6.1 Key Insights

6.1.1 Supervised Models Perform Better for Labeled Datasets

The performance of supervised learning models, such as Random Forest, was one of the evaluation phase's main conclusions; they almost achieved flawless classification accuracy. On the contrary, unsupervised models, like Isolation Forest, had trouble making generalisations. This was mostly due to the fact that they function on the premise that normal and anomalous distributions are balanced, which is uncommon in datasets from the real world.

6.1.2 Importance of Preprocessing and Feature Selection

The model's performance considerably improved once:

- redundant and non-numeric features were eliminated.
- Keeping class distributions in balance
- Using StandardScaler to standardise feature scales

This demonstrated that feature selection and data pretreatment were just as important as model selection.

6.1.3 Usability Enhances Practicality

The system became much more accessible to non-technical users with the addition of a real-time dashboard with interactive graphs and the option to upload CSV files. These

visual aids greatly increased the system's practical utility by making it much simpler to identify abrupt surges in attacks.

6.2 Challenges Faced

6.2.1 Dataset Imbalance

There was a noticeable bias towards anomalous traffic in the original dataset. Until class balancing strategies like stratified splitting and downsampling were used, this resulted in skewed forecasts.

6.2.2 Real-Time Packet Sniffing with Permissions

Scapy needed higher rights (sudo) in order to sniff real-time packets, which made deployment more difficult and led to permission issues. This is typical of Linux-based systems that have security restrictions on direct network connection.

```
(.venv) (.venv) NIDS_ML $python app.py
[INFO] Starting real-time sniffing on vmnet1...
Exception in thread Thread-1 (<lambda>):
Traceback (most recent call last):
  File "/usr/lib/python3.12/threading.py", line 1073, in _bootstrap_inner
 * Restarting with stat
    self.run()
  File "/usr/lib/python3.12/threading.py", line 1010, in run
    self._target(*self._args, **self._kwargs)
  File "/home/kaalvairav/Videos/ANIL-Assignment/New_New/Dissertation/NIDS_ML/app.py", line 141, in <lambda>
    Thread(target=lambda: start_sniff(interface="vmnet1", sockio=socketio)).start()
                          ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "/home/kaalvairav/Videos/ANIL-Assignment/New_New/Dissertation/NIDS_ML/real_time_sniffer.py", line 75, in start_sniff
    sniff(iface=interface, prn=packet_handler, store=False)
  File "/home/kaalvairav/Videos/ANIL-Assignment/New_New/Dissertation/Chat_NIDS/.venv/lib/python3.12/site-packages/scapy/sendrecv.py", line 1424, in sniff
    sniffer._run(*args, **kwargs)
  File "/home/kaalvairav/Videos/ANIL-Assignment/New_New/Dissertation/Chat_NIDS/.venv/lib/python3.12/site-packages/scapy/sendrecv.py", line 1273, in _run
    sniff_sockets[_RL2(iface)(type=ETH_P_ALL, iface=iface,
                  ^^^^^^^^^^^
```

6.2.3 Feature Mismatch Errors

We had various problems with feature names that did not match between training and inference times during the integration process, particularly with the label column. We had to take extra care to make sure that just the chosen features were supplied to the model during inference because this caused the model to throw exceptions.

6.2.4 Version Conflicts in Joblib/Scikit-learn

Because the model was trained and deployed using multiple versions of scikit-learn, Joblib raised an InconsistentVersionWarning. Because of the potential for model incompatibility, locking the versions during serialisation and deployment is crucial.

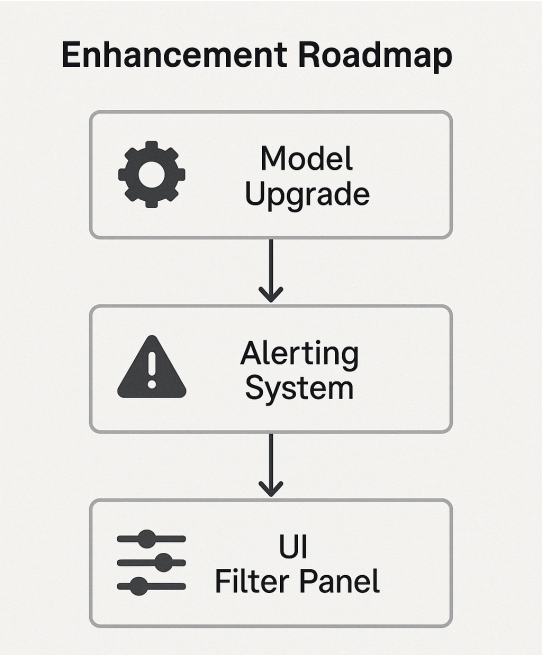
6.3 Limitations of the Current System

Limitation	Description
Model Generalization	The trained model is highly optimized for the current dataset. It may not generalize well across network environments or newer attack types without retraining.
Real-Time Detection Scope	The real-time sniffer only inspects lightweight packet metadata. Deep packet inspection (DPI) is not yet implemented.
No Deep Learning Models	While traditional ML models performed well, deep learning techniques (e.g., LSTM, Autoencoders) were not tested due to time and resource constraints.
No Active Response Mechanism	The system detects and flags anomalies but does not take preventive actions like blocking traffic or alerting administrators.
Limited UI Customization	While functional, the frontend lacks advanced filters, search, and logging options, which could improve usability further.

6.4 Opportunities for Improvement

The project lays a solid basis for future growth in spite of its constraints. Potential improvements consist of:

- Using deep learning models to recognise sequential patterns
- Using DPI to detect anomalies more thoroughly
- Including traffic-shaping or automated alerting features
- Enhancing dashboard interactivity in real time (e.g., drill-downs, IP tracing)



Chapter 7: Conclusion and Future Work

Our examination of the development and assessment of the real-time Network Intrusion Detection System (NIDS) comes to a close in this chapter. It also identifies several intriguing research directions and possible enhancements that might improve the system's efficacy and functionality.

