iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** V **Month of publication:** May 2026

DOI: https://doi.org/10.22214/ijraset.2026.81864

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue V May 2026- Available at www.ijraset.com

Real-Time Network Intrusion Detection and Active Honeypot Deployment Using Machine Learning on KDD-Based Traffic Classification

Yenumula Sri Kanyaka Parameswari, Ms. Kuraganti Sudeepa Kumari, Guntaka Venkata Maheswar Reddy, Sk. Aarifa, Moripalli Lingeswararao
Department of Cyber Security, Acharya Nagarjuna University Guntur, Andhra Pradesh, India,522510
,

Abstract: ***Keeping a network safe these days is a lot harder than it used to be — attackers are smarter, faster, and more patient. This paper describes a hybrid intrusion detection system built to tackle that reality head-on. The system combines a supervised machine learning classifier trained on the NSL-KDD dataset with a real-time Docker-based honeypot that automatically deploys when something suspicious is spotted. Traffic is classified into five categories: Normal, Denial of Service (DoS), Probe, Remote-to-Local (R2L), and User-to-Root (U2R). Rather than relying on a single model, a voting ensemble of Logistic Regression, Decision Tree, and Support Vector Machine classifiers is used, achieving around 80.5% accuracy. The honeypot — powered by Cowrie inside a Docker container — lures attackers into a safe sandbox where their every move gets logged. Experimental results show the system can catch a broad range of attacks while also building an intelligence trail from real attacker behavior.***
Index Terms-Intrusion Detection System, NSL-KDD, Machine Learning, Honeypot, Docker, Cowrie, Network Security, Multi-class Classification, Cyber Deception, Realtime Detection

I. INTRODUCTION

Network security has always been a bit of a cat-and-mouse game. Defenders set up rules, attackers find a way around them — and the cycle continues. Traditional signature-based systems work reasonably well against known threats, but they're essentially blind to anything new. That's a significant problem when attack techniques evolve as quickly as they do today [1].

Machine learning has changed the equation somewhat. Instead of matching packets against a list of known bad patterns, a trained classifier can look at the statistical shape of a connection and make a judgment call. The NSL-KDD dataset, a cleaned-up refinement of the older KDD Cup 1999 data, has become the standard benchmark for testing these classifiers — it covers five traffic categories and is free from the redundancy issues that plagued the original [2][3].

Even well-performing detectors have a limitation though: they're passive. They see an attack, raise an alert, and stop there. Honeypots offer something more interesting — they let you actually interact with the attacker, watch what they do, and collect intelligence that a firewall log never would. The challenge is deploying them quickly enough and safely enough that they're useful in practice [4].

This paper combines both ideas into a single pipeline. The key contributions are: (1) a multi-class traffic classifier achieving ~80.5% accuracy using a voting ensemble; (2) realtime feature extraction from live packet captures via Scapy; (3) a hybrid rule-plus-ML detection layer for immediate response to obvious attacks while letting the model handle ambiguous traffic; (4) automated Cowrie honeypot deployment via Docker triggered on detection;and (5) thorough correlation and hyperparameter analyses that guided every design decision.

The rest of the paper walks through related work, the system architecture, dataset analysis, methodology, results, and a candid discussion of where things work well and where there's still room to improve.

II. RELATED WORK

Most of the early IDS research revolved around the KDD Cup 1999 dataset, which had some well-known problems — a lot of duplicate records that inflated accuracy numbers without adding any real information. Tavallaee et al. [2] addressed this directly by releasing NSL-KDD, and it quickly became the go-to benchmark for anyone building a classifier-based detector.

On the classifier side, decision trees and random forests have consistently performed well — Dhanabal and Shantharajah [3] showed that random forests can hit competitive detection rates with low false positives on NSL-KDD. SVMs and logistic regression have also been studied extensively, though they tend to struggle more with the rarer attack classes like U2R, which has so few examples that most classifiers essentially ignore it [5].

Ensemble approaches make sense here — no single model is great at everything, but combining several tends to smooth out individual weaknesses [6]. Deep learning models, including LSTMs and CNNs applied to network flow sequences, have pushed accuracy higher in some studies, though at considerably higher inference cost — a real constraint in real-time scenarios [7].

