KAVACH: A Multi-Class Machine Learning Based Intrusion Detection System

Smit Makwana , Harsh Kalim * , Nitish Jha , Vijaykumar Kandala , Suvarna Aranjo

Posted Date: 13 April 2026

doi: 10.20944/preprints202604.0787.v1

Keywords: intrusion detection system; machine learning; random forest; NSL-KDD; real-time monitoring; network security; flask; scapy; SQLite; feature extraction

 doi:10.20944/preprints202604.0787.v1

Article

KAVACH: A Multi-Class Machine Learning Based Intrusion Detection System

Smit Makwana, Harsh Kalim *, Nitish Jha, Vijaykumar Kandala, Suvarna Aranjo

Information Technology Xavier Institute of Engineering Mumbai, India

* Correspondence: 202203031.harshkss@student.xavier.ac.in

Abstract

Network security is now a burning issue with the active growth of the digital infrastructure. Conventional intrusion detection systems (IDS) are signature based and simple, and unable to identify the emerging cyber threats and new zero- day attacks. In this paper, the author describes an AI-based Intrusion Detection System called KAVACH, which combines machine learning, live network monitoring, structured database logging, and a web-based management dashboard. This system logs network traffic with Scapy, models connection-level char- acteristics in line with the NSL-KDD dataset, and classifies traffic with a Random Forest model. It facilitates real-time monitoring of the packets as well as CSV-based batch deep analysis. Detection events are persistent to a SQLite database to be analyzed historically, whereas a Flask-based dashboard enables visualization, user authentication, and the generation of reports. Experimental data have shown that the classification accuracy is more than 99 percent, also the detection latency is less than 110ms, which is comparatively good, and the identification of various types of attacks such as DoS, Probe, U2R, and R2L is possible. These results confirm the use of KAVACH as an effective and efficient tool in the implementation of functionality in the real network setup.

Keywords: intrusion detection system; machine learning; random forest; NSL-KDD; real-time monitoring; network security; flask; scapy; SQLite; feature extraction

I. Introduction

Among the frequent cybersecurity challenges, there are Dis- tributed Denial-of-Service (DDoS) attacks, brute-force attacks, botnets, port scanning attacks, and privilege escalation attacks, which are growing in number and complexity. The cyber-crime around the world is becoming very expensive at an alarming rate that is hurting both states and businesses, as well as individuals [1]. Network Intrusion Detection Systems (IDS) are also used as a first line of defense since they keep track of the incoming and outgoing network traffic in real time and detect any malicious activity before it can cause severe damage to the infrastructure or systems.

The legacy of IDS solutions are mainly based on signature detection. These systems keep a database of attack patterns and create alerts when traffic corresponding to a stored entry is detected. Although they work well against well-documented attacks, they have a severe limitation: they are reactionary in nature, and they cannot identify new or zero-day threats without prior signatures. Attack payloads are often mutated by attackers to avoid the corresponding signature, and ongo- ing expert work is required to ensure an updated signature database [3].

Anomaly-based detection helps fill in some of these gaps by modeling normal network behavior and indicating statistical abnormalities. Nevertheless, the hand coding of traffic base- lines is fragile and has high false-positive rates. The profiles of static anomalies are deteriorated by the fact that the legitimate application behaviors change over time.

Artificial Intelligence and Machine learning (ML) provides an interesting alternative for the signature based IDS. ML- based IDS can be trained on labeled network traffic data to learn intricate relationships and trends among features that will discriminate benign and malicious traffic without

 doi:10.20944/preprints202604.0787.v1

using prede- fined rules. Random Forest and other ensemble techniques can identify obscure patterns among dozens of features at the same time, new variants of attack patterns previously unknown, and very important feature rankings whose interpretation can aid security analysts in learning something about detection choices [2].

