

AI-BASED INTRUSION DETECTION SYSTEM FOR NETWORK SECURITY

Soniya S Dadhanian

Lecturer in Computer Engineering

R C Technical Institute, Ahmedabad

Abstract— *Modern networks face a continuously changing range of threats, including scanning, denial-of-service activity, credential attacks, malware communication, and lateral movement. Signature-based intrusion detection remains useful for known patterns but may miss novel or modified attacks. This paper presents a conceptual design and research methodology for an artificial intelligence (AI)-based intrusion detection system (IDS) that combines traffic and log collection, preprocessing, feature engineering, supervised classification, anomaly detection, and analyst feedback. The paper reviews representative approaches, proposes an evaluation framework, and discusses operational concerns such as class imbalance, data leakage, explainability, privacy, and model drift. No experimental results are claimed; instead, the evaluation table defines metrics and comparisons that can be completed after implementation on a documented dataset. The proposed approach treats AI as decision support within a layered security architecture rather than as a replacement for established controls.*

Index Terms— *intrusion detection, network security, machine learning, anomaly detection, cybersecurity, explainable AI.*

I. INTRODUCTION

Networked systems support banking, education, healthcare, government services, cloud platforms, and industrial operations. This connectivity expands access to information but also increases the number of possible entry points for attackers. Security teams must detect suspicious activity in large volumes of network flows, endpoint events, authentication records, and application logs. A delayed or missed detection can allow an attacker to maintain access, steal information, or disrupt services.

Traditional intrusion detection systems commonly rely on predefined signatures, rules, and thresholds. These methods are effective when an attack resembles a known pattern and the rules are maintained. However, they can struggle with new variants, encrypted traffic, subtle multi-stage attacks, and environments where normal behaviour changes frequently. Machine learning (ML) can learn patterns from labelled examples, while anomaly-detection techniques can identify deviations from a baseline. Both approaches have limitations: models can produce false alarms, inherit dataset bias, or fail when deployment traffic differs from training data.

This paper proposes an AI-assisted IDS architecture and a reproducible evaluation plan. The central research question is: how can supervised and anomaly-based models be combined to improve detection coverage while keeping false positives and operational complexity manageable? The contribution is a system design, a literature-informed discussion, and a practical evaluation framework—not a claim of measured performance.

II. BACKGROUND AND RELATED WORK

An IDS monitors events and identifies activity that may violate security policy. Network-based IDS (NIDS) inspects network packets or flow metadata; host-based IDS (HIDS) examines events from individual systems. A detection engine may be signature-based, anomaly-based, or hybrid. Signatures provide interpretable matches for known threats, whereas anomaly methods estimate how unusual an event is relative to learned behaviour.

Machine learning research has explored decision trees, random forests, support vector machines, k-nearest neighbours, naïve Bayes, clustering, and neural networks for intrusion detection. Tree ensembles can model nonlinear relationships and often provide a useful baseline for tabular flow features. Support vector machines can separate classes in engineered feature spaces but may require careful scaling and can be costly on large datasets. Neural networks can learn complex representations, although they require suitable training data, tuning, and monitoring. Unsupervised methods such as isolation forests or clustering can help surface previously unseen patterns, but an anomaly is not necessarily malicious.

Public datasets such as KDD Cup 1999, NSL-KDD, UNSW-NB15, and CICIDS2017 have been used in IDS studies. They differ in age, traffic generation, feature definitions, attack taxonomy, and realism. Older benchmark datasets may not represent modern networks, while newer datasets can still contain collection artefacts or class distributions unlike a live environment. Therefore, high benchmark accuracy

alone does not prove operational effectiveness. Data provenance, split strategy, per-class recall, false-alarm rate, and external validation should be reported.

A practical design should combine complementary signals rather than depend on a single model. Rules can handle high-confidence known indicators; supervised ML can classify patterns represented in labelled data; anomaly detection can prioritise unusual events for analyst review. Human validation and incident-response procedures remain necessary, particularly before automatically blocking traffic.

III. PROPOSED SYSTEM ARCHITECTURE

The proposed system consists of six logical stages: (1) collection of network-flow records and security logs; (2) normalization, validation, and removal of unusable records; (3) feature extraction and encoding; (4) model inference; (5) alert prioritization and analyst review; and (6) feedback, monitoring, and controlled retraining. Figure 1 illustrates the components and the feedback loop.

