IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511190

AI Based Network Intrusion Detection System

Prajwal[1], Avinash[2], Sarvamangala[3], Kavita[4], Dr. V. B. Pagi[5]

Department of Computer Science and Engineering, Basaveshwar Engineering College, Bagalkote[1-4]

Project Guide, Department of Computer Science and Engineering, Basaveshwar Engineering College, Bagalkote[2-5]

Abstract: We built AI-NIDS, a prototype intrusion detection system that connects a trained classifier to live packet capture and a web-based security dashboard. The classifier is trained on the NSL-KDD benchmark. After cleaning and encoding the data, we kept 20 features and trained a Random Forest (RF) and a Decision Tree (DT). On the 22,544-record test set, the Random Forest scored 78.00% accuracy, 96.68% precision, 63.53% recall and a 76.68% F1-score; the Decision Tree scored 75.96%, 96.13%, 60.19% and 74.03%. The backend uses FastAPI with SQLAlchemy, the front end uses React and TypeScript, and they communicate over REST and WebSockets. For live capture, Scapy and Npcap (on Windows) sniff IPv4 and IPv6 packets from the chosen Wi-Fi or Ethernet interface, which are grouped into bidirectional five-tuple flows and converted into the model's feature vector. When the user switches interfaces, the old capture stops, a new one starts, and session counters and network details are refreshed. Simulated test-bench traffic is kept apart from real traffic. Results: Random Forest beat the Decision Tree on every metric, reaching 78.00% accuracy, but its recall was only 63.53%, so many attacks went undetected. A single host also sees only part of its network's traffic.

Keywords: Network Intrusion Detection, NSL-KDD, Random Forest, Decision Tree, Scapy, Npcap, FastAPI, WebSocket, Flow Aggregation, Security Operations Centre.

I. INTRODUCTION

As more of our work moves onto wired and wireless networks, spotting suspicious traffic early has become a basic security need. An intrusion detection system (IDS) watches network traffic and raises an alert when it sees patterns resembling known attacks. Supervised machine learning suits this, since a classifier learns from labelled connection records and judges new connections by their features.

We built a working prototype that combines an IDS with a small Security Operations Centre (SOC). It has a model trained on NSL-KDD, a backend API, a web interface, passive packet capture, flow-level inference, alerting and analytics. Results from real and simulated test-bench traffic are kept separate, so synthetic events are never mistaken for real network traffic.

Contributions: (1) a comparison of Random Forest and Decision Tree on the official NSL-KDD test set; (2) a model trained on 20 features; (3) a passive Scapy/Npcap capture module that aggregates bidirectional five-tuple flows; (4) a web SOC with alerting, role-based access, auditing and analytics; (5) session handling that keeps capture and dashboard in step across network changes.

II. RELATED WORK

The KDD Cup 1999 data was used for intrusion detection for years, but many of its records were duplicates, so a classifier could score well just by memorising the common ones [1]. NSL-KDD removed those duplicates, which is why we used it [2]. It is also the dataset most papers report on, so our results can be compared with theirs.

We tried two models. The Decision Tree is simple, and we could follow which features led to each prediction [4]. The Random Forest combines the votes of many trees, which usually makes it steadier than a single tree [3]. Comparing them shows what an ensemble gains over a single tree.

Others have tested similar models on NSL-KDD. Abualkibash [8] trained on the full KDDTrain+ file and tested on KDDTest+, getting 74% accuracy with a Decision Tree and 72% with a Random Forest. Sapre et al. [9] reported a Random Forest precision near 0.97 and recall near 0.62, the same weak recall we saw. Illy et al. [10] reached 85.81% accuracy on KDDTest+ with a bagged decision-tree ensemble, so combining models can beat a single one. Our Random Forest scored 78.00% accuracy, 96.68% precision and 63.53% recall, within the range of these results. Differences may come from feature selection, preprocessing, model parameters and settings.

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511190

We used Scapy with Npcap for packet capture on Windows [5], [6]. The backend is built on FastAPI, and WebSockets push live alerts to the dashboard [7]. AI-NIDS is only a prototype: it sees the traffic of the one machine it runs on and does not block attacks by itself.