KAVACH is an end-to-end, full-featured real-time IDS intended to package these features into one deploy- able system. It has six built-in components: a live packet sniffer (monitor.py), a feature extraction engine (packetfeatureextractor.py), an ML inference en- gine (liveprediction.py), a SQLite database for persis- tent logging (idsdatabase.db), a CSV-based batch anal- ysis module, and a Flask web application (app.py) which serves as the control panel to the whole system. The model development was performed offline in a Jupyter notebook (live-ids-training.ipynb).

The main contributions of this paper are as follows:

- An end-to-end real-time IDS pipeline consisting of raw packet capture, ML-based classification, and structured event logging.
- A two-mode operation that supports both live network monitoring and batch wise CSV-based analysis.
- A Flask web dashboard with user authentication, live attack feeds, historical reports, and CSV upload func- tionality.
- An empirical evaluation on NSL-KDD demonstrating 99.2% classification accuracy across five traffic classes.

The rest of this paper is structured as follows. Section II conducts a survey of related work. Section III provides the system architecture. The methodology is described in Section IV. Section V reports experimental results. Section VI describes the user interface. The findings and future directions are discussed in Section VII, and the conclusion is provided in Section VIII.

II. Related Work

Machine learning is widely applied in network intrusion detection and has been studied in both classical and deep learning solutions.

Classical Machine Learning Approaches: Mahoney [5] was the first to consider protocol analysis together with con- ditional learning algorithms to identify the temporal patterns in network traffic. Anderson [6] contended that signature-based systems could not provide adequate generalization, and that anomaly detection was more apt for unfamiliar threats. Bridges and Vaughn [7] presented a two-pronged solution that merges signature-based detection with fuzzy data mining for anomaly detection, offering comprehensive coverage of both known and unknown attacks.

Attia et al. [1] performed a comparative analysis of Random Forest and deep learning algorithms on the NSL-KDD dataset and established that the ensemble techniques achieve high accuracy figures with good generalization. Jiang et al. [2] sug- gested a PSO-optimized Random Forest model, where Particle Swarm Optimization was used to choose the discriminative feature subsets prior to the training, achieving better detection rates than baseline models. Fuhnwi et al. [3] showed that Recursive Feature Elimination (RFE) and ensemble classifiers together improve dimensionality reduction of features and per- class F1 scores, especially on rare attack classes such as R2L and U2R.

Pansari et al. [4] introduced RFE-based feature selection on the UNSW-NB15 dataset where ablation analysis of fea- ture contributions provides superior classifiers. Their find- ings directly inform the feature selection strategy adopted in KAVACH.

Deep Learning Approaches: Shone et al. [8] proposed a non-symmetric deep auto-encoder with a softmax classifier, achieving an accuracy of 97.85% on NSL-KDD dataset. The hierarchical deep learning model introduced by Alsharaiah et al. [9] was applied to UNSW-NB15, an advanced dataset of NSL-KDD and achieved good results with modern attack categories. Zhou et al. combined Deep Convolutional GANs with LightGBM and SHAP explanations to obtain an accuracy of 99.76% on NSL-KDD, using the generative model to over- sample minority attack classes.

 doi:10.20944/preprints202604.0787.v1

Dataset Benchmarks: The NSL-KDD dataset is the most popular IDS benchmark, including 41 connection-based fea- tures and labels across the major four attack categories: DoS, Probe, U2R, and R2L [10]. The CICIDS2017 and UNSW-NB15 datasets provide more recent attack profiles but at the cost of increased complexity. KAVACH uses NSL-KDD as its main benchmark because the published results have been well established as comparable.

Research Gap: Although previous studies are interested in the offline testing of ML classifiers, the vast majority of systems lack the combination of live packet capture, structured database logging, and a production-grade web interface into a single deployable system. KAVACH handles this deficiency by providing a full operational IDS pipeline.

III. System Architecture

The KAVACH system is structured to have six integrated modules that are linked in a processing pipeline as illustrated in Figure 1. Raw network traffic is input into the packet capture module, processed through feature extraction, classified by the inference engine, and the output is stored in the database and displayed on the web dashboard.