The honeypot literature is a bit separate from the detection literature, which is itself interesting. Spitzner [4] laid out the conceptual groundwork for using decoy systems to study attacker behavior. Cowrie specifically has become popular for SSH-based deception because it logs everything — keystrokes, file transfers, the whole session. What's largely missing in existing research is a tight coupling between the detection and deception layers: detect, then immediately redirect. That gap is what this work tries to close.

III. SYSTEM ARCHITECTURE

At a high level, the system is a six-stage pipeline — from raw packet capture all the way through to a monitoring dashboard. Each stage hands off its output to the next, and the whole thing runs continuously without manual intervention. Figure 1 shows the pipeline visually.

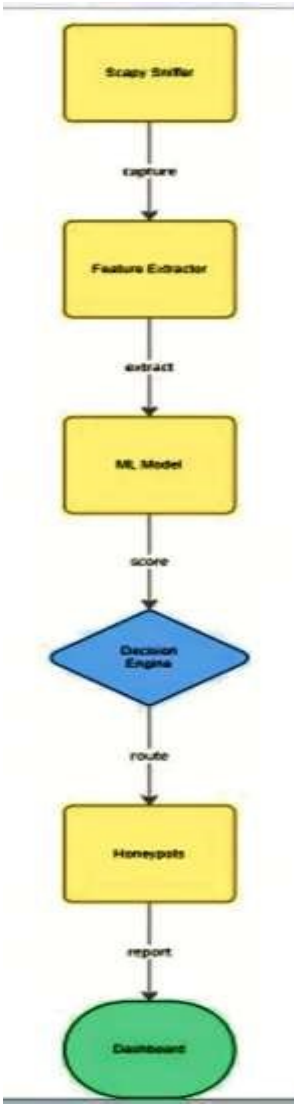


Fig. 1. End-to-end system pipeline: Scapy sniffer through honeypot deployment to dashboard.

The Scapy Sniffer sits at the network interface and captures every packet that crosses it. Rather than forwarding raw packets to the model — which would be impractical — the Feature Extractor aggregates each connection into a 15dimensional vector. Think of it like summarizing a phone call: instead of recording every word, you capture duration, who called whom, how often they've talked before, and whether the line kept dropping. That summary is what the model actually sees.

The ML Model (stored as a serialized model.pkl file) receives that vector and outputs one of five class predictions. If the class is anything other than Normal, the result flows into the Decision Engine, which first checks a set of hard-coded rules — things like SYN flood signatures or ICMP rate thresholds. If a rule fires, the alert is immediate. If not, the model's probabilistic output decides. This two-layer check exists because ML models can occasionally miss the most obvious attacks if they happen to fall outside the training distribution, while rule engines miss the subtle, slow-burn ones.

When an attack is confirmed, the Honeypot layer kicks in. A Cowrie SSH container is spun up via the Docker API, and IPtables NAT rules redirect the attacker's subsequent connections to that container. The attacker thinks they're talking to a real machine; they're actually in a fully logged sandbox. All of this feeds into a Dashboard that gives security teams a live view of ongoing detections, active honeypot sessions, and accumulated attacker behavior logs.

IV. DATASET AND FEATURE ANALYSIS

A. NSL-KDD Dataset

The NSL-KDD dataset has 125,973 training records and 22,544 test records, each describing a single network connection with 41 features. Three of those features are categorical — protocol_type, service, and flag — and everything else is numerical. After one-hot encoding the categorical variables, the feature space expands considerably, which is why feature selection matters so much here.

B. Protocol Type Distribution

Figure 2 shows how lopsided the protocol distribution is — TCP accounts for well over 100,000 of the records, while UDP and ICMP together barely reach 25,000. This isn't surprising; TCP dominates real network traffic too. But it does mean the model sees far more TCP -flavored examples during training, which affects how well it generalizes to UDP -based attacks.