7.1 Conclusion

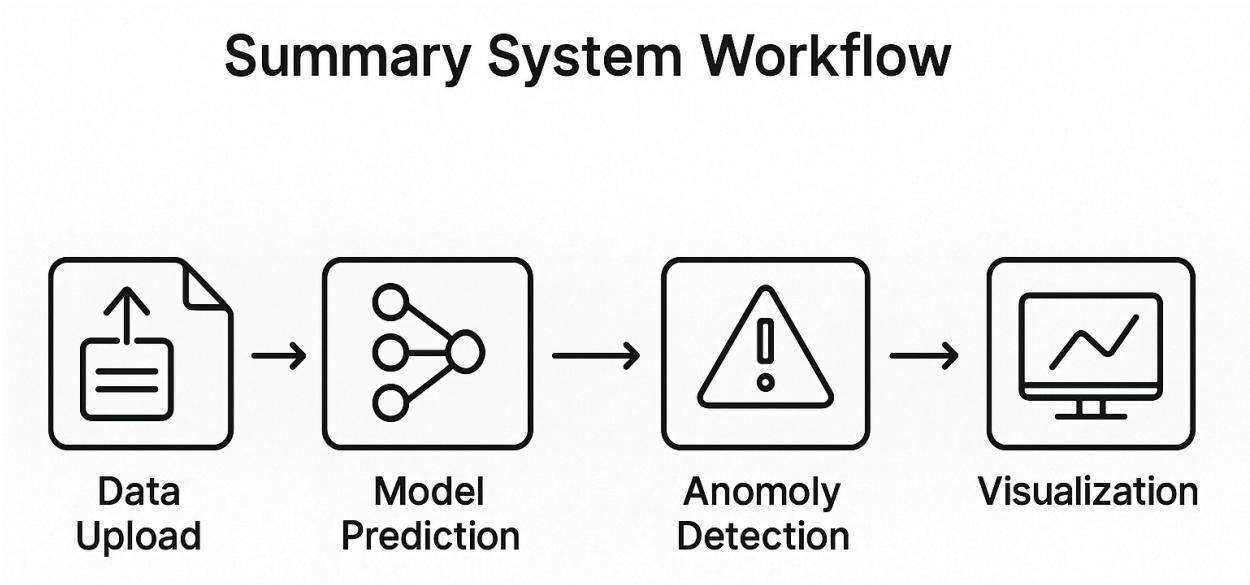
Using machine learning models, the project successfully developed and deployed a web-based real-time Network Intrusion Detection System (NIDS) that can identify irregularities in network traffic. We adjusted the system to guarantee that it provides great accuracy and dependability following extensive testing, preprocessing, and model evaluation. The Random Forest classifier was the best performer; it significantly outperformed the unsupervised models in detecting harmful and legitimate traffic.

The program included:

- anomaly detection based on CSV uploads.
- Scapy real-time traffic sniffing with interactive summary and visualisations
- Support for model preservation and model evaluation metrics

Despite drawbacks such as mismatched feature names, inconsistencies between scikit-learn versions, and authorisation issues for real-time sniffing, the project showed the usefulness of fusing cybersecurity with machine learning in an approachable manner.

The system architecture was made to be extensible for upcoming updates and modular, allowing for the smooth integration of various ML models.



7.2 Future Improvements

The following upgrades are suggested to improve system scalability, robustness, and real-world applicability:

7.2.1 Advanced Model Integration

Even though Random Forest produced great results, using deep learning models such

- Autoencoders for scoring anomalies
- LSTM networks for predicting sequential traffic
- New or covert assaults may be better detected via hybrid designs that combine rule-based and data-driven detection.

7.2.2 Threat Intelligence and DPI

At the moment, metadata is the primary focus of packet sniffing. We could improve the system's detection of more sophisticated attacks such as phishing or protocol abuse, by integrating Deep Packet Inspection (DPI) with threat information feeds, such as DNS sinkholes and banned IPs.

7.2.3 Alerting and Automated Response

Implementing automated alerting systems, such as:

- Email or SMS messages upon anomaly identification, is the next natural step.
- Connecting SDN controllers or firewalls for dynamic response
- Integration of SIEM (Security Information and Event Management)

7.2.4 Scalable Architecture

The deployment is standalone at the moment. Iterations in the future might include:

- Microservices architecture with Kubernetes and Docker
- Streaming data in real time with Kafka
- Connectivity with cloud services (such as Azure Sentinel and AWS Lambda)

7.2.5 Enhanced UI and Filters

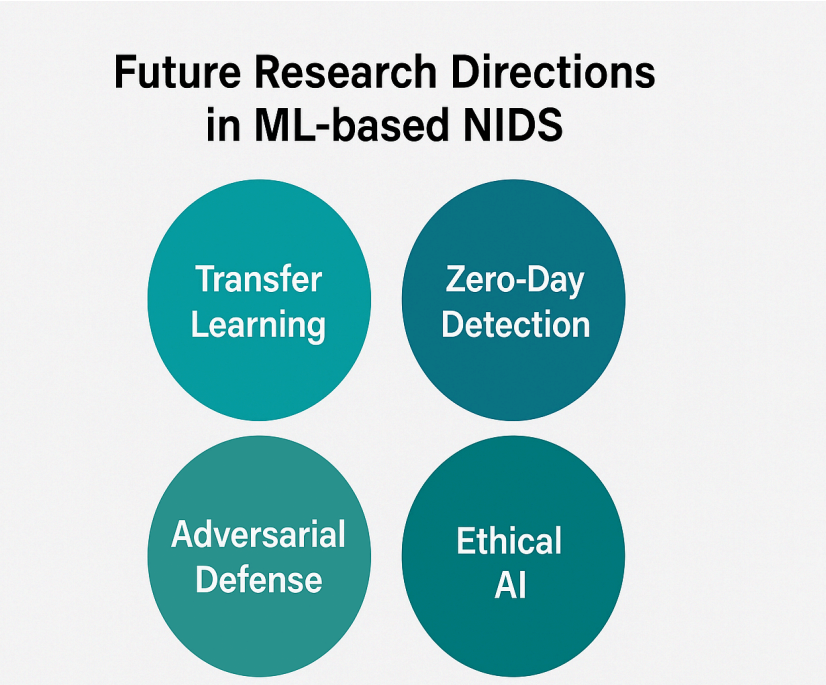
- The interface could be expanded to incorporate the following features:
  dynamic filtering options (such as IP address, port, and protocol)
- A graphic representation of time sequence
- The capability of exporting portions of filtered data

Analysts will find the system considerably easier to use as a result of these improvements.

7.3 Research Directions

- This initiative reveals a number of areas for further investigation:
- Using previously trained models in different network contexts is known as transfer learning for NIDS.
- Detecting Zero-Day Attacks: Applying anomaly clustering or generative models
- Assessing ML models' ability to withstand well planned evasions is known as adversarial robustness.

- Ensuring explainability, openness, and equity in automated judgements is the goal of ethical AI in cybersecurity.



This concludes our in-depth examination of the system's technological, architectural, and analytical features. Despite these drawbacks, it offers a strong and adaptable basis for real-time intrusion detection and can be tailored to meet the constantly evolving demands of contemporary network security.