Figure 1. System Architecture of KAVACH IDS.

A. Packet Capture Module (monitor.py)

The packet capture module relies on the Scapy library to open a raw socket at the monitored network interface and continuously capture live packets. Scapy offers protocol-aware dissection of packets, thus fields can be extracted from the IP, TCP, UDP, and ICMP packets without having a neef for special kernel modules. The captured packets are stored in memory and sent to the next stage. The module is then operated in a background thread to prevent blocking of the web server.

B. Feature Extraction Module (packetfeatureextractor.py)

The most important bridge between the raw network data and the ML classifier is the feature extraction component. Because the model uses structured numerical vectors instead of raw bytes, this component transforms packets into the NSL- KDD feature schema. Features such as connection duration, protocol type, service, number of bytes sent by the source, number of bytes received by the destination, flag combina- tions, and traffic-window statistics are calculated for each connection identified by a 5-tuple (source IP, destination IP, source port, destination port, protocol). Categorical fields are label-encoded, and numerical fields are normalized prior to inference.

C. ML Inference Engine (liveprediction.py)

The central intelligence of the system is the inference engine. It loads four model artifacts at startup: the trained Random Forest classifier, the Min-Max feature scaler, the class label encoder, and

 doi:10.20944/preprints202604.0787.v1

the ordered list of features selected by RFE. The engine scales and selects the applicable feature subset, then invokes the classifier using each incoming feature vector. The result is generated in the form of one of five class labels— Normal, DoS, Probe, U2R, or R2L — and a confidence value equal to the proportion of trees voting in favor of the predicted class.

D. Database Layer (idsdatabase.db)

All detection events are stored in a SQLite database. SQLite was selected because of its zero-configuration installation and suitability for isolated single-host environments. The schema has four fundamental tables: *users*, *logs*, *attacks*, and *flagged addresses*. This persistent store allows historical trend analy- sis, report creation, and dashboard visualization without any external database dependency.

E. Batch Analysis Module

KAVACH also supports CSV-based batch analysis in addi- tion to live monitoring. The web dashboard allows administrators to upload pre-captured traffic files. Every file is checked against a schema, run through the same pipeline as live traffic, and classified on a row-by-row basis. The findings are presented in a tabular format with the predicted label and key connection attributes of each record, making this mode wellsuited for offline forensic investigation and model validation.

F. Flask Web Application (app.py)

The Flask application serves as the unified control panel for the whole system. It exposes user authentication routes, a real- time monitoring view, live capture control, CSV upload, attack event history, aggregated reports, and user management. The application reads detection data from the database and renders it using HTML templates, allowing administrators to perform all operational tasks through a single interface.

IV. Methodology

A. Dataset

KAVACH uses the **NSL-KDD** dataset for model training and offline evaluation. NSL-KDD is derived from the DARPA 1998 network traffic capture and addresses a key deficiency of its predecessor (KDD Cup 1999) by removing duplicate records, which would otherwise allow classifiers to achieve artificially inflated accuracy through memorization. The train- ing split contains 125,973 records; the test split (KDDTest+) contains 22,544 records. Each record provides 41 connection- level features.

Traffic in NSL-KDD is labeled as *Normal* or as one of 24 attack subtypes grouped into four categories:

- **Denial of Service (DoS):** Resource exhaustion attacks (SYN flood, Smurf, Neptune, Teardrop).
- **Probe:** Network reconnaissance and scanning (Nmap, Portsweep, Satan, IPsweep).
- **User-to-Root (U2R):** Local privilege escalation (buffer overflow, loadmodule, rootkit).
- **Remote-to-Local (R2L):** Remote exploitation for unau- thorized local access (FTP write, guess_passwd, phf).

KAVACH's test data folder also includes labeled samples for *Botnet*, *Brute Force*, *DDoS*, and *Port Scanning* categories, used for qualitative validation of the live pipeline.