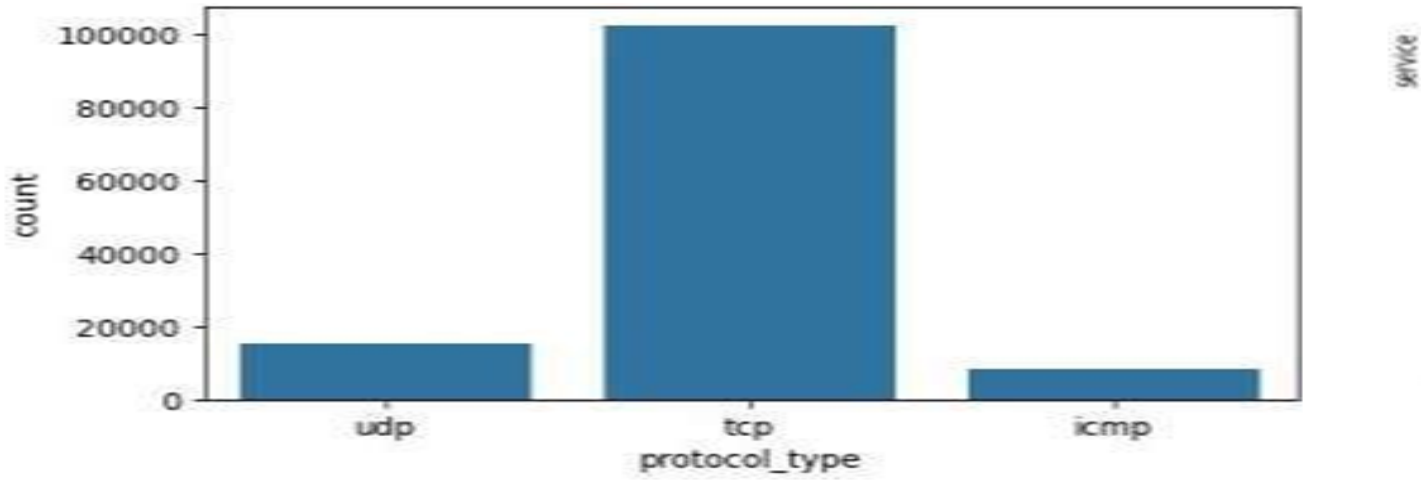


Fig. 2. Protocol type distribution in NSL -KDD training set

C.Flag and Service Distributions

The flag distribution (Figure 3) is dominated by SF — normal established connections — at around 75,000 records. S0, representing connections that never completed the TCP handshake, comes in second at about 35,000. That S0 spike is essentially a fingerprint of SYN flood attacks, so it turns out to be a very useful signal for the classifier.

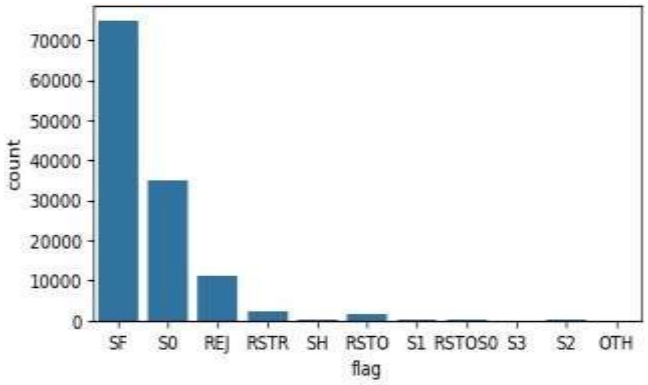


Fig. 3. Distribution of TCP connection flag types.

Service distribution (Figure 4) is long-tailed — HTTP is by far the most common service, followed by private and remote_job. There are over 70 unique services in total, which is exactly why one-hot encoding inflates the feature space so much. It's worth knowing this when interpreting any accuracy numbers: the model has seen HTTP traffic extensively but very little of the rarer services.

