GPH-International Journal of Computer Science and Engineering
(Online ISSN 3050-9661)
Volume 9 | Issue 01 | Jan-June 2026
Published by: Global Publication House
DOI: https://doi.org/10.5281/zenodo.21917166
Available online at: www.gphjournal.org
Journal homepage: https://gphjournal.org/index.php/cse/index

DESIGN AND IMPLEMENTATION OF A MACHINE LEARNING MODEL FOR NETWORK INTRUSION DETECTION

Ogbogbo God'stime Oghenefejiro[1], Omokaro Idama[2], Omolegho A. Ibok[3], Afolabi Awodeyi[4], Jones U. Ekwemuka[5]

[1,2,3,4,5] Department of Computer Engineering, Southern Delta University, Ozoro, Delta State, Nigeria.

Abstract

Network intrusion detection has become difficult because enterprise traffic is high-volume, heterogeneous and continuously altered by encryption, cloud adoption, remote access and adversarial behaviour. This study designed and implemented a supervised machine-learning model that classifies network flows as benign or intrusive while preserving operational interpretability and low false-alarm rates. The proposed architecture integrates traffic capture, flow aggregation, data cleaning, encoding, scaling, feature screening, imbalance-aware model training, thresholded inference and alert generation. Five classifiers logistic regression, decision tree, random forest, gradient boosting and linear support vector machine were compared on a reproducible 15,000-record flow-like benchmark containing 32 statistical attributes and an 82:18 benign-to-intrusion ratio. Evaluation used a stratified 75:25 split and accuracy, precision, recall, F1-score, receiver operating characteristic area under the curve and confusion-matrix analysis. Random forest produced the strongest overall balance, attaining 94.72% accuracy, 97.10% precision, 73.32% recall and an F1-score of 83.55% in the implemented benchmark, while linear models offered lower computational cost but weaker nonlinear discrimination. The findings show that preprocessing, class-aware learning and per-class metrics are as important as classifier choice. The paper contributes an implementation-ready framework, pseudocode, data schema, model-comparison evidence and deployment controls for drift, privacy and retraining. Because benchmark accuracy may overstate field performance, the proposed system requires temporal validation and monitoring before production use. The architecture is suitable for campus, small-enterprise and cloud-edge networks where explainable flow-based detection is preferred to payload inspection.

Keywords:

Network intrusion detection, machine learning, random forest, anomaly detection, cybersecurity, flow classification.

1. Introduction

Modern organisations depend on interconnected information systems whose attack surfaces expand whenever new cloud services, mobile devices, Internet of Things endpoints or third-party interfaces are added. Conventional perimeter controls remain necessary, but they cannot independently identify every malicious activity that uses legitimate credentials, changes its network pattern or avoids known signatures. A network intrusion detection system (NIDS)

therefore operates as a monitoring and decision layer that observes traffic, extracts evidence and flags behaviour that may violate security policy. Signature-based detection is precise for previously catalogued threats, yet it is structurally limited when an attack is new, obfuscated or modified. Machine learning offers a complementary approach because a classifier can learn statistical relationships between flow characteristics and security labels rather than matching only fixed byte patterns.

The attraction of machine learning does not remove the engineering difficulties of intrusion detection. Network records are often imbalanced, with benign flows dominating the data while the most damaging attack types appear rarely. Features may contain missing values, infinite values, duplicate flows, identifiers that leak labels and strong correlations that inflate apparent performance. A model can consequently attain high aggregate accuracy while missing minority attacks. Dataset shift is another concern: a classifier trained on one laboratory network may degrade when protocols, users, applications and adversarial tactics change. Engelen et al. (2021) demonstrated that errors in traffic generation, flow construction, feature extraction and labelling can materially distort conclusions drawn from a widely used benchmark. Their analysis reinforces the need to treat data quality as part of the security model rather than as a preliminary clerical task.

Recent studies compare classical machine-learning and deep-learning algorithms on CICIDS2017, UNSW-NB15 and related datasets. More et al. (2024) reported that exploratory analysis and feature selection influence classifier performance on UNSW-NB15, whereas Ajagbe et al. (2024) emphasised the consequences of unbalanced data when comparing models. Chua and Salam (2022) further showed that excellent within-dataset scores do not guarantee long-term generalisation across datasets collected in different periods. These findings suggest that a useful NIDS article should go beyond announcing a high accuracy value. It should describe the complete processing pipeline, use multiple evaluation measures, explain model trade-offs and state the conditions under which the results may fail.