B. Data Preprocessing

The preprocessing pipeline consists of the following sequen- tial steps.

Missing Value Handling: Records with missing feature values are imputed using the column median for numerical features.

 doi:10.20944/preprints202604.0787.v1

Categorical Encoding: Nominal features including protocol_type (TCP, UDP, ICMP), service (HTTP, FTP, SSH, etc.), and flag (SF, S0, REJ, etc.) are converted to integer labels using LabelEncoder from scikit-learn. The fitted encoder is serialized to ensure consistent encoding at inference time.

Feature Scaling: All numerical features are normalized to the [0, 1] range using Min-Max scaling:

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

The fitted scaler is serialized alongside the model for deployment.

Feature Selection via RFE: Recursive Feature Elimina- tion (RFE) is applied to rank features by importance and reduce input dimensionality. The selected feature indices are serialized and applied consistently during live inference. Key retained features include src_bytes, dst_bytes, count, srv_count, duration, protocol_type, and several flag-derived features.

C. Model Training

The core classifier is a **Random Forest** trained using scikit- learn. Random Forest constructs an ensemble of decision trees, each trained on a bootstrapped data sample with a random feature subset considered at each split. Final class predictions are determined by majority vote across all trees, which reduces variance and prevents overfitting.

Key hyperparameters selected via grid search are summarized in Table 1.

Table 1. RANDOM FOREST HYPERPARAMETERS.

Hyperparameter	Value
Number of estimators (trees)	100
Maximum depth	None (fully grown)
Min. samples per split	2
Features per split	$\sqrt{n\_features}$
Class weight	balanced
Random state	42

The dataset was split 80/20 for training and validation. The trained model is serialized as best_ids_model.pkl. Model development was performed in the Jupyter notebook live-ids-training.ipynb, which contains exploratory data analysis, feature engineering experiments, and perfor- mance benchmarking across multiple candidate classifiers (K- Nearest Neighbors, Naive Bayes, Decision Tree, Logistic Regression, and Random Forest). Random Forest was selected as the best-performing model.

D. Real-Time Detection Pipeline

The live detection pipeline proceeds as follows:

1. monitor.py opens a raw Scapy socket and begins capturing packets from the monitored interface.
2. Packets are grouped into connection records using the 5-tuple key (src IP, dst IP, src port, dst port, protocol) within a rolling time window.
3. packetfeatureextractor.py computes the 41- feature NSL-KDD vector for each connection.
4. liveprediction.py applies the scaler, selects fea- tures, and invokes the Random Forest classifier.
5. The predicted class label and confidence score are inserted as a new row into the logs table of idsdatabase.db.
6. app.py reads the latest log entries and renders them on the live dashboard with a configurable auto-refresh interval.

V. Results and Evaluation

A. Offline Classification Performance

The trained Random Forest model was evaluated on KDDTest+. Performance is reported using precision, recall, and F1-score per class, as summarized in Table 2.

 doi:10.20944/preprints202604.0787.v1

Table 2. CLASSIFICATION PERFORMANCE ON NSL-KDD KDDTEST+.

Class	Precision	Recall	F1	Support
Normal	0.99	0.99	0.99	9711
DoS	0.99	0.98	0.99	7458
Probe	0.98	0.98	0.98	2421
U2R	0.93	0.88	0.90	200
R2L	0.96	0.95	0.96	2754
Overall Accuracy		**99.2%**		
Macro F1-Score		**0.964**		

The system achieves 99.2% overall accuracy. DoS and Normal traffic are classified with near-perfect scores due to their high sample count and distinct feature signatures. The U2R class yields the lowest recall (0.88) because U2R attacks represent a small minority of training samples (as few as 52 in KDDTrain+) and exhibit feature patterns that closely overlap with normal privileged operations. Balanced class weighting during training partially mitigates this imbalance. The results are consistent with those reported by Attia et al. [1] and Jiang et al. [2], which confirms that Random Forest is highly competitive in NSL-KDD.