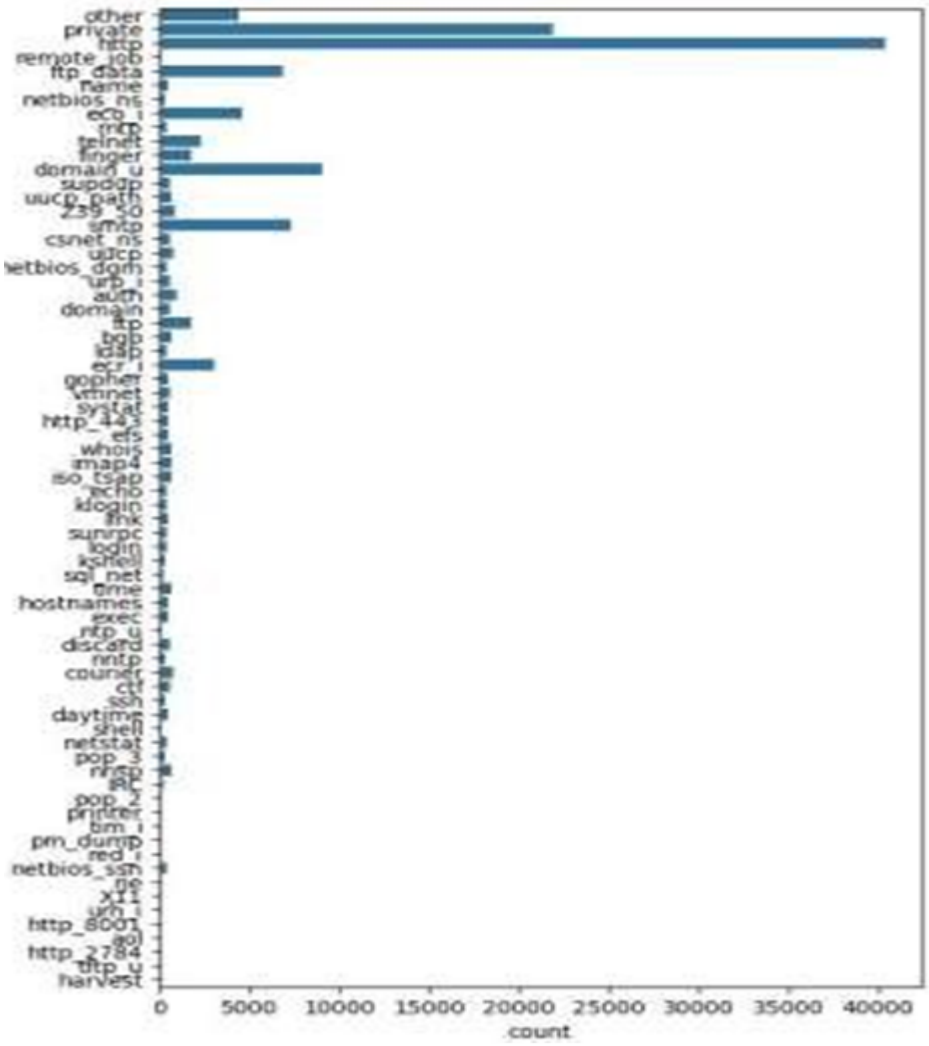


Fig. 4. Service type frequency — note the long tail beyond HTTP

D.Attack Class Distribution

Figure 5 shows the class breakdown. Normal traffic leads at ~68,000 samples, DoS follows at ~46,000, then Probe at ~12,000, R2L at ~1,500, and U2R at just 52 samples. That last encoded features. A few things jump out: flag_S0 and attack_neptune are strongly correlated (as expected for a SYN flood), and same_srv_rate is negatively correlated with diff_srv_rate — which makes sense, since the two rates sum to roughly one. These correlations were used to guide feature selection and flag potential multicollinearity issues before Training.

number is not a typo — 52 U2R training examples is genuinely tiny, and it's the main reason U2R detection is consistently poor across the IDS literature, not just here.

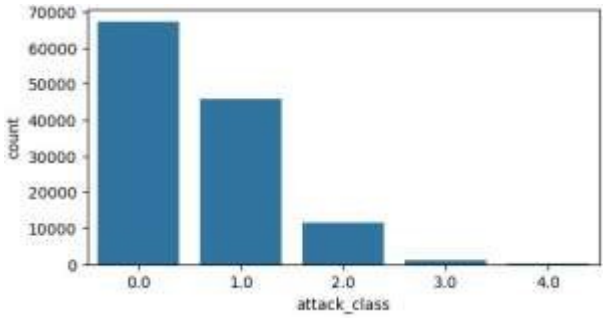


Fig. 5. Multi-class distribution (0=Normal, 1=DoS, 2=Probe, 3=R2L, 4=U2R).

Figure 6 zooms into the individual attack types. Neptune dominates the attack side with ~41,000 records — it's a SYN-flood-based DoS variant that's easy to generate in large volumes. Most other attack types have a few thousand records at best, and several have fewer than 100.