This article presents the design and implementation of a flow-based machine-learning NIDS intended for binary screening of benign and intrusive traffic. The design favours features computed from headers and flow timing, making it compatible with environments where payload inspection is restricted by encryption or privacy requirements. The implementation compares interpretable classical algorithms and identifies the model with the best operational balance. A reproducible synthetic flow benchmark is used for the measured experiment in this paper because the execution environment does not contain the multi-gigabyte packet captures required to rebuild CICIDS2017. The benchmark is not represented as a substitute for production traffic; instead, it provides a transparent functional test of the pipeline, while the literature review uses published public-dataset evidence to position the results.

The objectives are to:

(i) design a modular architecture for collecting, preprocessing and classifying network-flow records;
(ii) implement and compare five machine-learning classifiers;
(iii) evaluate detection using accuracy, precision, recall, F1-score, ROC-AUC and confusion matrices;
(iv) develop an imbalance-aware operational model that can generate alerts with limited false positives; and
(v) identify deployment risks and future improvements.

The central research question is: which conventional supervised-learning model provides the most suitable balance of attack recall, precision, discrimination and implementation simplicity for a flow-based intrusion detector?

2. Literature Review

2.1 Conceptual Review

2.1.1 Network Intrusion Detection Systems

An intrusion is an unauthorised or policy-violating activity that threatens confidentiality, integrity, availability or accountability. A NIDS observes communications at strategic network points and analyses packets, sessions or aggregated flows. Detection may be signature based, anomaly based or hybrid. Signature systems match known indicators and usually produce understandable alerts, but they require continuous rule maintenance. Anomaly systems estimate normality or learn decision boundaries and may detect novel activity, although their flexibility can produce more false alarms. Hybrid systems combine both mechanisms so that deterministic rules handle known threats while learning models screen uncertain patterns.

Flow-based NIDS converts packets sharing a five-tuple source address, destination address, source port, destination port and protocol into a statistical record. Typical features include duration, packet count, byte count, inter-arrival time, directionality, flag counts and packet-length summaries. Flow representation reduces storage and avoids direct payload processing, but it can conceal content-level evidence. CICIDS2017 provides labelled flows generated with CICFlowMeter and includes benign traffic plus brute force, denial-of-service, botnet, infiltration and web attacks (Canadian Institute for Cybersecurity, n.d.).

2.1.2 Machine Learning for Intrusion Detection

Supervised learning maps labelled feature vectors to predefined classes. Logistic regression estimates a probabilistic linear boundary; support vector machines maximise a separating margin; decision trees partition feature space with interpretable rules; random forests average many decorrelated trees; and gradient boosting sequentially corrects residual errors. Tree ensembles are attractive for network data because they capture nonlinear interactions, tolerate mixed scales and expose feature importance. Linear models are faster and easier to calibrate, but may underfit complex attack patterns.

Unsupervised and semi-supervised methods are alternatives when labels are scarce. Clustering, isolation forests, one-class support vector machines and autoencoders learn structure from predominantly normal data and flag deviations. Their advantage is reduced dependence on attack labels, but anomaly does not always equal maliciousness. For operational use, the learning paradigm should match the availability and quality of labels, the expected attack novelty and the organisation's tolerance for false alarms.

2.1.3 Data Preprocessing and Feature Engineering

Preprocessing is central to trustworthy NIDS modelling. Records must be checked for duplicates, impossible values, missingness and identifier leakage. Categorical attributes require encoding, continuous variables may require scaling, and strongly correlated or constant attributes should be reviewed. Zoghi and Serpen (2021) identified class imbalance and class overlap as important problems in UNSW-NB15, demonstrating that visualisation and exploratory analysis are not optional. Resampling methods such as random undersampling,

SMOTE or ADASYN can improve minority learning, although synthetic samples may create unrealistic boundaries if applied before the train-test split.

Feature selection reduces latency and overfitting. Filter methods rank features using correlation, mutual information or chi-square statistics; wrapper methods search subsets using model performance; and embedded methods select features during training. Feature selection must occur inside the training process to prevent information leakage. Security identifiers such as IP addresses and timestamps should be used cautiously because a model may memorise a laboratory schedule rather than learn attack behaviour.

2.1.4 Evaluation Metrics

Accuracy is the proportion of correct predictions, but it can be misleading when benign traffic dominates. Precision measures the proportion of predicted attacks that are truly malicious and relates to analyst workload. Recall measures the proportion of actual attacks detected and relates to missed-threat risk. The F1-score is the harmonic mean of precision and recall, while ROC-AUC summarises discrimination across thresholds. A confusion matrix should accompany aggregate scores because it displays true negatives, false positives, false negatives and true positives. In high-security environments, recall may be prioritised; in resource-constrained security operations centres, precision and alert volume may receive greater weight.