Figure 1. Conceptual architecture of an AI-based intrusion detection system

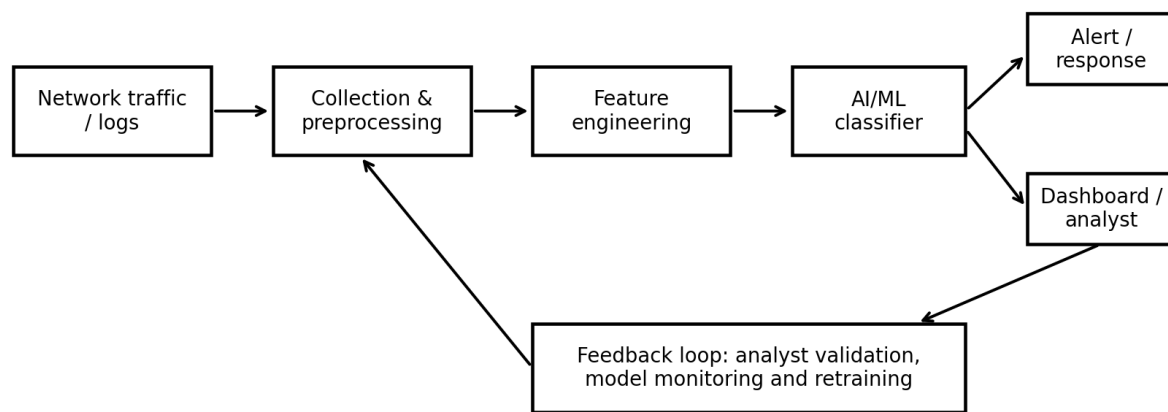


Fig. 1. Conceptual architecture of the proposed AI-based IDS.

Data collection should prefer flow metadata and relevant security events where full packet capture is unnecessary. Potential features include protocol, source/destination port, flow duration, packet and byte counts, connection state, failed-login frequency, and time-window aggregates. Personally identifiable information should be minimized, access-controlled, and retained only as required by policy. Encrypted payloads may be unavailable or inappropriate to inspect; metadata-based detection can still be useful but has limits.

Preprocessing must be fitted on training data only. Categorical variables may be encoded, numeric variables scaled where required, missing values handled consistently, and duplicate or invalid records examined. Features that directly reveal the label or encode information unavailable at prediction time must be removed. The trained model should output a class or risk score, with alert thresholds selected according to the cost of missed attacks and false alarms.

IV. METHODOLOGY

A. Research design and dataset

The study should begin by defining the deployment scenario, threat model, and unit of prediction (for example, a flow, time window, or host event). A suitable public benchmark can be selected based on feature availability, documentation, licensing, and attack coverage. The dataset version, download date, preprocessing code, class mapping, and exclusions should be recorded to support reproducibility. If possible, results should be validated on a second dataset or on a later time period.

B. Candidate models

A transparent baseline should be established before complex models are introduced. Recommended comparisons are: (i) a rule/signature baseline where available; (ii) a decision tree or random forest for supervised classification; (iii) a gradient-boosted tree model if tooling permits; and (iv) an anomaly detector such as Isolation Forest. A hybrid design can combine high-confidence rules, a supervised classifier, and anomaly scores. Model choice should reflect latency, interpretability, maintenance burden, and available computing resources—not accuracy alone.

C. Training and validation

Split data by time, capture session, device, or source where possible, rather than randomly splitting highly related records. A random split can place near-duplicates in both training and test sets and overstate generalization. Hyperparameters should be tuned on a validation set or through cross-validation confined to the training portion. The test set should remain untouched until final evaluation. Class weights or resampling may address imbalance, but resampling must occur only inside training folds.

Figure 2. Research and deployment workflow

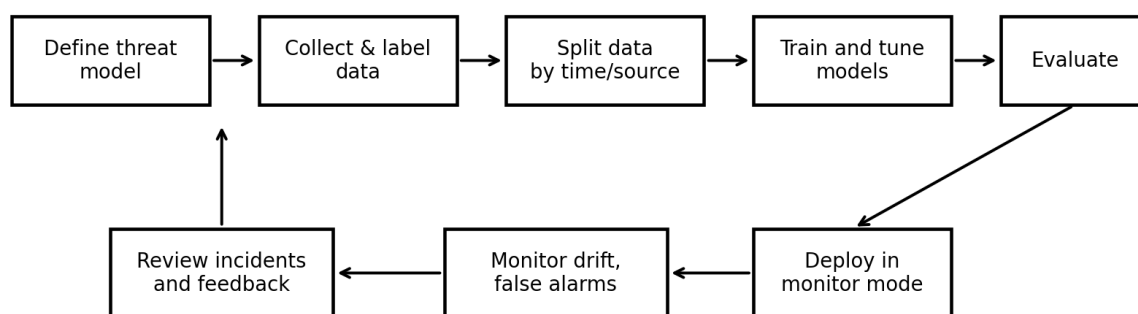


Fig. 2. Recommended research, validation, and deployment workflow.