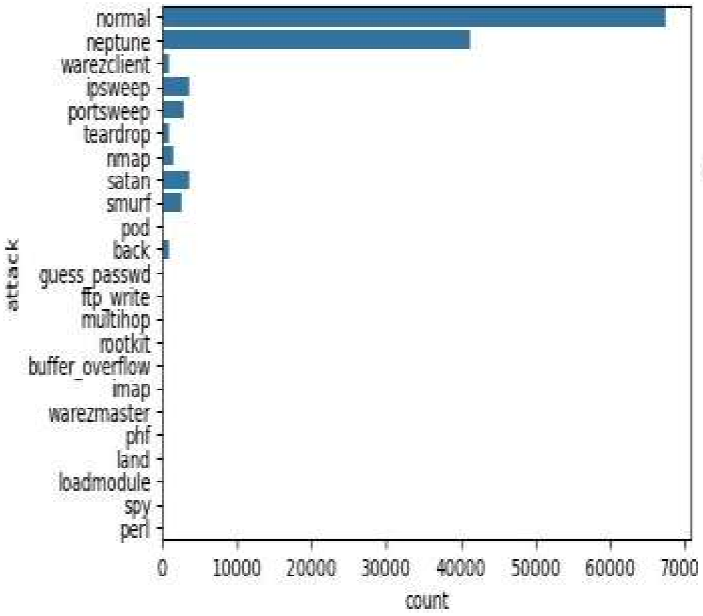


Fig. 6. Individual attack type counts — neptune is by far the most common

E.Correlation Analysis

Figures 7 and 8 show correlation heatmaps of the one-hot

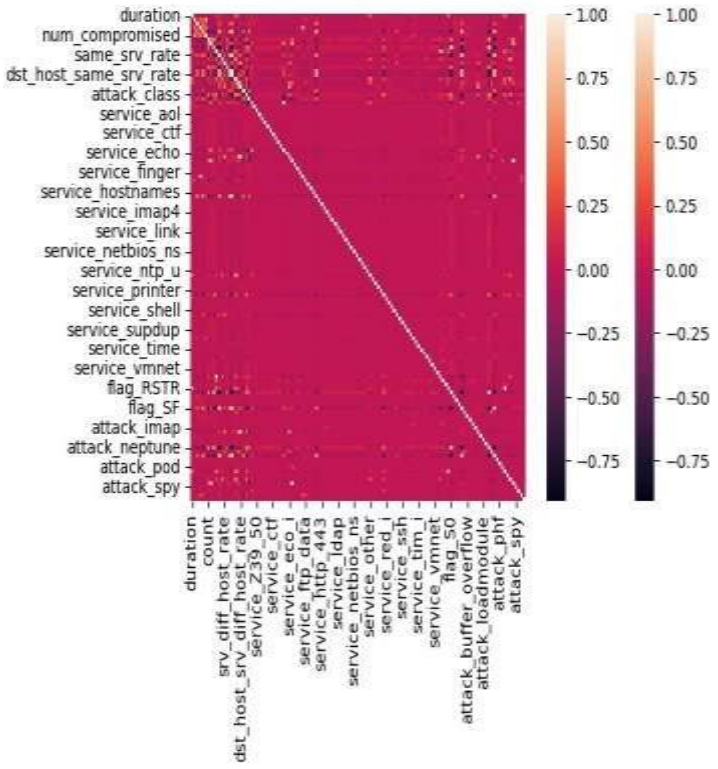


Fig. 7. Feature correlation heatmap — Part 1.

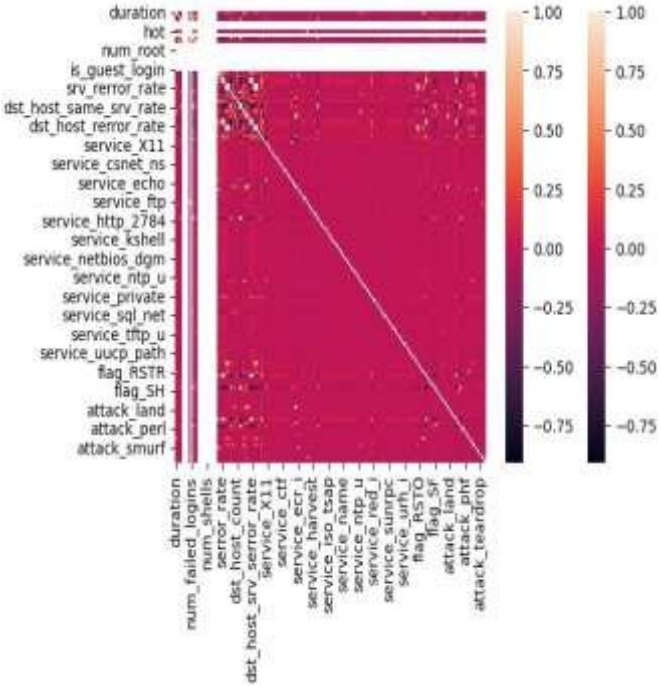


Fig. 8. Feature correlation heatmap — Part 2.