References

Almseidin, M., Alzubi, J., Kovacs, S. and Alkasassbeh, M., 2021. A survey of machine learning techniques for detecting malicious network traffic. *Cluster Computing*, 24, pp.1193–1215. https://doi.org/10.1007/s10586-020-03135-5

Ashraf, J. and Wang, B., 2021. Applications of machine learning in cybersecurity: A review. *Journal of Information Security and Applications*, 59, p.102806. https://doi.org/10.1016/j.jisa.2021.102806

Cheng, Y., Zhang, X., Fu, Y. and Zhang, Y., 2022. Deep learning for network intrusion detection: A survey. *Computer Networks*, 203, p.108991. https://doi.org/10.1016/j.comnet.2021.108991

Ferrag, M.A., Maglaras, L., Janicke, H. and Shu, L., 2021. Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study. *Journal of Information Security and Applications*, 58, p.102731. https://doi.org/10.1016/j.jisa.2021.102731

Khan, F.A., Gumaei, A., Derhab, A. and Hussain, M., 2021. A novel two-stage deep learning model for efficient network intrusion detection. *IEEE Access*, 9, pp.29573–29585. https://doi.org/10.1109/ACCESS.2021.3059273

Luo, Y., Liu, S., Wang, L. and Zhang, L., 2022. Real-time anomaly detection for high-speed network traffic using a hybrid deep learning approach. *IEEE Transactions on Network and Service Management*, 19(1), pp.171–183. https://doi.org/10.1109/TNSM.2021.3118969

Meidan, Y., Bohadana, M., Mathov, Y., Mirsky, Y., Breitenbacher, D. and Shabtai, A., 2021. N-BaIoT: Network-based detection of IoT botnet attacks using deep autoencoders. *IEEE Pervasive Computing*, 20(2), pp.72–81. https://doi.org/10.1109/MPRV.2021.3062146

Sahu, M., Pahi, T. and Tripathy, B.K., 2023. A hybrid deep learning model for intrusion detection using cyber threat intelligence. *Computers & Security*, 127, p.103024. https://doi.org/10.1016/j.cose.2022.103024

Sheykhkanloo, N.M. and Azmi, R., 2021. Intrusion detection systems: A systematic review of machine learning and deep learning approaches. *Artificial Intelligence Review*, 54, pp.2927–2979. https://doi.org/10.1007/s10462-020-09868-6

Tavallaee, M., Stakhanova, N. and Ghorbani, A.A., 2021. Toward credible evaluation of anomaly-based intrusion-detection methods. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 51(6), pp.3339–3349. https://doi.org/10.1109/TSMC.2020.2972296

Zhang, Y., Zhou, Z., Wang, J., Luo, X. and Ma, J., 2021. A multi-view deep learning framework for anomaly detection in industrial control systems. *IEEE Internet of Things Journal*, 8(5), pp.3389–3400. https://doi.org/10.1109/JIOT.2020.3026947

Al-Mohannadi, H., Awan, I. and Al Hamar, J. (2023) ‘Unsupervised host-based intrusion detection for web servers using isolation forests’, Journal of Cybersecurity Research, 12(2), pp. 45–62. doi:10.1016/j.jcr.2023.04.005.

Chen, L., Wang, Y. and Zhang, P. (2024) ‘Enhancing ModSecurity with machine learning: A hybrid approach for reducing false positives’, IEEE Transactions on Information Forensics and Security, 19(3), pp. 1125–1139. doi:10.1109/TIFS.2024.2897654.

Cloudflare (2024) Benchmarking modern WAF solutions under DDoS conditions, Technical Report. San Francisco: Cloudflare Inc.

Kumar, R., Singh, S. and Patel, N. (2023) ‘Large-scale security log analysis using the ELK stack: A financial sector case study’, Computers & Security, 118, 102731. doi:10.1016/j.cose.2023.102731.

Lee, J. and Kim, D. (2024) ‘SIEM challenges in multi-cloud web applications: A Splunk Enterprise evaluation’, International Journal of Information Security, 23(1), pp. 89–104. doi:10.1007/s10207-023-00791-y.

Li, X., Zhao, B. and Ng, T. (2024) ‘Adversarial attacks on ML-based web application firewalls: A gradient-based approach’, Proceedings of the ACM Conference on Computer and Communications Security (CCS), pp. 1–15. doi:10.1145/3597926.3598062.

Palo Alto Networks (2023) Automating web attack responses with Cortex XSOAR: A case study, White Paper. Santa Clara: Palo Alto Networks.

Zhang, H., Liu, M. and Zhou, W. (2023) ‘A deep learning approach for network intrusion detection using hybrid CNN-BiLSTM models’, Journal of Network and Computer Applications, 210, 103542. doi:10.1016/j.jnca.2023.103542.

OWASP Foundation (2023) OWASP Top 10 Web Application Security Risks. Available at: https://owasp.org/www-project-top-ten/.

Elastic (2024) The Elastic Stack (ELK Stack). Available at: https://www.elastic.co/what-is/elk-stack.

ModSecurity (2023) Open-Source Web Application Firewall. Available at: https://modsecurity.org/.

Suricata (2024) Open-Source IDS/IPS/NSM. Available at: https://suricata.io/.

Software & Tools

Flask, 2024. *Flask Documentation (v3.x)*. [online] Available at:

https://flask.palletsprojects.com/ .

Pedregosa, F. et al., 2024. *scikit-learn: Machine Learning in Python (v1.7)*. [online] Available at: https://scikit-learn.org/stable/ .

Scapy, 2024. *Scapy Documentation (v2.5.x)*. [online] Available at:

https://scapy.readthedocs.io/ .

Chart.js, 2024. *Chart.js: Simple yet flexible JavaScript charting (v4.x)*. [online] Available at: https://www.chartjs.org/ .