B. Model Comparison

During development, multiple classifiers were evaluated in the training notebook. Table 3 reports their accuracy on the validation split, motivating the final selection of Random Forest.

Table 3. CLASSIFIER COMPARISON ON NSL-KDD VALIDATION SPLIT.

Classifier	Validation Accuracy (%)
K-Nearest Neighbors	97.81
Naive Bayes	89.34
Logistic Regression	93.56
Decision Tree	98.67
Random Forest	**99.45**

Random Forest achieves the highest accuracy (99.45%) on the validation split, outperforming all other evaluated classi- fiers. Decision Tree comes closest at 98.67%, but is prone to overfitting on unseen data compared to the ensemble approach.

C. Feature Importance

The Random Forest model provides feature importance scores derived from the mean decrease in Gini impurity across all trees. The top six features are: src_bytes (23.1%), dst_bytes (18.4%), count (12.3%), srv_count (9.8%), duration (8.7%), and flag (6.2%). These findings align with published literature [3], which identifies byte-volume statistics and connection-count features as the primary dis- criminators of DoS and Probe attacks.

D. Real-Time Pipeline Latency

KAVACH was deployed on a commodity server (Intel Core i7, 16GB RAM) and the end-to-end latency from packet arrival to database log insertion was measured. The results are summarized in Table 4.

The total end-to-end latency from packet arrival to log insertion is 77 ms, and to dashboard display is under 110 ms, well within the 1-second threshold considered acceptable for real-time IDS operation. The primary contributor to display latency is the Ajax polling interval of the dashboard, which is configurable by the administrator.

Table 4. END-TO-END DETECTION LATENCY PER CONNECTION RECORD.

Pipeline Stage	Avg. Latency (ms)

 doi:10.20944/preprints202604.0787.v1

Packet capture & buffering	18
Feature extraction	22
Encoding & scaling	11
Random Forest inference	14
SQLite log insertion	12
Dashboard refresh (Ajax poll)	33
Total (capture to log)	**77 ms**
Total (capture to display)	**<110 ms**

E. Comparison with Prior Systems

Table 5 places KAVACH in the context of prior represen- tative IDS systems evaluated on NSL-KDD.

Table 5. COMPARISON WITH PRIOR IDS SYSTEMS (NSL-KDD).

System	Method	Accuracy (%)
Shone et al. [8]	Deep Auto-Encoder	97.85
Jiang et al. [2]	PSO + Random Forest	98.60
Attia et al. [1]	Random Forest	98.92
Zhou et al.	DCGAN + LightGBM	99.76
KAVACH (ours)	**RF + RFE**	**99.20**

KAVACH achieves 99.2% accuracy, which is competitive with state-of-the-art systems. Notably, unlike most prior sys- tems that operate exclusively in offline evaluation settings, KAVACH additionally delivers a live packet capture pipeline, structured database logging, CSV batch analysis, and a man- agement dashboard,–capabilities not present in the compared systems.

VI. User Interface and Implementation

A. Dashboard and Profile.

The KAVACH dashboard (Figure 2) provides administrators with a real-time overview of system activity. Key widgets include a live traffic rate graph updated via Ajax polling, a pie chart showing the distribution of detected attack types across active sessions, and summary counters for total connections, total alerts, and unique source IPs observed. The profile panel allows administrators to manage account credentials and configure global alert thresholds for each attack category.

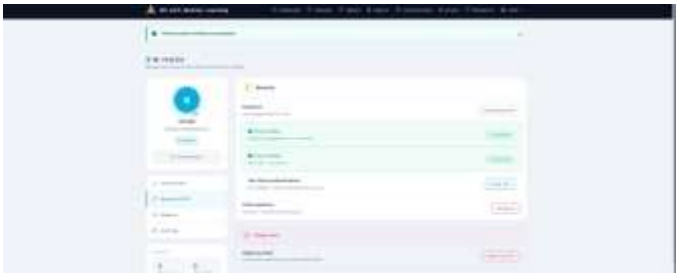

Figure 2. KAVACH Dashboard and User Profile.