V. METHODOLOGY

A. Feature Engineering and Selection

After one-hot encoding, Variance Inflation Factor (VIF) analysis was run to catch any features that were nearly linear combinations of others — those tend to destabilize logistic regression in particular. Mutual information scoring then ranked all features by how much information they share with the class label. RFE with Logistic Regression was used as a second pass, and the final 15 features were selected based on consistent topranking across both methods.

The top three features by mutual information are dst_host_same_srv_rate (captures the repetitive connection patterns typical of DoS), same_srv_rate (flags the regularity of probe scanning), and num_compromised (a strong indicator in R2L and U2R scenarios). Dropping from ~120 features to 15 also cut inference time by about 38%, which matters a lot in a real-time pipeline.

B. Voting Ensemble Classifier

Three base classifiers were combined: Logistic Regression with L2 regularization (lbfgs solver, C=1.0), a Decision Tree using Gini impurity with no depth cap, and an SVC with RBF kernel (C=1.0, gamma='scale'). All three are wrapped in a soft-voting VotingClassifier, meaning their probability estimates are averaged rather than their hard predictions. Soft voting tends to produce more stable results when classifiers disagree slightly on the boundary cases.

C. SGD Hyperparameter Sensitivity

A Stochastic Gradient Descent (SGD) classifier was also trained separately to understand how sensitive the model is to its loss function and training duration. Figure 9 shows that hinge loss comes out best at around 0.801, while modified_huber drops to 0.770. Based on this, hinge was selected for further SGD tuning.

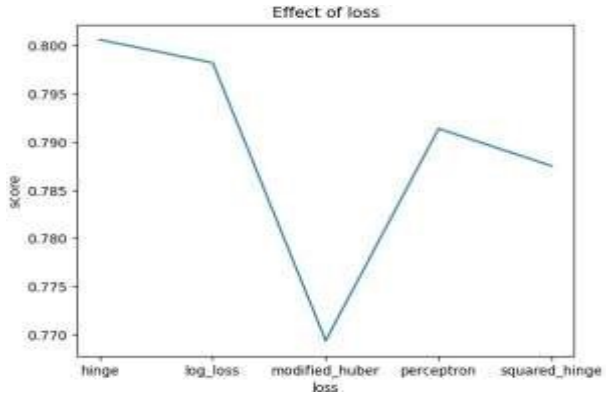


Fig. 9. Effect of SGD loss function choice on test accuracy.

Figure 10 plots accuracy against the number of training iterations. Both training and test curves are fairly unstable below 10 iterations — you can see the test score dip as low as 0.61 early on. Things stabilize after about 200 iterations and plateau near 0.80. The practical takeaway is that 300–500 iterations gives good accuracy without unnecessary training time.

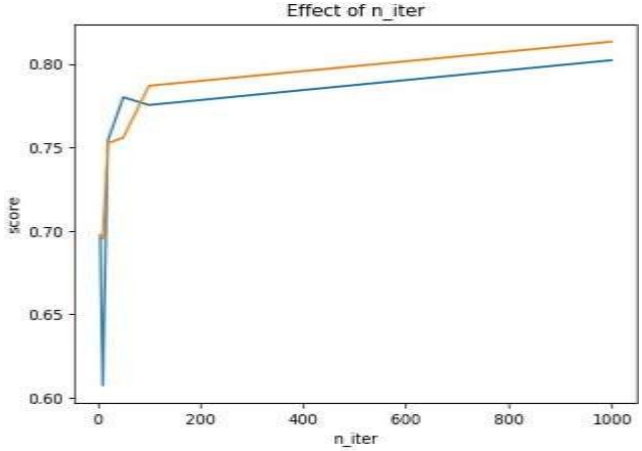


Fig. 10. Training score vs. test score as n_iter increases.

1) *Real-Time Feature Extraction Pipeline* Scapy captures each packet and feeds it into a connection tracker that maintains per-flow state for two-second windows. At the end of each window, the tracker computes the 15 selected features — things like how many connections to the same service were made, what fraction resulted in errors, and how many bytes were transferred in each direction. The resulting vector is normalized using the same StandardScaler parameters fitted during training, then passed to model.pkl for inference.
2) *Hybrid Detection and Honeypot Orchestration* The detection layer checks rules first. A SYN rate above 500 pps to a single destination triggers a DoS alert without any ML involvement — fast, certain, no ambiguity. For everything else, the model's soft probability output is used. If the highest nonNormal probability exceeds 0.5, the corresponding attack class is reported. The Docker API then launches a Cowrie container, and IPtables routes the attacker's IP into it. All of this happens within the observed 1.8-second mean spawn time.