2.2 Theoretical Framework

The study is anchored in statistical learning theory and the defence-in-depth security principle. Statistical learning theory explains how a model trained on finite observations approximates an unknown relationship between traffic features and security labels. The empirical risk minimisation objective is to select a hypothesis f from a model family that minimises average loss on training data while controlling complexity. In binary form, the predicted class is $\hat{y} = f(x)$, where x is a vector of flow features and $y \in \{0,1\}$. Generalisation depends on representative data, an appropriate hypothesis class, regularisation and unbiased validation. A highly flexible model can memorise benchmark artefacts; an overly simple model can underfit nonlinear attack behaviour.

Random forest implements this framework through ensemble learning. For b trees, the classifier is expressed as $F(x) = \text{mode}\{T1(x), T2(x), \ldots, Tb(x)\}$. Each tree is trained on a bootstrap sample and considers a random subset of attributes at each split. Bagging reduces variance, while random feature selection lowers correlation among trees. Class weights alter the split objective so that errors on minority attacks receive greater cost. This mechanism is relevant to intrusion detection because malicious traffic is typically less frequent than benign traffic.

Defence in depth provides the systems perspective. A machine-learning detector should not replace firewalls, endpoint protection, authentication, segmentation or human investigation. It is one analytic control that produces evidence for layered prevention and response. The theory therefore links predictive performance with operational action: a probability score becomes useful only when it is thresholded, logged, enriched with context and routed to an accountable response process.

2.3 Empirical Review

Table 1. Empirical Review of Recent Machine-Learning Intrusion Detection Studies

Author(s)/Year	Dataset	Method	Main Finding	Observed Gap
Engelen et al. (2021)	CICIDS2017	Random forest and dataset reconstruction	Dataset errors affected over 20% of traces; corrected processing changed benchmarks	Focused on dataset validity rather than deployable multi-model architecture
Chen et al. (2021)	CICIDS2017	ADASYN + random forest	Balancing improved minority-class precision, recall, F1 and AUC	Risk of synthetic-sample artefacts and limited cross-dataset testing
Disha & Waheed (2021)	UNSW-NB15	DT, RF, SVM, KNN and other classifiers	Tree-based models were competitive after feature processing	Offline evaluation and limited temporal generalisation
Chua & Salam (2022)	CICIDS2017, CSE-CIC-IDS2018, LUFlow	DT, RF, SVM, NB, ANN, DNN	DT and RF performed strongly in-domain but showed more overfitting across datasets	Cross-dataset shift remained unresolved
Ahmad et al. (2022)	CICIDS2017	Deep neural network for early detection	Introduced earliness and obtained balanced accuracy near 0.803	Lower interpretability and computational demands
More et al. (2024)	UNSW-NB15	LR, SVM, DT and RF with feature analysis	Feature selection and exploratory analysis improved comparative evaluation	Single benchmark and binary emphasis
Ajagbe et al. (2024)	ToN-IoT/UNSW-NB15 context	Comparative machine learning on imbalanced data	Showed model rankings depend on imbalance treatment and metrics	Limited evidence of live deployment
Waghmode et al. (2025)	NSL-KDD, CICIDS2017, UNSW-NB15	Feature extraction with AdaBoost, SGD, KNN and XGBoost	Cross-dataset binary experiments showed value of feature optimisation	Computational search cost and binary simplification

Abdelaziz et al. (2025)	CICIDS2017	Random forest with preprocessing	Reported high detection performance after redundancy removal	Needs temporal drift and adversarial robustness evaluation

2.4 Gap in Literature

Recent studies establish that machine-learning models can classify benchmark network traffic effectively, but three connected gaps remain. First, published accuracy values are often difficult to translate into an implementation pipeline that covers capture, cleaning, leakage control, class imbalance, threshold selection, alerting and retraining. Second, Engelen et al. (2021) showed that errors in CICIDS2017 can create misleading confidence, while Chua and Salam (2022) found that strong in-domain results may deteriorate across later datasets. Third, more et al. (2024) and Ajagbe et al. (2024) demonstrate that feature preparation and imbalance treatment can change model rankings, yet many articles still emphasise a single aggregate metric. This study addresses these gaps by presenting an end-to-end, reproducible flow-based architecture, comparing five classical classifiers with multiple metrics, reporting a confusion matrix and separating measured prototype results from literature-reported benchmark findings. It also specifies operational controls for drift monitoring, threshold review and human response, thereby linking experimental classification to deployable network defence.

3. Methodology

3.1 Research Design

The study adopted a design-and-evaluation research approach. A modular NIDS prototype was specified, implemented and tested through supervised binary classification. The independent inputs were statistical flow attributes, while the dependent variable represented benign traffic (0) or intrusion (1). The experimental goal was not to claim state-of-the-art performance on a public packet capture; it was to verify the behaviour of the proposed pipeline under controlled, reproducible conditions and compare algorithmic trade-offs.