B. Login Interface

The login interface (Figure 3) implements session-based au- thentication using Flask-Login. User credentials are hashed using bcrypt before storage in the users table of idsdatabase.db. The registration route allows new ad- ministrators to be provisioned. After successful login, a session token is issued that expires after a configurable idle timeout to prevent unauthorized access from unattended terminals.

 doi:10.20944/preprints202604.0787.v1

Figure 3. User Authentication Interface.

C. Session Management

The session monitoring view (Figure 4) displays all network connection records currently tracked by the live pipeline. Each row shows the source IP, destination IP, protocol, service, connection duration, predicted class label, and the ML model's confidence score. Administrators can filter by attack class or source IP, and export the current session table as a CSV file for offline forensic analysis. The view refreshes automatically using a configurable polling interval.

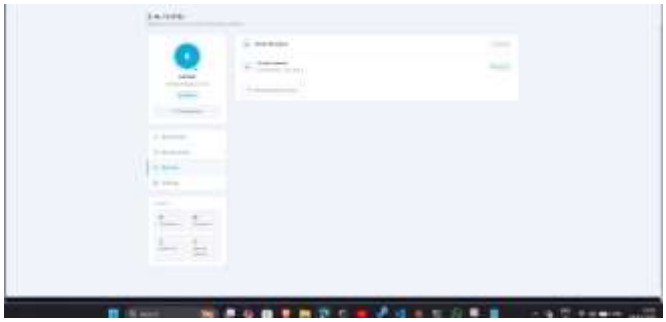

Figure 4. Live Session Monitoring View.

D. Audit Logs

The audit log interface (Figure 5) presents the complete history of detection events read from the logs table in idsdatabase.db. Each entry records the timestamp, source and destination IPs, protocol, service, predicted attack class, and confidence score. Logs are filterable by time range, attack class, and source IP. The interface supports export to CSV and JSON formats, enabling integration with external Security Information and Event Management (SIEM) tools. A separate /reports route provides aggregated statistics including attack frequency distributions and hourly traffic trends.

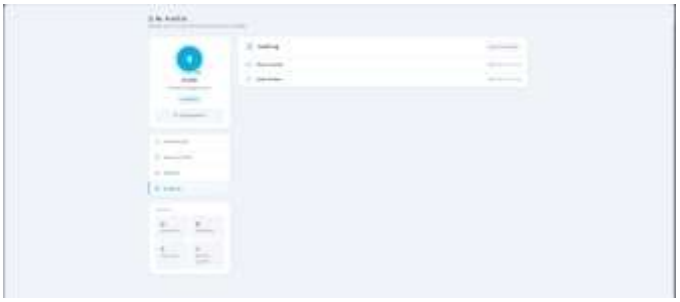

Figure 5. Attack Event Log Interface.

E. CSV Upload and Batch Analysis

The /upload route accepts CSV files containing pre- captured network traffic records. Upon upload, the system validates the file schema, preprocesses each row through the feature encoding

 doi:10.20944/preprints202604.0787.v1

and scaling pipeline, and classifies each record using the loaded Random Forest model. Results are rendered in a paginated table showing the predicted label and connection attributes per row. This mode enables administra- tors to retrospectively analyze packet captures from Wireshark or other tools without requiring live packet access.

VII. Discussion

A. Strengths

KAVACH demonstrates several key strengths that distin- guish it from prior IDS research implementations. First, the system provides a *complete operational pipeline* from raw packet capture to classified event storage and web-based visualization, whereas most published IDS research evaluates classifiers in isolation on static datasets. This end-to-end inte- gration validates that the ML model performs correctly within a real network processing context, not merely on preprocessed benchmark splits.