D. Evaluation metrics

Overall accuracy is insufficient when benign traffic dominates. Report precision, recall, F1-score, confusion matrix, false-positive rate, and detection latency. For imbalanced data, precision-recall curves and macro-averaged scores can be more informative than accuracy alone. Results should be shown per attack category as well as overall. Operational evaluation should include alerts per hour, analyst workload, inference latency, memory use, and the proportion of alerts that lead to confirmed incidents.

E. Deployment safeguards

Initially run the model in monitor-only mode and compare alerts with analyst findings. Document threshold selection, model version, feature schema, and rollback steps. Automatic blocking should be introduced only for narrowly defined, high-confidence cases with safeguards and a recovery path. Monitor input distributions, false alarms, missed incidents, and model performance over time. Retraining should require data-quality checks, approval, testing, and versioned deployment.

V. EVALUATION PLAN AND EXPECTED ANALYSIS

The evaluation should compare candidate models on the same held-out records and under the same label definitions. Table I is a reporting template; its entries are not experimental findings. Replace the pending cells with measured results and include confidence intervals or repeated-run variation where feasible.

Model	Precision	Recall	F1-score	False-positive rate
Rule/signature baseline	Pending	Pending	Pending	Pending
Random Forest	Pending	Pending	Pending	Pending
Gradient-boosted trees	Pending	Pending	Pending	Pending
Isolation Forest / anomaly model	Pending	Pending	Pending	Pending
Hybrid rules + ML	Pending	Pending	Pending	Pending

TABLE I. Evaluation template (populate only after running the experiment).

The analysis should identify which attack classes are missed and whether false alarms cluster around legitimate but unusual behaviour, such as software updates or peak traffic. A model with a slightly lower F1-score may be preferable if it has lower latency, clearer explanations, and fewer disruptive false positives. Thresholds should be assessed against a realistic security-team capacity rather than selected solely to maximize a single metric.

VI. SECURITY, ETHICAL, AND OPERATIONAL CONSIDERATIONS

AI-based IDS deployments introduce risks beyond ordinary model error. Attackers may attempt evasion by changing traffic patterns or poisoning data used for retraining. Training data may underrepresent rare attacks, new devices, or particular network segments. Data leakage can make evaluation results misleading. Model outputs should therefore be treated as evidence for investigation, not as definitive proof of malicious intent.

Explainability can help analysts understand which features contributed to an alert, but explanations are approximate and should not be mistaken for causal proof. Access to logs and model outputs should follow least-privilege principles. Sensitive fields should be masked or excluded where possible, and retention should follow applicable law and organizational policy. A documented incident-response workflow should specify alert ownership, escalation, evidence preservation, and procedures for correcting false positives.

A further limitation is distribution shift: legitimate network behaviour changes with new applications, devices, and working patterns. Continuous monitoring and periodic revalidation are essential. Any retraining pipeline should be protected against untrusted labels and unauthorized model changes. Security controls such as patching, access control, segmentation, backups, and endpoint protection remain necessary; an IDS is one layer in a defence-in-depth strategy.

VII. LIMITATIONS

This paper provides a proposed architecture and evaluation methodology rather than empirical validation. It does not claim that one algorithm is universally superior or provide measured detection rates. Results will depend on the dataset, network environment, feature quality, threat model, and operating threshold. Public benchmarks cannot fully reproduce adversarial behaviour in a live enterprise. Future implementation should document these limitations and avoid generalizing beyond the tested conditions.

VIII. IMPLEMENTATION DETAILS AND SYSTEM MODULES

A. Data acquisition and event normalization

The data acquisition layer receives records from network sensors, routers, firewalls, intrusion prevention devices, servers, and authentication systems. Sources may use different timestamps, field names, encodings, and levels of detail. The collector should therefore normalize timestamps to a common time standard, map source-specific fields to a documented schema, and attach metadata describing the sensor and collection point. Clock synchronization is important because an investigation may depend on correlating events across systems. If a source becomes unavailable, the collector should record the gap rather than silently treating missing data as normal traffic.