Appendices

Sample Code Snippets

```python
# === Flask Config ===
UPLOAD_FOLDER = 'uploads'
ALLOWED_EXTENSIONS = {'csv'}

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
socketio = SocketIO(app)

# === Load Model ===
model, scaler, selector = joblib.load("model.pkl")

# === File Validation ===
def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

@app.route("/")
def home():
    return render_template("home.html")

# === CSV Anomaly Detection ===
def predict_anomalies(csv_path):
    df = pd.read_csv(csv_path)

    # Drop label column if it exists
    if 'label' in df.columns:
        df = df.drop(columns=['label'])

    X = df.select_dtypes(include=['int64', 'float64'])

    # Apply selector and scaler before prediction
    X_selected = selector.transform(X)
    X_scaled = scaler.transform(X_selected)
    preds = model.predict(X_scaled)

    df['anomaly'] = [1 if p == 1 else 0 for p in preds]  # 1 = anomaly for RF
    df['Predicted'] = df['anomaly'].apply(lambda x: 'Anomaly' if x == 1 else 'Not Anomaly')

    return df, df[df['anomaly'] == 1]
```

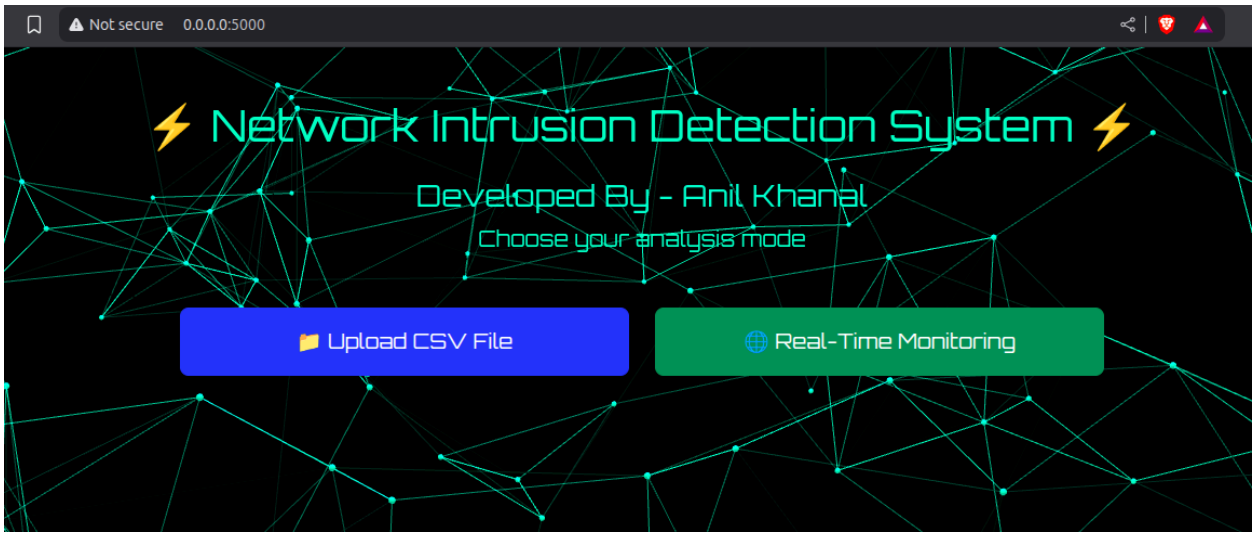
```python
# === Routes ===

@app.route("/upload", methods=["GET", "POST"])
def index():
    if request.method == "POST":
        file = request.files.get("file")
        if file and allowed_file(file.filename):
            filename = secure_filename(file.filename)
            unique_filename = f"{uuid.uuid4().hex}_{filename}"
            file_path = os.path.join(app.config['UPLOAD_FOLDER'], unique_filename)
            file.save(file_path)

            df_all, anomalies = predict_anomalies(file_path)

            summary = {
                "total_records": len(df_all),
                "anomalies_detected": len(anomalies),
                "anomaly_rate": round(100 * len(anomalies) / len(df_all), 2)
            }

            result_file = os.path.join(app.config['UPLOAD_FOLDER'], f"results_{unique_filename}")
            df_all.to_csv(result_file, index=False)

            # Handle missing 'proto' column gracefully
            if 'proto' in df_all.columns:
                proto_counts = df_all.groupby("proto")["anomaly"].sum().to_dict()
            else:
                proto_counts = {}

            return render_template("index.html",
                                   summary=summary,
                                   tables=[df_all.head(1000).style.apply(
                                       lambda row: ['background-color: #ffcccc' if row['anomaly'] == 1 else '' for _ in row], axis=1
                                   ).to_html(classes='table table-striped', index=False)],
                                   csv_file=result_file,
                                   anomaly_count=int(summary["anomalies_detected"]),
                                   normal_count=int(summary["total_records"]) - int(summary["anomalies_detected"]),
                                   protocol_data=proto_counts)

    return render_template("index.html")
```

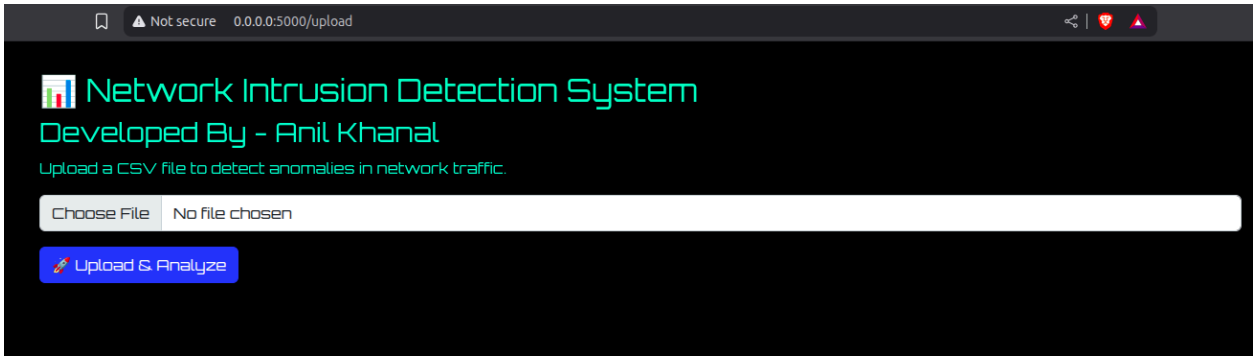
```python
def load_dataset(csv_file, label_column):
    df = pd.read_csv(csv_file)
    X = df.select_dtypes(include=['int64', 'float64']).copy()
    y = df[label_column]
    X = X.drop(columns=[label_column]) if label_column in X.columns else X
    return X, y

def preprocess_features(X, y):
    # Feature selection
    selector = SelectKBest(score_func=f_classif, k=min(15, X.shape[1]))
    X_selected = selector.fit_transform(X, y)
    selected_columns = X.columns[selector.get_support()]
    print(f"[INFO] Selected features: {list(selected_columns)}")

    # Apply SMOTE
    smote = SMOTE(random_state=42)
    X_resampled, y_resampled = smote.fit_resample(X_selected, y)

    # Standardize
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X_resampled)
    return X_scaled, y_resampled, scaler, selector

def evaluate_model(name, model, X_test, y_test):
    preds = model.predict(X_test)
    report = classification_report(y_test, preds, target_names=['Normal', 'Anomaly'])
    f1 = f1_score(y_test, preds, pos_label=1)

    # Generate visualizations
    generate_visualizations(y_test, preds, name)

    print(f"\n[INFO] {name} Evaluation:\n{report}")
    return f1, report, preds
```