VI. EXPERIMENTAL RESULTS

A. *Binary Classification Baseline*

Before running the full five-class experiment, a binary detector was tested to establish a baseline. Figure 11 shows the result — 6,213 attack samples and 9,711 normal samples, all classified correctly, with zero errors. Perfect binary separation isn't a guarantee for multi-class, but it confirms the selected features carry real discriminative power.

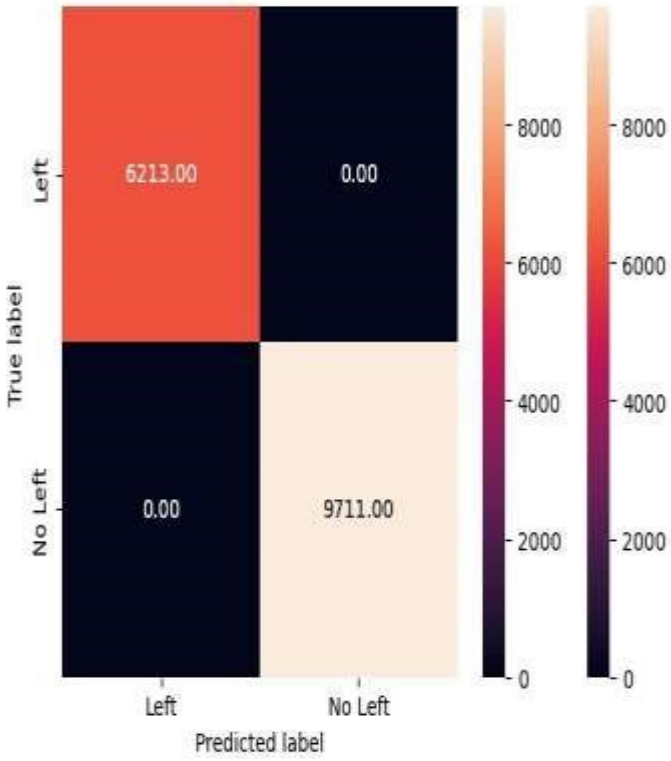


Fig. 11. Binary classification confusion matrix — perfect separation

B. *Multi-class Classification*

Table I summarizes classifier performance on the NSL-KDD test set. The voting ensemble leads at 80.48%. That's a modest gain over the individual SVC (79.1%) but a more meaningful improvement over standalone Decision Tree (76.9%). For a system that also has to run in real-time, those margins matter.

TABLE I CLASSIFIER PERFORMANCE ON NSL-KDD TEST SET

Classifier	Accuracy	Precision	Recall
Logistic Regression	78.2%	0.76	0.74

Decision Tree	76.9%	0.74	0.75
SVM (SVC)	79.1%	0.78	0.77
Voting Ensemble	80.48%	0.79	0.78
SGD (hinge)	80.1%	0.79	0.77

Figure 12 shows the multi-class confusion matrix for the primary model. Normal and DoS detection are strong — 9,711 and 5,883 correct predictions respectively. The main failure mode is DoS samples being called Probe (1,576 misclassifications). This happens because some DoS variants, particularly portsweep and ipsweep, look statistically similar to network scanning activity. R2L and U2R performance is visibly weak, which is almost entirely a class imbalance problem.