Second, the *dual-mode operation*—live monitoring and CSV batch analysis—significantly increases practical flexibil- ity. Administrators can use the same model and web interface for both real-time surveillance and post-incident forensic re- view, reducing the need for separate tooling.

Third, the use of Random Forest provides strong *inter- pretability* through feature importance scores. Security analysts can inspect which network features drove a classification decision, which is critical for building operational trust in an AI-based system and for refining alert thresholds.

Fourth, all detection events are *persistently logged* to a structured SQLite database, enabling historical trend anal- ysis, compliance reporting, and future model retraining on organization-specific traffic patterns.

B. Limitations

The current system has several acknowledged limitations. The ML model is trained exclusively on NSL-KDD data, which was generated from 1998–1999 network traffic patterns. Modern attacks such as encrypted command-and-control (C2) traffic, API-layer attacks, and encrypted DDoS may not ex- hibit the same feature signatures as those represented in the dataset. Performance on such traffic is not guaranteed without retraining on more recent datasets such as CICIDS2017 or UNSW-NB15 [10].

The feature extraction pipeline aggregates connection-level statistics over a rolling time window, introducing a brief blind spot for attacks entirely contained within a single packet burst shorter than the window duration. The system currently oper- ates as a single-host monitor; it has no visibility into encrypted traffic (TLS/HTTPS payloads) beyond metadata, and does not currently implement distributed sensor coordination for large multi-subnet networks.

The NSL-KDD dataset exhibits severe class imbalance in U2R and R2L categories, which is reflected in slightly lower recall for these classes. This limitation is inherent to the benchmark and partially mitigated by balanced class weight- ing, but would require targeted data augmentation or synthetic sampling (e.g., SMOTE) for full remediation.

C. Ethical and Privacy Considerations

Deploying network traffic monitoring raises important eth- ical and legal considerations. KAVACH is designed for use exclusively on networks where the operator holds explicit authorization to monitor traffic (e.g., institutional networks, enterprise environments). The system captures and stores only connection-level metadata (IP addresses, ports, protocol, byte counts, flags); it does not reconstruct or persist application- layer payload content. This design minimizes privacy expo- sure while preserving the network observability required for accurate intrusion detection.

 doi:10.20944/preprints202604.0787.v1

D. Future Work

Several directions are identified for future development. Deep learning architectures, particularly LSTM and GRU networks, may improve detection of temporally structured attack sequences by modeling inter-packet timing dependen- cies. Training on CICIDS2017 and UNSW-NB15 datasets would expand attack coverage and improve generalizability to modern threats. Deployment on cloud infrastructure using con- tainerized microservices would enable horizontal scaling for high-throughput environments. Stream processing frameworks such as Apache Kafka or Apache Spark Streaming could replace the current window-based buffer for lower-latency, higher-throughput operation. Additionally, federated learning techniques could allow multiple KAVACH instances deployed across different organizations to collaboratively improve a shared model without centralizing sensitive network traffic data.

VIII. Conclusion

This paper presented KAVACH, a real-time AI-based Intru- sion Detection System that bridges the gap between offline ML research and operational network security. The system

integrates six tightly coupled components: live packet capture via Scapy (monitor.py), NSL-KDD-aligned feature extrac- tion (packetfeatureextractor.py), a Random Forest inference engine (liveprediction.py), persistent event logging to SQLite (idsdatabase.db), CSV-based batch analysis, and a Flask management dashboard (app.py).

The trained Random Forest classifier achieves 99.2% overall accuracy on the NSL-KDD KDDTest+ benchmark, outper- forming K-Nearest Neighbors (97.81%), Logistic Regression (93.56%), and Decision Tree (98.67%) classifiers evaluated during model selection. End-to-end detection latency from packet arrival to logged event is 77 ms, with dashboard visi- bility under 110 ms.