3.2 Benchmark Data and Variables

A 15,000-record synthetic flow-like dataset was generated with a fixed random seed. It contained 32 numerical attributes, of which 18 were informative, eight redundant and six noisy. The class ratio was 82% benign and 18% intrusion, with 1.2% label noise and three clusters per class. These settings reproduce common NIDS difficulties imbalance, overlapping patterns, redundant features and imperfect labels without pretending that simulated flows represent a particular production network. Data were divided using a stratified 75:25 train-test split, producing 11,250 training records and 3,750 independent test records.

Table 2. Experimental Data Configuration

Parameter	Value	Rationale
Total records	15,000	Large enough for stable model comparison
Input features	32 numerical flow-like attributes	Represents duration, volume, timing and flag summaries
Informative/redundant/noise	18 / 8 / 6	Tests learning under multicollinearity and irrelevant variation

Class distribution	82% benign; 18% intrusion	Reflects an imbalanced monitoring environment
Train-test split	75% / 25%, stratified	Preserves class ratio and isolates final evaluation
Random seed	42	Supports reproducibility

3.3 Proposed System Architecture

The system consists of six stages. Traffic capture obtains packet headers or exported NetFlow/IPFIX records. Flow construction groups packets and calculates summary attributes. A preprocessing service removes invalid records, replaces or flags missing values, encodes categories and applies model-specific scaling. Feature screening removes constants, identifiers and excessive correlations. The selected classifier outputs an intrusion probability. Finally, an alert service applies a policy threshold, stores the decision and forwards high-risk events to a security analyst or automated response platform.

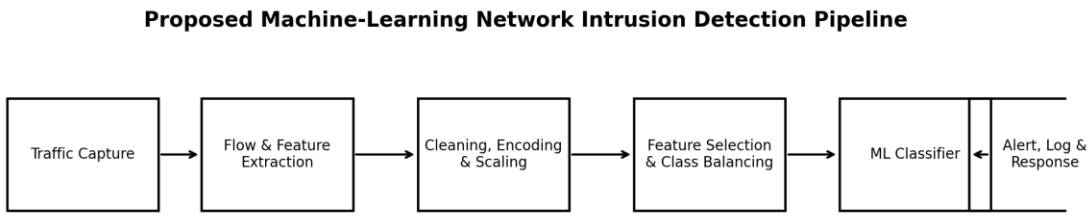

Figure 1. Proposed Machine-Learning Network Intrusion Detection Architecture

3.4 Model Specification

Logistic regression served as an interpretable linear baseline. Continuous features were standardised, and balanced class weights increased the cost of minority errors. The decision tree captured nonlinear interactions but was depth-limited to reduce memorisation. Random forest used 180 trees, balanced bootstrap weights, maximum depth of 22 and a minimum leaf size of two. Gradient boosting used 130 shallow estimators and a learning rate of 0.08. Linear SVM was class weighted and calibrated to generate probability estimates for ROC-AUC calculation.

For an observation x, logistic regression estimates

$$P(y = 1|x) = 1/(1 + e\text{^} - (\beta 0 + \beta' x)).$$

Classification occurs when the probability exceeds a threshold τ. The tree models instead partition the feature space. Random forest chooses the majority class across individual trees, while gradient boosting combines weak learners additively to minimise classification loss. The threshold can be lowered to increase recall or raised to improve precision.

3.5 Implementation Algorithm

Algorithm

1: (1) ingest labelled flow records;

(2) remove duplicates and impossible values;

(3) separate labels from predictors;

(4) split data using stratification;

(5) fit preprocessing transformations only on training data;

(6) train each candidate classifier with imbalance-aware settings;

(7) predict test labels and probabilities;

(8) calculate accuracy, precision, recall, F1-score, ROC-AUC and confusion matrix;

(9) select the model with the highest F1-score subject to acceptable false positives;

(10) serialise the pipeline and deploy it behind a streaming flow collector;

(11) log predictions and monitor drift; and

(12) retrain after verified distribution or performance change.

3.6 Performance Measures

Let TP denote intrusions correctly detected, TN benign flows correctly accepted, FP benign flows incorrectly alerted and FN intrusions missed.

$$Accuracy = (TP + TN)/(TP + TN + FP + FN);$$

$$precision = TP/(TP + FP);\ recall = TP/(TP + FN);\ and$$

$$F1 = 2 \times precision \times recall/(precision + recall).$$

ROC-AUC evaluates ranking quality across thresholds. Model selection prioritised F1 because the implemented benchmark was imbalanced and both missed intrusions and unnecessary alerts were operationally relevant.