III. SYSTEM ARCHITECTURE

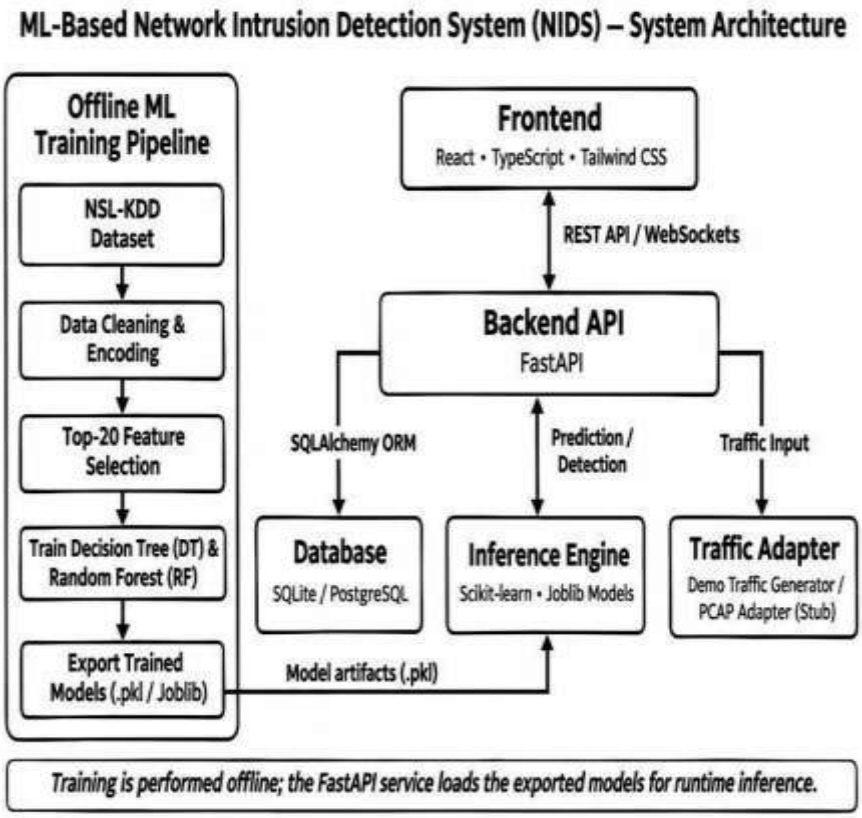


Fig. 1 System Architecture

IV. DATASET AND FEATURE PIPELINE

Models are trained offline and exported as Joblib files. When the FastAPI backend starts, it loads them to classify live flows. The backend sits in the middle and exchanges data with the front end, the database, the inference engine and the traffic adapter.

V. DATASET & FEATURES

NSL-KDD gives 125,973 training records and 22,544 test records. We chose 20 features: src_bytes, dst_bytes, flag, logged_in, same_srv_rate, dst_host_srv_count, diff_srv_rate, dst_host_same_srv_rate, serror_rate, srv_serror_rate, service, count, dst_host_same_src_port_rate, protocol_type, dst_host_srv_diff_host_rate, dst_host_srv_serror_rate, dst_host_diff_srv_rate, srv_count, dst_host_count and dst_host_rerror_rate.

We cleaned the data and label-encoded the categorical columns (protocol_type, service and flag). Model settings and any separate validation split are not documented here. Trained models were saved with Joblib and loaded by the inference module.

Mapping live traffic onto NSL-KDD features was the hardest part. Many features, such as count and serror_rate, describe a host's recent connections over a time window, not a single packet. We therefore group packets into bidirectional five-tuple flows and compute features over a configured time window. Features that cannot be read from raw packets, such as logged_in, are approximated during flow processing, which is one reason live results may differ from the benchmark.

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511190

VI. MODELS & RESULTS

Both models were evaluated on the official NSL-KDD test set of 22,544 records. Table I lists the results, which match the project README and the dashboard's ML Model page.

TABLE I PERFORMANCE COMPARISON OF CLASSIFIERS

Classifier	Accuracy	Precision	Recall	F1-Score
Random Forest	78.00%	96.68%	63.53%	76.68%
Decision Tree	75.96%	96.13%	60.19%	74.03%

Random Forest led on all four metrics, so it is our default model. Precision is high for both, around 96%, so an alert is almost always right. Recall is the weak spot at 60 to 64%, so over a third of attacks slip through: Random Forest missed 4,680 and Decision Tree missed 5,109. A missed attack costs more than a false alarm, so we read accuracy and F1 together with recall.

VII. REAL DATA CAPTURE & ANALYSIS

A. CAPTURE

Live capture is turned on from the settings page. Scapy and Npcap read IPv4 and IPv6 packets from the selected Wi-Fi or Ethernet interface; we aggregate bidirectional five-tuple traffic, extract the 20 features and classify them with the Random Forest or Decision Tree. The settings page reports Npcap driver status, capture permission, interface, SSID, gateway IP, host IP, and packet and flow totals. The live traffic page lists timestamps, interface names and source IP:port pairs, with search, filters and CSV export.