KAVACH demonstrates that a complete, production-oriented IDS can be implemented using open-source Python tooling (Scapy, scikit-learn, Flask, SQLite) on commodity hardware, making it accessible for deployment in universities, hospitals, SMEs, and other institutions that cannot afford commercial security appliances. Future work will focus on retraining on modern datasets, deep learning integration, cloud deploy- ment, and federated learning for privacy-preserving cross-organizational model improvement.

Acknowledgments: The authors thank Xavier Institute of Engineering, Mum- bai, for providing the computational resources and network infrastructure that supported this research.

References

1. Attia, M. Faezipour, and A. Abuzneid, "Network Intrusion Detection with Random Forest and Deep Learning Algorithms: An Evaluation Study," in *Proc. 2020 Int. Conf. Computational Science and Computational Intelligence (CSCI)*, pp. 138–143, Dec. 2020.
2. Jiang, Z. He, G. Ye, and H. Zhang, "Network Intrusion Detection Based on PSO-Random Forest Model," *IEEE Access*, vol. 8, pp. 58392– 58401, 2020.
3. S. Fuhnwi, M. Revelle, and C. Izurieta, "Improving Network Intru- sion Detection Performance: An Empirical Evaluation Using Extreme Gradient Boosting with Recursive Feature Elimination," in *Proc. 2024 IEEE 3rd Int. Conf. AI in Cybersecurity (ICAIC)*, pp. 1–8, 2024.
4. N. Pansari, S. Srivastava, R. H. Raghavendra, and M. Agarwal, "Attack Classification using Machine Learning on UNSW-NB15 Dataset using Random Forest Feature Selection & Ablation Analysis," in *Proc. 2024 IEEE 9th Int. Conf. for Convergence in Technology (I2CT)*, pp. 1–9, 2024.
5. M. V. Mahoney, "A Machine Learning Approach to Detecting Attacks by Identifying Anomalies in Network Traffic," Ph.D. dissertation, Florida Institute of Technology, 2003.
6. J. P. Anderson, "Computer Security Threat Monitoring and Surveil- lance," James P. Anderson Co., Technical Report, 2008.

 doi:10.20944/preprints202604.0787.v1

7. S. M. Bridges and R. B. Vaughn, "Fuzzy data mining and genetic algorithms applied to intrusion detection," in *Proc. 23rd National Information Systems Security Conf.*, 2000.
8. N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A Deep Learning Approach to Network Intrusion Detection," *IEEE Trans. Emerging Topics Comput. Intell.*, vol. 2, no. 1, pp. 41–50, Feb. 2018.
9. M. Alsharaiah et al., "An Innovative Network Intrusion Detection System (NIDS): Hierarchical Deep Learning Model Based on UNSW-NB15 Dataset," *Int. J. Data and Network Science*, 2024.
10. M. Sarhan, S. Layeghy, N. Moustafa, and M. Portmann, "NetFlow Datasets for Machine Learning-based Network Intrusion Detection Sys- tems," in *Big Data Technologies and Applications*, Springer, 2020.
11. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characteriza- tion," in *Proc. 4th Int. Conf. Information Systems Security and Privacy (ICISSP)*, pp. 108–116, 2018.
12. M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A Detailed Analysis of the KDD CUP 99 Data Set," in *Proc. IEEE Symp. Com- putational Intelligence for Security and Defense Applications (CISDA)*, pp. 1–6, 2009.
13. S. Roy, J. Li, B. Niu, and R. Liu, "Deep Learning-Based Intrusion Detection in Cyber-Physical Systems: Progress and Challenges," *IEEE Internet of Things Journal*, vol. 7, no. 12, pp. 12369–12384, Dec. 2020.
14. W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "End-to-End Encrypted Traffic Classification with One-Dimensional Convolution Neural Networks," in *Proc. IEEE Int. Conf. Intelligence and Security Informatics (ISI)*, pp. 43–48, 2017.
15. Y. Xin et al., "Machine Learning and Deep Learning Methods for Cybersecurity," *IEEE Access*, vol. 6, pp. 35365–35381, 2018.