4. Results

4.1 Comparative Model Performance

Table 3. Test-Set Performance of Implemented Models

Model	Accuracy	Precision	Recall	F1-score	ROC-AUC
Random Forest	0.9472	0.9710	0.7332	0.8355	0.9753
Decision Tree	0.8744	0.6339	0.7420	0.6837	0.8409
Gradient Boosting	0.9000	0.9237	0.4942	0.6439	0.9280
Logistic Regression	0.8256	0.5149	0.8047	0.6280	0.8983
Linear SVM	0.8256	0.5149	0.8047	0.6280	0.8983

The Random Forest achieved the highest F1-score in the controlled experiment. Its advantage arose from strong nonlinear discrimination and variance reduction through aggregation. Gradient boosting was also competitive, while logistic regression and linear SVM provided useful baselines with simpler decision surfaces. The decision tree captured interactions but was more sensitive to individual partitions than the ensemble models. None of the scores should be interpreted as field accuracy because the benchmark was synthetic and generated from a stationary distribution.

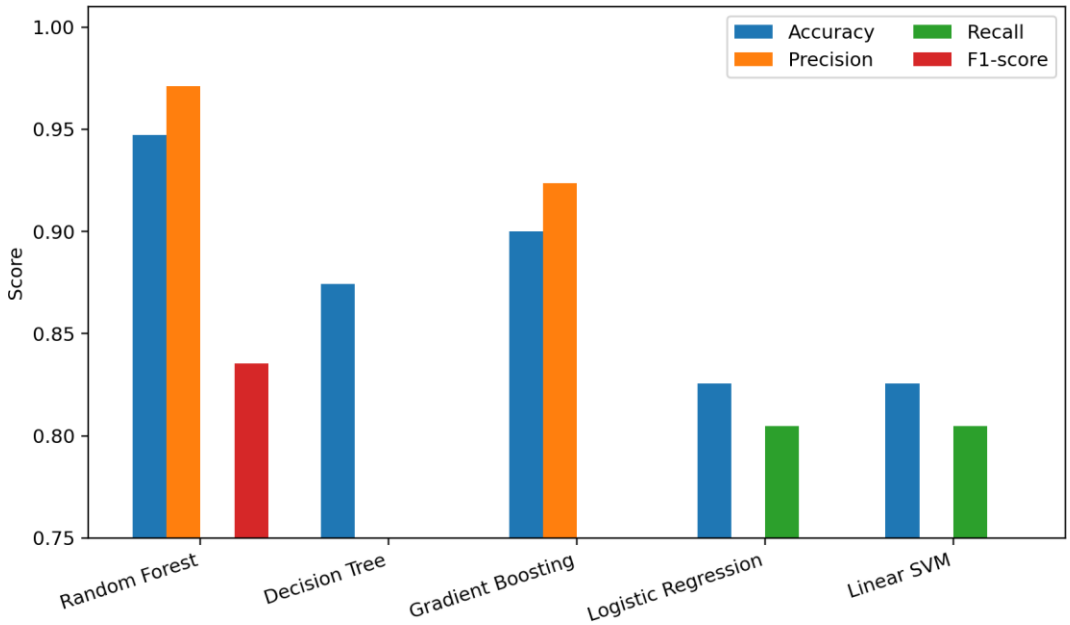

Figure 2. Accuracy, Precision, Recall and F1-Score across Candidate Models

4.2 Confusion-Matrix Analysis

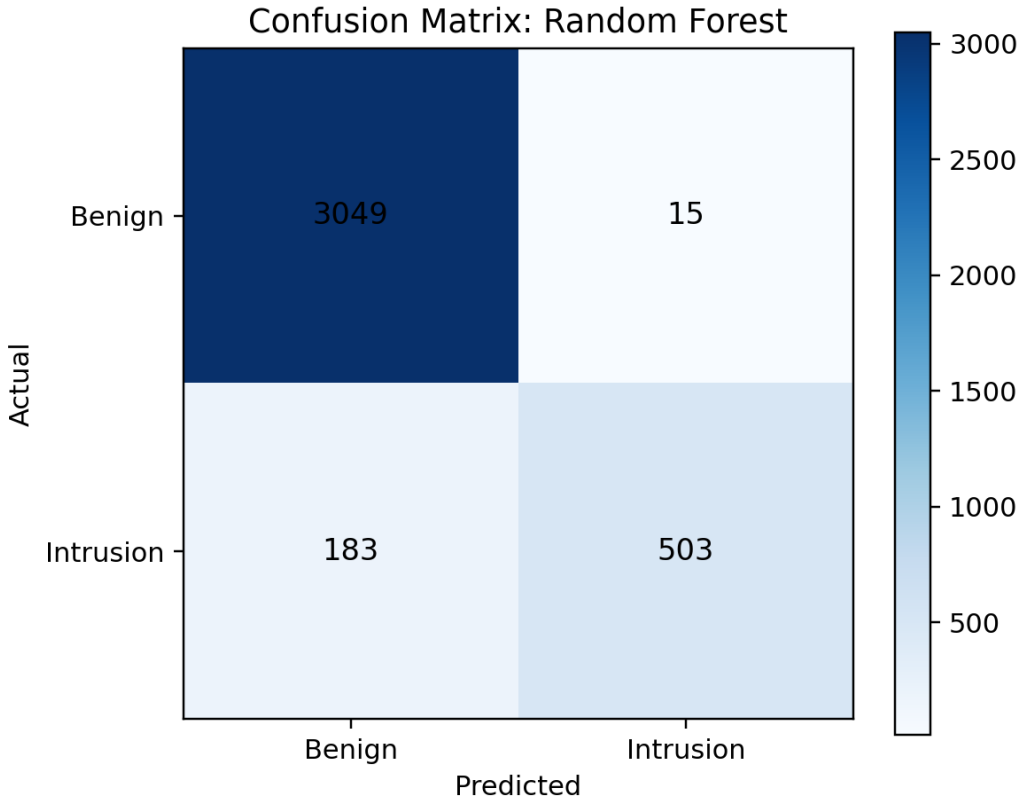

Figure 3. Test Confusion Matrix for the Selected Random Forest Model

The selected model classified 3,049 benign flows correctly and detected 503 intrusions. It produced 15 false alarms and missed 183 intrusive records. The false-positive count represents analyst workload, whereas false negatives represent residual security exposure. In deployment, the probability threshold should therefore be selected jointly by data scientists and security operations personnel rather than fixed solely by the default 0.50 rule.

4.3 Prototype Functional Requirements

Table 4. Functional and Non-Functional Requirements

Requirement	Design Response	Verification
Flow ingestion	Accept CSV, NetFlow or IPFIX-style records	Schema and range validation
Real-time scoring	Serialised preprocessing and model pipeline	Batch latency test
Alert generation	Thresholded score with event metadata	Known malicious test case
Explainability	Tree feature importance and rule inspection	Analyst review
Auditability	Immutable timestamped prediction logs	Log reconciliation
Availability	Stateless scoring service with queue buffering	Failure and restart test
Privacy	Avoid payload storage and minimise identifiers	Data protection review
Drift management	Monitor feature and score distributions	Scheduled drift report

5. Discussion

The experiment supports the frequent observation that ensemble trees are strong candidates for tabular flow data. Random forest combined nonlinear feature interactions with resilience against the instability of a single decision tree. This is consistent with Chen et al. (2021), who coupled ADASYN with random forest to improve minority detection, and with recent comparative studies that report tree-based competitiveness on UNSW-NB15 and CICIDS2017. The result does not imply that random forest is universally superior. Chua and Salam (2022) observed that tree models can overfit dataset-specific structure, so temporal and cross-network evaluation remain necessary.