B. ANALYSIS

The analytics page shows total traffic, normal traffic, detected attacks, open alerts, detection rate and the model's offline benchmark accuracy. It also reports the main threat vector, most common protocol, average confidence over recent flows, intrusion trends, the normal-versus-attack split and feature importance charts.

A detection test bench accepts custom feature vectors or presets such as DoS SYN Flood, Port Sweep Probe and Normal HTTP. A WebSocket stream sends simulated flows and results to the SOC dashboard. These flows exist to test alerts and the interface; they are not evidence of malicious traffic captured from a real network.

VIII. ALERTING, ACCESS CONTROL & AUDITING

The SOC raises an alert whenever an attack is detected. Alerts carry a severity of CRITICAL, HIGH, MEDIUM or LOW and a status of OPEN, ACKNOWLEDGED or RESOLVED. The dashboard lists each intrusion with its threat type, source address, time, protocol, confidence and stream status, and a siren test page checks audio and voice alerts.

Login uses JSON Web Tokens (JWT) with role-based access: admins have full access and can resolve alerts, analysts can view traffic and acknowledge or resolve alerts, and viewers can only look. Audit logs record login attempts, manual predictions, acknowledged alerts and API usage. The demonstration build ships with default accounts for each role, which must be replaced before any use beyond a local demo.

IX. NETWORK CHANGES

When the host moves to a different network, AI-NIDS records the new active interface, default gateway IP and SSID (where one exists). It stops the running capture, starts a new one on the new interface, resets the session's dashboard counters and keeps earlier records in the database. The dashboard then shows the new SSID, gateway IP, host IP, interface name and a fresh session ID.

Short disconnections are ignored to avoid unnecessary resets. Interface and gateway changes are compared separately, networks without an SSID are handled, and start and stop calls are serialised with a lock to prevent overlapping capture operations. Capture errors are shown on the dashboard. During Wi-Fi and Ethernet switches, capture resumes once the new interface is ready; recovery time was not measured.

 This work is licensed under a Creative Commons Attribution 4.0 International License



International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026
DOI: 10.17148/IJARCCE.2026.1511190

X. LIMITATIONS & SECURITY

On Windows, capturing from a normal Wi-Fi or Ethernet interface shows only traffic to and from the host, plus broadcast and multicast. Unicast traffic between other devices on a switched network, or hidden inside encrypted Wi-Fi, is not visible unless the network offers port mirroring (SPAN) or an inline tap. AI-NIDS therefore works at host level.

NSL-KDD is a good research benchmark, but it is old, and a deployed system would need retraining on current traffic. Supervised models mostly recognise patterns seen in training, so novel zero-day attacks remain future work. Other limitations are moderate recall and the difficulty of rebuilding benchmark features from live flows. Inference time, CPU usage and long-run reliability have not been measured.

The system should only run on networks the user has permission to monitor. Automatic firewall blocking is off by default, and any blocking action needs an analyst's approval.

XI. CONCLUSION

We presented AI-NIDS, a prototype intrusion detection system with a FastAPI backend, a React and TypeScript SOC dashboard, NSL-KDD model training, passive Scapy/Npcap capture, flow aggregation, alerting, access control, auditing and network-aware sessions. On the 22,544-record test set, Random Forest reached 78.00% accuracy and a 76.68% F1-score, against 75.96% and 74.03% for the Decision Tree. Because recall is modest for both, our first priority is reducing missed attacks.

REFERENCES

[1]. 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.
[2]. Canadian Institute for Cybersecurity, “NSL-KDD dataset,” University of New Brunswick. [Online]. Available: https://www.unb.ca/cic/datasets/nsl.html
[3]. L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.
[4]. J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA, USA: Morgan Kaufmann, 1993.
[5]. Scapy Project, “Scapy documentation.” [Online]. Available: https://scapy.net/
[6]. Npcap, “Npcap User’s Guide.” [Online]. Available: https://npcap.com/guide/
[7]. FastAPI, “FastAPI documentation.” [Online]. Available: https://fastapi.tiangolo.com/
[8]. M. Abualkibash, “Machine learning in network security using KNIME analytics,” *Int. J. Netw. Secur. Appl. (IJNSA)*, vol. 11, no. 5, Sept. 2019, doi: 10.5121/ijnsa.2019.11501.
[9]. S. Sapre, P. Ahmadi, and K. Islam, “A robust comparison of the KDDCup99 and NSL-KDD IoT network intrusion detection datasets through various machine learning algorithms,” arXiv:1912.13204, 2019.
[10]. P. Illy, G. Kaddoum, C. M. Moreira, K. Kaur, and S. Garg, “Securing fog-to-things environment using intrusion detection system based on ensemble learning,” arXiv:1901.10933, 2019.

 This work is licensed under a Creative Commons Attribution 4.0 International License