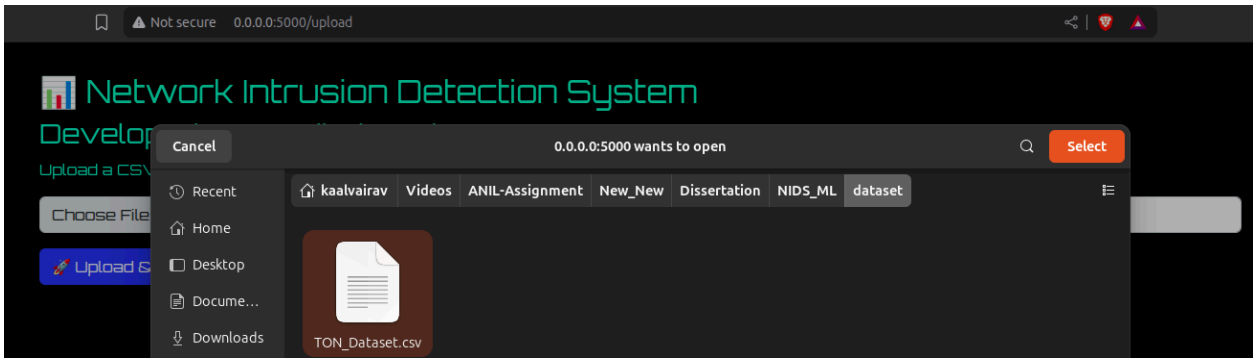
```python
def generate_visualizations(y_true, y_pred, model_name):
    # Confusion Matrix
    cm = confusion_matrix(y_true, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Normal", "Anomaly"])
    disp.plot(cmap="Blues")
    plt.title(f"Upload_Confusion Matrix - {model_name}")
    plt.savefig(f"upload_confusion_matrix_{model_name}.png")
    plt.close()

    # Metrics Bar Plot
    report = classification_report(y_true, y_pred, output_dict=True)
    df_report = pd.DataFrame(report).transpose().drop(["accuracy", "macro avg", "weighted avg"])

    df_report[["precision", "recall", "f1-score"]].plot(
        kind="bar",
        figsize=(8, 5),
        color=["skyblue", "orange", "green"]
    )
    plt.title(f"Evaluation Metrics by Class - {model_name}")
    plt.ylabel("Score")
    plt.xticks(rotation=0)
    plt.ylim(0, 1.05)
    plt.grid(axis='y')
    plt.tight_layout()
    plt.savefig(f"upload_metrics_{model_name}.png")
    plt.close()

def save_model(model, scaler, selector, filename="model.pkl"):
    joblib.dump((model, scaler, selector), filename)
    print(f"[INFO] Best model saved to {filename}")

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Train and Evaluate Multiple Models for NIDS")
    parser.add_argument("dataset", help="Path to the CSV dataset")
    parser.add_argument("--label", required=True, help="Label column name (e.g., 'label')")
    parser.add_argument("--output", default="model.pkl", help="Path to save the best model")
    args = parser.parse_args()

    print("[INFO] Loading dataset...")
    X, y = load_dataset(args.dataset, args.label)

    print("[INFO] Preprocessing features and balancing data...")
    X_train_scaled, y_train, scaler, selector = preprocess_features(X, y)

    print("[INFO] Splitting into train/test sets...")
    X_train, X_test, y_train_split, y_test = train_test_split(
        X_train_scaled,
        y_train,
        test_size=0.2,
        random_state=42
    )
```

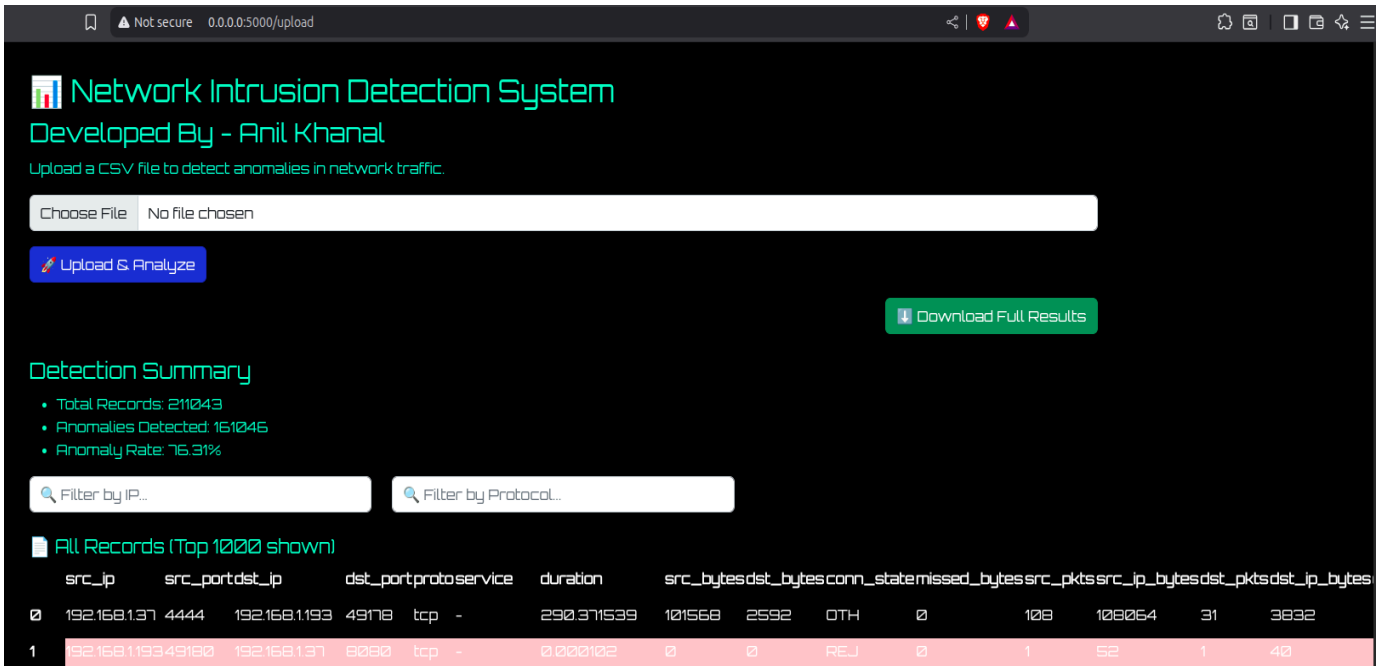
```python
# === Load and preprocess TON dataset ===
df = pd.read_csv("dataset/TON_Dataset.csv")

# Replace 'proto' with numeric mapping for real-time compatibility
proto_mapping = {proto: idx for idx, proto in enumerate(df['proto'].unique())}
df['proto_num'] = df['proto'].map(proto_mapping)

# Calculate 'length' as src_bytes + dst_bytes (to simulate packet length)
df['length'] = df['src_bytes'] + df['dst_bytes']

# Select real-time compatible features
realtime_features = ['src_port', 'dst_port', 'proto_num', 'length']
df_realtime = df[realtime_features].fillna(0)

# Scale and train
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_realtime)

model = IsolationForest(n_estimators=100, contamination=0.05, random_state=42)
model.fit(X_scaled)

# Save model and scaler for real-time use
joblib.dump((model, scaler, proto_mapping), "realtime_model.pkl")
print("[✅] Real-time model and scaler saved as 'realtime_model.pkl'")
```