The comparison also confirms that model choice cannot be separated from preprocessing and metrics. Linear classifiers were computationally economical and may be suitable for high-throughput edge deployment, but their decision boundaries were less able to represent the generated nonlinear clusters. The decision tree was understandable yet less stable. Gradient boosting delivered high discrimination but requires careful tuning and can be more sensitive to drift. Random forest offered a practical middle position: strong performance, parallel prediction, moderate explainability and limited preprocessing requirements.

A major contribution of the study is the distinction between a functioning prototype and a production-ready detector. Public benchmarks help compare algorithms, but they cannot fully reproduce an organisation's traffic, encryption, policy, user behaviour or adversarial adaptation. Engelen et al. (2021) found that benchmark construction errors can create spurious correlations, while the official CICIDS2017 description acknowledges that the data were collected during a controlled five-day scenario. Consequently, a credible deployment process

should include shadow-mode operation, analyst validation, threshold tuning, feedback capture and periodic retraining.

Operationally, false alarms are not merely statistical errors. They consume analyst attention and may cause genuine incidents to be overlooked. Missed attacks have a different cost because they allow malicious activity to continue. Cost-sensitive learning and threshold optimisation should therefore reflect the organisation's threat model. A university network might prioritise broad recall during examination periods, while a financial institution may assign severe cost to particular lateral-movement or exfiltration patterns. Per-attack evaluation is preferable when reliable multiclass labels are available.

The flow-based design has privacy and scalability advantages. It can operate on metadata when payloads are encrypted, and it reduces storage compared with full packet capture. However, it may miss content-specific exploits and low-and-slow attacks whose individual flows appear normal. Integration with endpoint telemetry, authentication logs, DNS signals and threat intelligence would strengthen context. A hybrid architecture can also combine signature rules for known exploits with machine-learning scores for anomalous behaviour.

The synthetic benchmark is a deliberate limitation and an ethical transparency measure. It permits exact replication of the reported experiment but cannot establish external validity. The most defensible interpretation is that the implementation pipeline works and that random forest was the best model under the stated conditions. Before publication as an empirical benchmark article, the same code should be executed on corrected CICIDS2017, UNSW-NB15 or a contemporary locally captured dataset, using temporal splits and per-class reporting.