B. Feature construction

Feature engineering converts raw records into measurable inputs suitable for a model. In addition to single-flow values such as duration, packet count, byte count, and protocol, window-based features can summarize repeated events. Examples include the number of connection attempts from a source during a short interval, the ratio of failed to successful logins, the number of distinct destinations contacted by one host, or the change in outbound data volume relative to a baseline. Window size should be selected according to the threat scenario: very short windows may capture bursts but be noisy, while long windows can hide short-lived attacks. Features must be available at prediction time and should not encode the target label indirectly.

C. Model service and alert generation

The inference service loads a versioned model and applies the same preprocessing steps used during training. Each prediction should be accompanied by a timestamp, model version, relevant input metadata, and a risk score where supported. Alerts can be prioritized using model confidence, asset criticality, threat intelligence, and the potential impact of the event. A low-confidence anomaly may be placed in a review queue, while a high-confidence match to a validated signature may trigger an established response playbook. The alert interface should group repeated events to reduce duplicate notifications and provide enough evidence for an analyst to reproduce the decision.

D. Feedback and model governance

Analyst feedback can improve the system when it is collected consistently. Useful labels include confirmed malicious, benign, inconclusive, and duplicate incident. The system should preserve the difference between an analyst's initial judgement and a later confirmed incident outcome. Feedback should not be inserted directly into training data without quality checks, because incorrect labels can degrade the model. Retraining should be version controlled and tested against a fixed regression suite before deployment. A rollback mechanism is necessary if a new model produces an unexpected rise in false positives or misses.

IX. THREAT MODEL AND ATTACK SCENARIOS

A. Threat assumptions

The proposed IDS assume that an organization has authorized visibility into selected network metadata and security logs. It aims to identify suspicious activity such as reconnaissance, brute-force attempts, denial-of-service patterns, malware command-and-control indicators, and unusual internal movement. It does not assume that every attack is visible in network flows. End-to-end encryption, missing sensors, compromised logging systems, and activity that resembles legitimate administration can limit detection. The IDS should be deployed alongside endpoint protection, identity controls, secure configuration, segmentation, patch management, and incident response.

B. Representative scenarios

During reconnaissance, a host may contact many ports or destinations in a pattern that differs from normal use. Repeated authentication failures may indicate password guessing, although legitimate users can also mistype credentials. A denial-of-service event may be reflected in unusual connection rates, packet volumes, or resource utilization. Malware communication may create repeated periodic connections to a remote endpoint, while lateral movement may appear as a workstation contacting servers it rarely accesses. These examples are indicators rather than definitive proof; each should be interpreted with asset role, user activity, and other security evidence.

C. Adversarial adaptation

Attackers may deliberately change timing, packet sizes, destination rotation, or other observable characteristics to evade a learned classifier. They may also attempt to poison a feedback or retraining process by generating traffic that is incorrectly labelled as benign. Defensive measures include keeping the feature pipeline and training process access-controlled, validating labels, monitoring sudden changes in input distributions, and testing models against realistic variations. A detector should not be treated as secure merely because it performs well on a static benchmark.

X. PRACTICAL DEPLOYMENT AND PERFORMANCE REQUIREMENTS

A. Latency and scalability

Detection latency is the time between an observable event and the creation of a useful alert. A model that is accurate but too slow for the traffic volume may not be operationally suitable. Performance testing should measure records processed per second, average and tail inference latency, memory consumption, and behaviour during traffic spikes. A scalable architecture can separate collection, feature processing, inference, and storage so that each stage can be adjusted independently. Batching may improve throughput, but it can add delay; the design must balance these requirements.

B. Alert fatigue and analyst workload

False positives consume analyst time and may cause genuine incidents to be overlooked. Evaluation should therefore report not only the false-positive rate but also the number of alerts per hour or day and the proportion that become confirmed incidents. Alert aggregation, suppression of repeated equivalent events, and risk-based prioritization can reduce workload. Suppression rules must be reviewed carefully so that they do not hide repeated low-volume attacks. Analysts should be able to inspect the evidence and record why an alert was closed.

C. Explainability and auditability

For tabular models, feature importance or local explanation methods may help analysts understand why a record received a high risk score. Such methods are explanatory aids, not proof of causation, and may be unstable when features are correlated. The system should maintain an audit trail containing model version, feature schema version, alert time, score, analyst disposition, and any response action. Sensitive payloads should not be copied into reports unless necessary and authorized. Auditability supports incident review, model comparison, and compliance checks.