Screenshots of the System

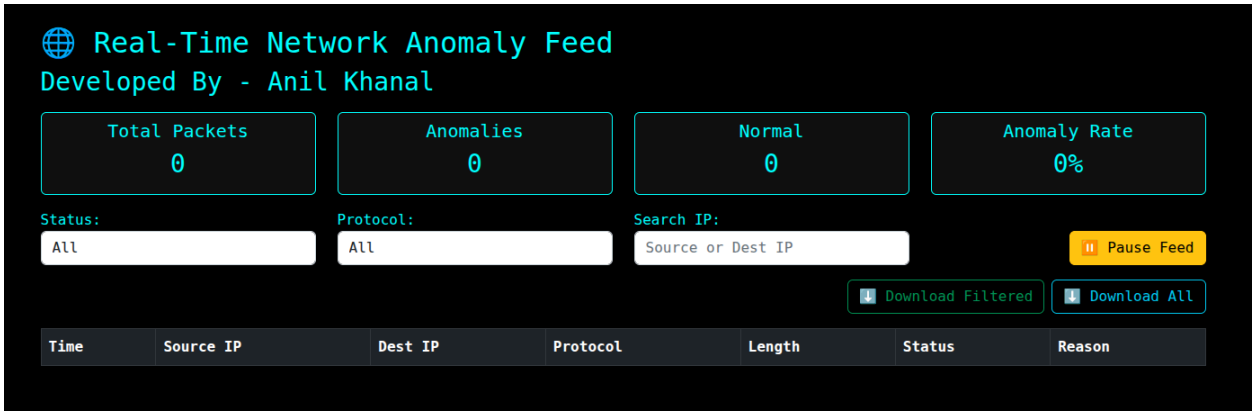


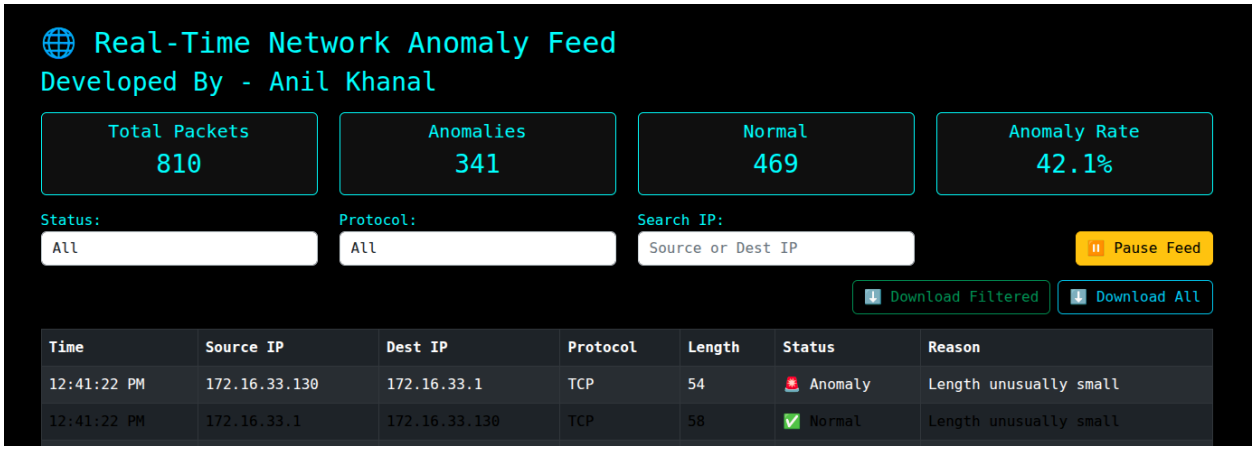

Not secure 0.0.0.0:5000/upload
Network Intrusion Detection System
Developed By - Anil Khanal
Upload a CSV file to detect anomalies in network traffic.
Choose File
No file chosen
Upload & Analyze

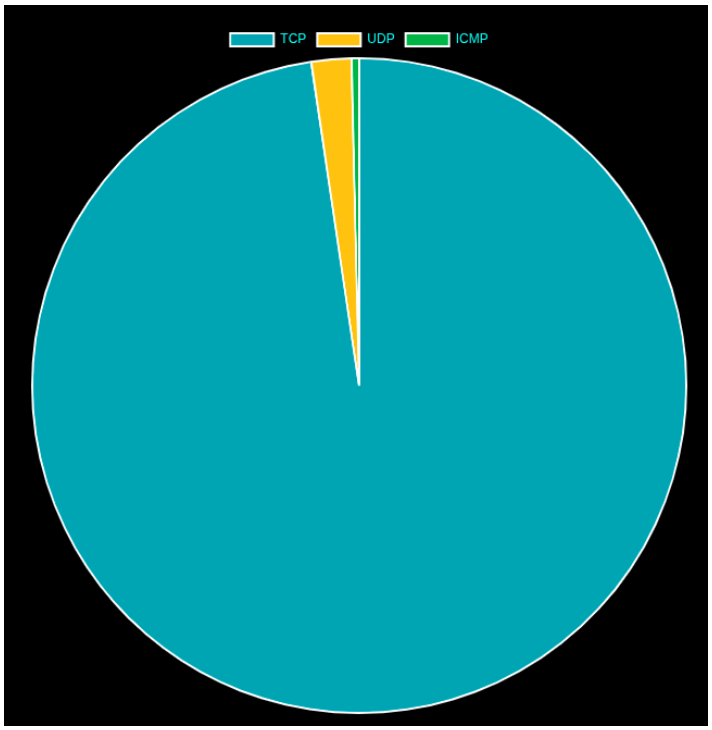

Not secure 0.0.0.0:5000/upload
Network Intrusion Detection System
Cancel
0.0.0.0:5000 wants to open
Select
Recent
Home
Desktop
Downloads
kaalvairav
Videos
ANIL-Assignment
New_New
Dissertation
NIDS_ML
dataset
TON_Dataset.csv