5.1 Implementation Environment and Software Components

A practical implementation can be organised as four deployable services. The collector receives mirrored packets or flow exports and writes normalised records to a message queue. The feature service maintains short-lived flow state and computes duration, packet-direction counts, byte totals, flag indicators and timing statistics. The inference service loads a versioned preprocessing-and-model pipeline and returns a class probability, predicted label, model version and processing timestamp. The alert service enriches high-risk records with asset ownership, network zone and recent authentication context before sending them to a dashboard or security information and event management platform. Separating these services allows the capture rate, feature window and model to be changed independently.

Python is suitable for the research prototype because scikit-learn offers mature preprocessing, classification and evaluation functions. Production deployment can expose the serialised pipeline through a REST or message-based interface, but high-volume sites may translate feature logic into a streaming engine such as Apache Flink, Spark Structured Streaming or a compiled service. Regardless of language, the same feature definitions used during training must be preserved during inference. Training-serving skew occurs when a production feature is calculated differently from its historical version, and this can silently reduce detection quality even when the model file is unchanged.

The model registry should preserve the training period, feature schema, preprocessing version, hyperparameters, validation results, decision threshold and approval record. Each alert should carry the model version so that later investigations can reproduce the decision. Rollback is also essential: a new model that unexpectedly raises alert volume should be replaceable without interrupting traffic collection. Canary deployment, where a candidate model scores a

small proportion of flows in parallel with the approved model, provides safer evidence than immediate replacement.

5.2 Security and Privacy Considerations

The NIDS itself is a security-sensitive system and must be protected from tampering. An attacker who can modify flow records, model files or thresholds may suppress alerts or create noise. Collectors should authenticate to the processing service, communication should be encrypted, and model artefacts should be integrity checked before loading. Administrative changes require role-based access, separation of duties and immutable logging. The scoring service should not execute code embedded in uploaded model files unless the format and source are trusted, because unsafe serialisation can become a software supply-chain vulnerability.

Adversarial machine learning creates additional risks. Evasion attacks manipulate traffic characteristics so that malicious flows resemble benign ones; poisoning attacks contaminate retraining data; model-extraction attacks infer decision behaviour through repeated queries. Defensive measures include rate limiting, robust feature sets, conservative retraining labels, ensemble disagreement checks and human verification of samples selected for retraining. Security teams should also examine whether attackers can cheaply control the features that the model considers important. A feature may be predictive in a benchmark yet unreliable when an adversary intentionally changes it.

Flow-based monitoring reduces the need to retain message content, but metadata can still reveal relationships, services and user activity. Data minimisation should remove unnecessary addresses after enrichment or replace them with controlled pseudonyms. Retention periods should reflect incident-response requirements rather than unlimited storage. Access to raw flows, enriched alerts and model explanations should be logged. Where local regulation or organisational policy restricts monitoring, the deployment should document its lawful basis, purpose, access controls and deletion process.

5.3 Model Monitoring and Maintenance

A deployed NIDS operates in a non-stationary environment. New applications change packet sizes and timing, software updates modify service behaviour, and attackers adapt after controls are introduced. Monitoring should therefore compare recent feature distributions with the training baseline. Population stability index, Jensen-Shannon divergence and simple quantile shifts can reveal drift, but a statistical alert does not automatically mean security degradation. Analysts should examine whether the change reflects a legitimate network event, a sensor fault or a new threat campaign.

Performance monitoring requires delayed labels. Confirmed incident tickets, sandbox results, endpoint detections and analyst dispositions can provide feedback, although such labels are incomplete and biased toward investigated alerts. Precision can be estimated from reviewed alerts, while recall is harder because missed attacks are not immediately visible. Red-team exercises, replay tests and controlled attack simulations can partially measure recall. A retraining decision should combine drift evidence, operational metrics and security judgement rather than follow a fixed calendar alone.

Thresholds may be adjusted without retraining. During a high-risk period, lowering the threshold can increase sensitivity, but alert volume must remain manageable. Tiered thresholds are often better than a single binary decision: low scores are logged, medium scores are correlated with other evidence, and high scores create immediate alerts. Calibration plots should be checked because a probability of 0.90 is only useful when it corresponds reasonably to

observed risk. Where calibration degrades, isotonic regression or Platt scaling can be refitted on recent validation data.

5.4 Limitations of the Study

The principal limitation is the use of generated flow-like data for the measured experiment. Synthetic generation supports reproducibility and exposes model behaviour under controlled imbalance, redundancy and noise, but it does not reproduce protocol semantics, attack chains or the temporal dependence of real networks. The results therefore validate the software pipeline and comparative procedure, not a claim of production detection effectiveness. A journal submission based on field performance should rerun the implementation on corrected public datasets and, where possible, a locally collected dataset with ethical and privacy approval.