D. Integration with incident response

The IDS should connect to a documented workflow for triage, escalation, evidence preservation, containment, eradication, and recovery. Before an automated response is enabled, the organization should define the assets and event types eligible for automation, the confidence threshold, approval requirements, and the procedure for reversing an action. A false automated block can disrupt business services, so monitor-only operation and controlled pilot deployment are recommended before broad enforcement.

XI. REPRODUCIBILITY AND RESEARCH VALIDITY

A. Avoiding data leakage

Data leakage occurs when training or evaluation uses information that would not be available in real operation. Examples include a feature derived from the label, preprocessing fitted on the full dataset, or near-duplicate flows split between training and test sets. To reduce this risk, fit encoders and scalers only on the training set, separate related records by capture session or time where possible, and keep the test set untouched until model selection is complete. All preprocessing decisions should be documented.

B. Fair comparison

A fair comparison uses the same data partitions, class definitions, and metric calculations for all candidate models. Hyperparameter search effort should be documented, and results should include the selected threshold. Where feasible, repeated runs or confidence intervals can show whether a small difference is meaningful. The paper should state hardware, software libraries, random seeds, dataset version, excluded fields, and class distribution. Negative results and classes with poor recall should be reported rather than omitted.

C. External validity

Performance on a public dataset may not transfer to a different organization because traffic mix, user behaviour, applications, and attack prevalence differ. A second dataset, time-based test, or carefully governed pilot can provide stronger evidence of generalization. If live data cannot be shared, privacy-preserving summaries and a reproducible evaluation script may still improve transparency. The final report should clearly separate benchmark performance from any production observations.

XII. FUTURE RESEARCH DIRECTIONS

A. Explainable and analyst-centred detection

Future research can investigate explanations that are concise enough for triage while remaining faithful to the model. Human-factors studies could measure whether explanations reduce investigation time or improve consistency between analysts. Explanations should be evaluated for stability and usefulness, not merely displayed as feature rankings.

B. Privacy-preserving learning

Organizations may be reluctant to centralize sensitive network data. Federated learning or privacy-preserving aggregation could allow multiple sites to contribute to model improvement without sharing all raw records. These approaches introduce their own challenges, including communication overhead, heterogeneous data distributions, and possible leakage through model updates. Their security and practical value require careful testing.

C. Continual learning and drift detection

A production network changes over time, so a static model may gradually lose effectiveness. Future systems can combine drift indicators, periodic human review, and controlled retraining. Drift alerts should not automatically trigger model replacement; a change may reflect legitimate business activity, a sensor update, or an attack. New models should pass the same regression and safety checks as initial versions.

D. Multi-source correlation

Combining network flows with endpoint, identity, DNS, and cloud audit events may help distinguish suspicious activity from normal but unusual traffic. Correlation can reveal a sequence of individually weak signals across multiple systems. Research should consider time synchronization, missing data, privacy, and the additional complexity of maintaining a common event schema.

XIII. CONCLUSION

An AI-based intrusion detection system can support network security by combining established signatures, supervised classification, anomaly detection, and analyst feedback. Its usefulness depends not only on the model but also on sound data collection, leakage-resistant validation, appropriate metrics, explainability, privacy safeguards, and continuous monitoring. The proposed architecture and workflow provide a reproducible starting point for implementation. Future work should evaluate the candidate models on a documented benchmark, test generalization across time or datasets, quantify false-alarm workload, and assess whether a hybrid approach offers practical benefits over simpler baselines.

REFERENCES

- [1] R. P. Lippmann et al., "The 1999 DARPA off-line intrusion detection evaluation," *Computer Networks*, vol. 34, no. 4, pp. 579–595, 2000.
- [2] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proc. IEEE Symposium on Computational Intelligence for Security and Defense Applications*, 2009, pp. 1–6.
- [3] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems," in *Proc. Military Communications and Information Systems Conference*, 2015, pp. 1–6.
- [4] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. International Conference on Information Systems Security and Privacy*, 2018, pp. 108–116.
- [5] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2016.
- [6] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Proc. IEEE Symposium on Security and Privacy*, 2010, pp. 305–316.
- [7] M. A. Ferrag et al., "Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study," *Journal of Information Security and Applications*, vol. 50, 2020.
- [8] National Institute of Standards and Technology, *Guide to Intrusion Detection and Prevention Systems (IDPS)*, Special Publication 800-94, 2007.