Not secure 0.0.0.0:5000/upload
Network Intrusion Detection System
Developed By - Anil Khanal
Upload a CSV file to detect anomalies in network traffic.
Choose File
No file chosen
Upload & Analyze
Download Full Results
Detection Summary
Total Records: 211043
Anomalies Detected: 161046
Anomaly Rate: 76.31%
Filter by IP...
Filter by Protocol...
All Records (Top 1000 shown)
src_ip src_port dst_ip dst_port proto service duration src_bytes dst_bytes conn_state missed_bytes src_pkts src_ip_bytes dst_pkts dst_ip_bytes
0 192.168.1.37 4444 192.168.1.193 49178 tcp - 290.371539 101568 2592 OTH 0 108 108064 31 3832
1 192.168.1.193 49180 192.168.1.37 8080 tcp - 0.000102 0 0 REJ 0 1 52 1 40

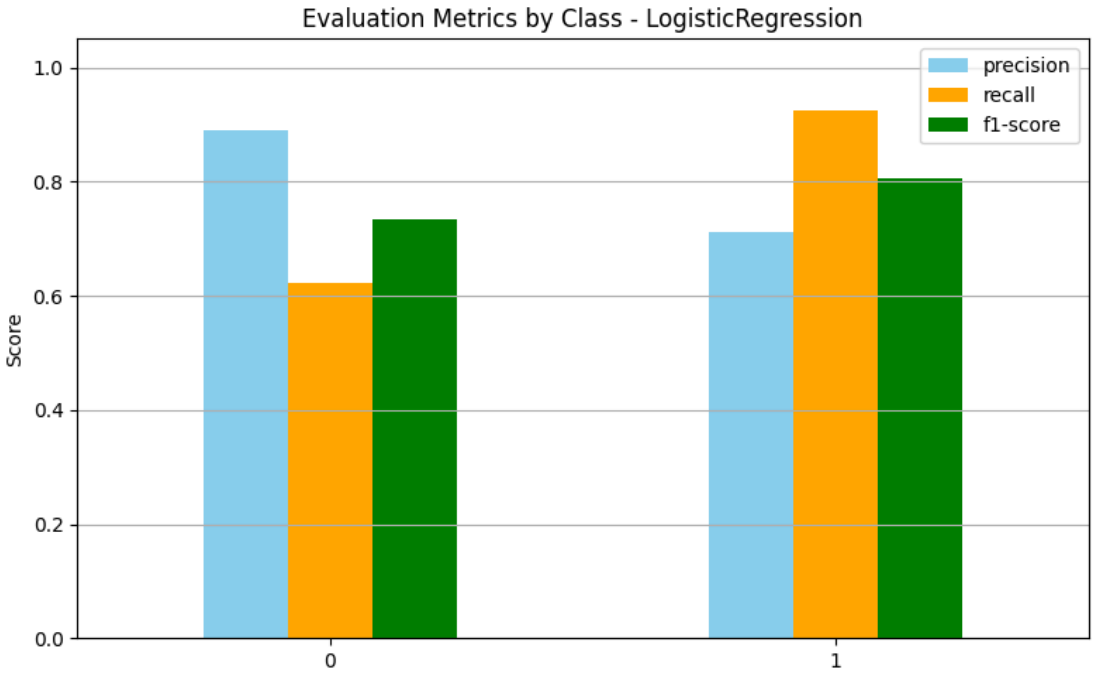

Visualizations
Normal
Anomaly
Anomalies by Protocol
142,800
132,600
122,400
112,200
102,000
91,800
81,600
71,400
61,200
51,000
40,800
30,600
20,400
10,200
0
icmp
tcp
udp

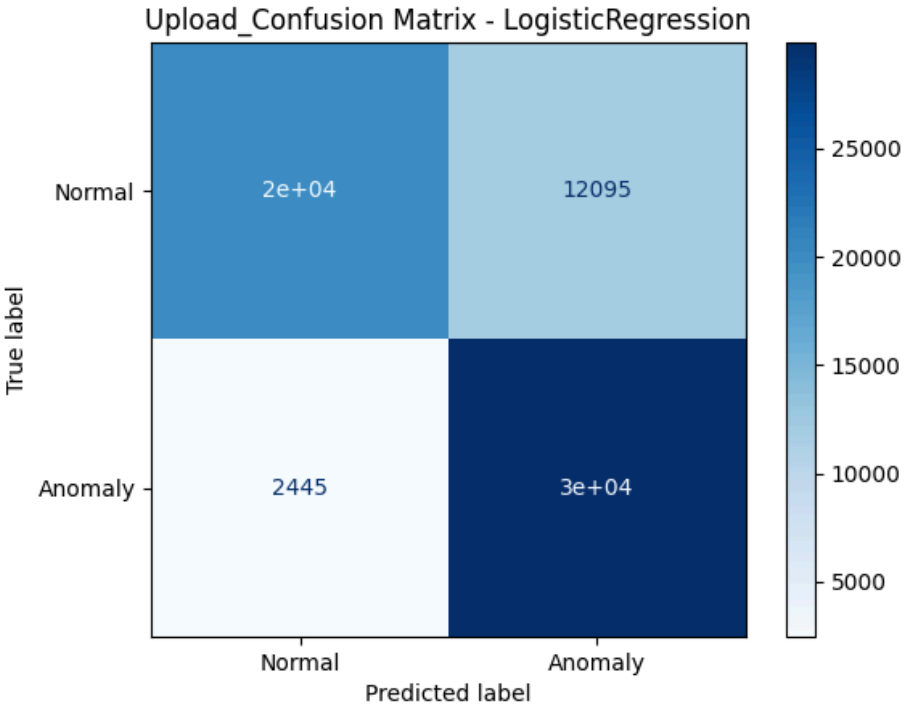

Real-Time Network Anomaly Feed
Developed By - Anil Khanal
Total Packets
0
Anomalies
0
Normal
0
Anomaly Rate
0%
Status:
All
Protocol:
All
Search IP:
Source or Dest IP
Pause Feed
Download Filtered
Download All
Time
Source IP
Dest IP
Protocol
Length
Status
Reason