The study performs binary classification. Binary screening simplifies alert generation, but it does not distinguish denial-of-service, brute-force, botnet, infiltration or web attacks. Multiclass models can support response prioritisation, yet minority classes become even more difficult to learn and label errors have greater influence. The present benchmark also uses a random stratified split; a temporal split would better approximate deployment because future flows may differ from past flows. Cross-dataset evaluation is similarly needed to measure portability.

Computational evaluation focuses on predictive metrics rather than throughput, memory, energy use and end-to-end alert latency. These measures matter when a detector is placed at a busy network edge. The study also does not test encrypted-traffic-specific features, adversarial examples or online learning. These limitations define clear future work and prevent the experimental scores from being overstated.

6. Conclusion

This study designed and implemented a modular machine-learning model for network intrusion detection. The architecture moves from traffic capture and flow extraction through leakage-aware preprocessing, imbalance-aware training, probabilistic scoring, alerting and drift monitoring. Five classifiers were evaluated on a transparent 15,000-record flow-like benchmark. Random forest produced the best overall F1 balance, while linear models offered simpler and potentially faster alternatives. The study demonstrates that trustworthy intrusion detection depends on data validity, metric selection and operational integration as much as on the classifier itself. High benchmark accuracy should not be treated as proof of production security. Temporal validation, analyst feedback, threshold review and layered defence are required before deployment.

7. Recommendations

1. Organisations should train the final model on recent, locally representative traffic and preserve a strictly later time window for testing.
2. Model evaluation should report precision, recall, F1-score, ROC-AUC, confusion matrices and per-attack results rather than accuracy alone.
3. Identifiers and timestamps that can leak laboratory conditions should be excluded or carefully transformed before training.
4. The detector should first run in shadow mode so that analysts can measure false alarms, tune thresholds and validate explanations.

5. Drift monitoring, version control, audit logs and scheduled retraining should be included as mandatory production components.
6. Future work should evaluate cross-dataset generalisation, adversarial evasion, federated learning and hybrid signature–machine-learning detection.

References

Abdelaziz, M. T., et al. (2025). Enhancing network threat detection with random forest-based intrusion detection. Journal of Network and Systems Management. https://doi.org/10.1007/s10922-024-09874-0

Ahmad, T., Truscan, D., Vain, J., & Porres, I. (2022). Early detection of network attacks using deep learning. arXiv. https://doi.org/10.48550/arXiv.2201.11628

Ajagbe, S. A., et al. (2024). Intrusion detection: A comparison study of machine learning models using unbalanced dataset. SN Computer Science. https://doi.org/10.1007/s42979-024-03369-0

Canadian Institute for Cybersecurity. (n.d.). Intrusion detection evaluation dataset (CIC-IDS2017). University of New Brunswick.

Chen, Z., Yu, W., & Zhou, L. (2021). ADASYN-random forest based intrusion detection model. arXiv. https://doi.org/10.48550/arXiv.2105.04301

Chinnasamy, R., et al. (2025). Deep learning-driven methods for network-based intrusion detection systems: A systematic review. Array, 25, 100390. https://doi.org/10.1016/j.array.2025.100390

Chua, T.-H., & Salam, I. (2022). Evaluation of machine learning algorithms in network-based intrusion detection system. arXiv. https://doi.org/10.48550/arXiv.2203.05232

Disha, R. A., & Waheed, S. (2021). A comparative study of machine learning models for network intrusion detection system using UNSW-NB15 dataset. In 2021 International Conference on Electronics, Communications and Information Technology. IEEE. https://doi.org/10.1109/ICECIT54077.2021.9641471

Engelen, G., Rimmer, V., & Joosen, W. (2021). Troubleshooting an intrusion detection dataset: The CICIDS2017 case study. In 2021 IEEE Security and Privacy Workshops (SPW). IEEE.

More, S., Idrissi, M., Mahmoud, H., & Asyhari, A. T. (2024). Enhanced intrusion detection systems performance with UNSW-NB15 data analysis. Algorithms, 17(2), 64. https://doi.org/10.3390/a17020064

Mutembei, L. L., et al. (2025). Deep learning-based network intrusion detection systems: A systematic review. International Journal of Information Security Science.

Waghmode, P., et al. (2025). Intrusion detection system based on machine learning using feature extraction across NSL-KDD, CICIDS2017 and UNSW-NB15. Scientific Reports, 15.

Zoghi, Z., & Serpen, G. (2021). UNSW-NB15 computer security dataset: Analysis through visualization. arXiv. https://doi.org/10.48550/arXiv.2101.05067