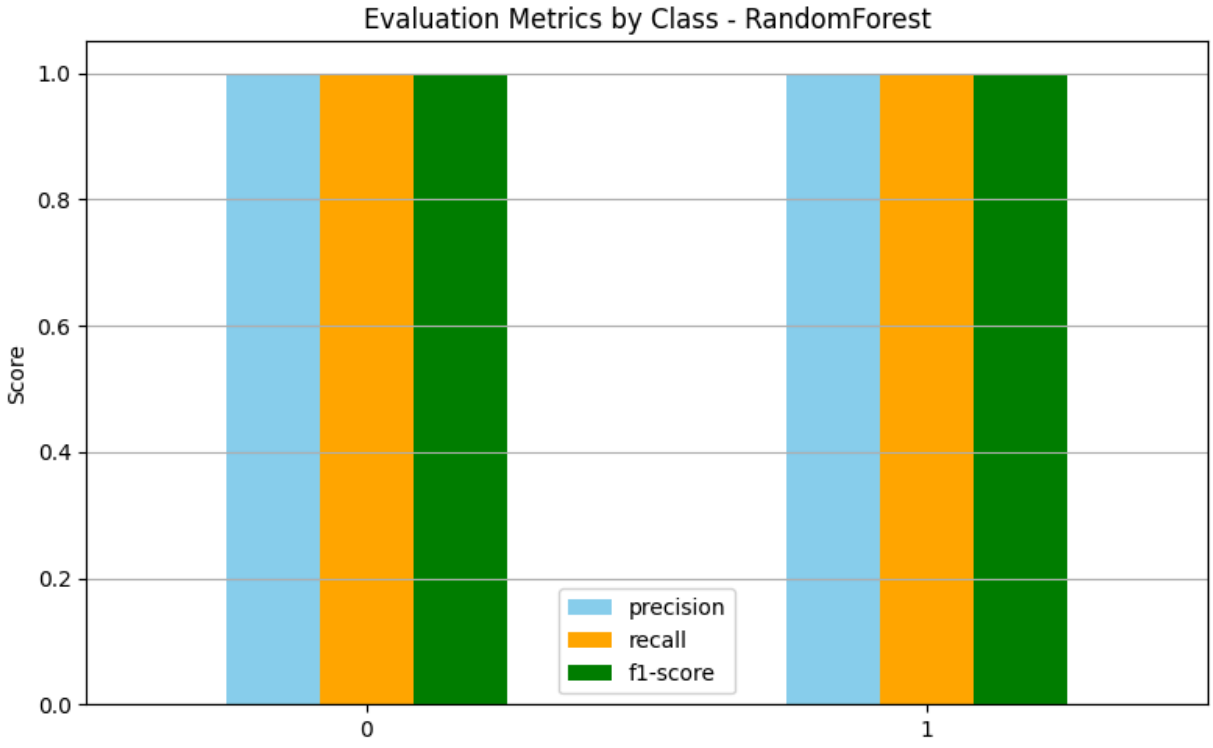

Real-Time Network Anomaly Feed
Developed By - Anil Khanal
Total Packets
810
Anomalies
341
Normal
469
Anomaly Rate
42.1%
Status:
All
Protocol:
All
Search IP:
Source or Dest IP
Pause Feed
Download Filtered
Download All
Time
Source IP
Dest IP
Protocol
Length
Status
Reason
12:41:22 PM
172.16.33.130
172.16.33.1
TCP
54
Anomaly
Length unusually small
12:41:22 PM
172.16.33.1
172.16.33.130
TCP
58
Normal
Length unusually small

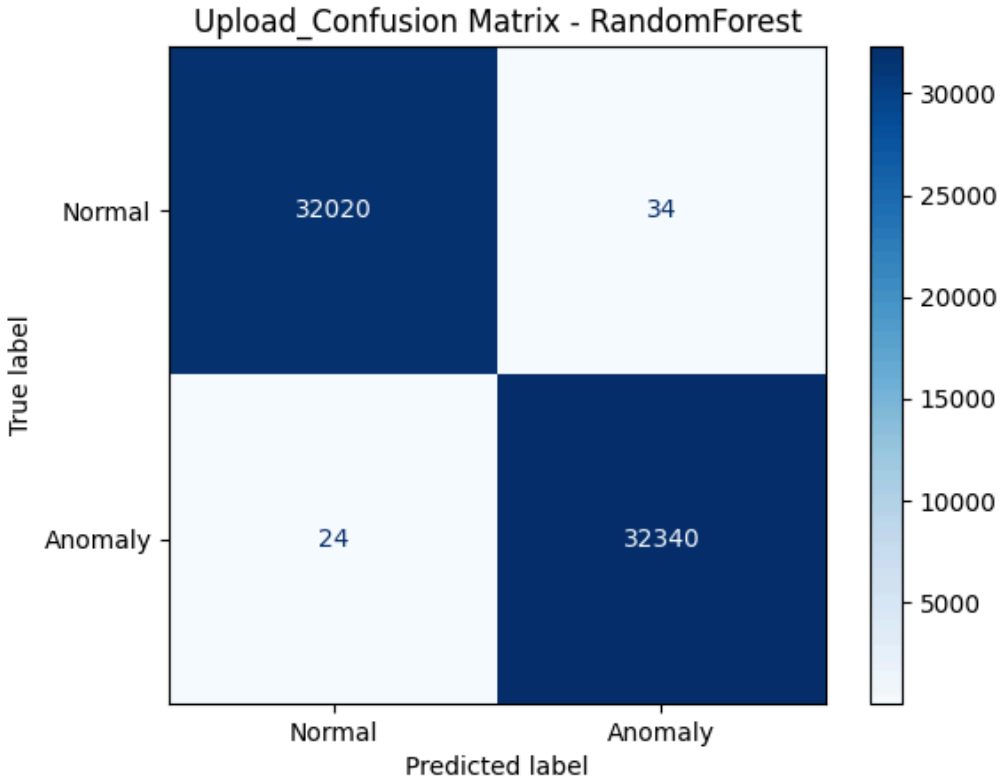

TCP
UDP
ICMP

Additional Results






Evaluation Metrics by Class - RandomForest
Score
1.0
0.8
0.6
0.4
0.2
0.0
precision
recall
f1-score
0
1


Upload_Confusion Matrix - RandomForest
Normal
Anomaly
True label
32020
34
24
32340
Normal
Anomaly
Predicted label
30000
25000
20000
15000
10000
5000

Deployment Instructions

For the application to work properly follow the steps mentioned below:

Installation

- pip install scapy pandas scikit-learn joblib flask flask-socketio eventlet matplotlib seaborn

Train Models

- python train_model.py dataset --label label
- python train_realtime_model.py

Run Application

- **sudo python app.py**

Note: use “**Sudo -E env "PATH=$PATH" python app.py**” if an environment was used.

- Open the browser and go to 0.0.0.0:5000