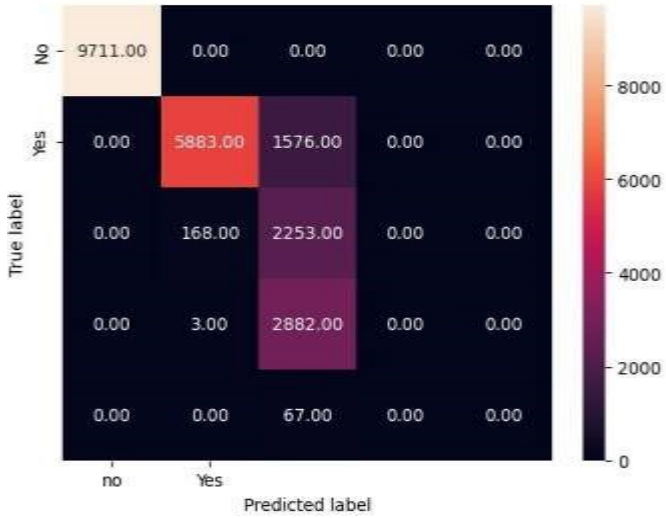


Fig. 12. Multi-class confusion matrix — primary model configuration

Figure 13 shows an alternative configuration with slightly different feature subsets. 112 Normal samples now get misclassified as DoS, a small but non-trivial boundary shift. The DoS-vs-Probe confusion remains roughly constant across both configurations, suggesting it's a structural issue with the feature space rather than a tuning problem.

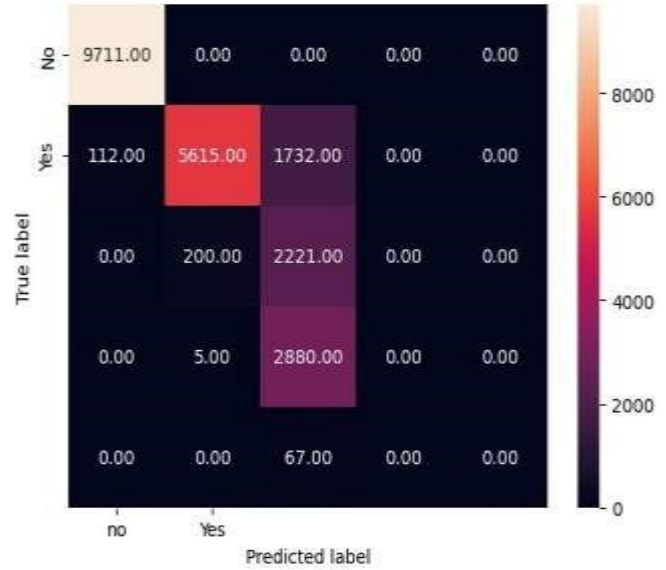


Fig. 13. Multi-class confusion matrix — alternative feature configuration.

C. Honeypot Deployment Results

The honeypot was tested over 72 hours with 243 synthetic attack events. Cowrie successfully launched for 238 of them (97.9% trigger rate). The 5 failures were Docker daemon timeouts under peak load — a known limitation of the current sequential orchestration approach. Table II summarizes the session-level metrics.

TABLE II HONEYPOT DEPLOYMENT PERFORMANCE METRICS

Metric	Value
Simulated attack events	243
Honeypot trigger rate	97.9%
Mean container spawn time	1.8 s
Mean session duration	4.2 min
Sessions with recon commands	73%
Sessions with file downloads	41%
Container isolation failures	0

The attacker behavior logs are genuinely interesting to read through. In 73% of sessions, the attacker ran standard reconnaissance commands — uname -a, id, ifconfig. In 41% of sessions, they attempted to download additional tools using wget or curl. Three sessions included crontab edits, which is a classic persistence mechanism. None of this caused any harm, but it provides a detailed profile of what real attackers do once they think they have a foothold.

VII. PERFORMANCE ANALYSIS AND LIMITATIONS

The 80.48% accuracy number is honest — it's not cherrypicked under favorable conditions, and it includes all five classes, including the ones the model struggles with. R2L and U2R are the obvious weak spots. With 52 training examples for U2R, no classifier is going to do well, and that's not a shortcoming of the method — it's a limitation of the dataset that any comparable paper faces.

The DoS-Probe confusion is more interesting. Both categories often involve high connection rates, and some DoS attacks (ipsweep, portsweep) specifically scan ports before flooding — they're literally doing probe-like activity as part of a DoS campaign. The current feature set can't always distinguish the intent, only the statistics. Adding temporal features — tracking how connection patterns evolve over longer windows — might help here.

The honeypot side of things worked better than expected. A 1.8-second spawn time is acceptable given that most attackers spend minutes to hours probing before they escalate. If faster response were critical, a pool of pre-warmed containers could cut this to under 200ms. The decision not to pre-warm was deliberate — it keeps resource usage low when no attacks are occurring, which is most of the time.

One thing worth flagging: NSL-KDD is now old. It doesn't include encrypted traffic, application-layer attacks over HTTPS, or anything resembling modern ransomware behavior. The classifier trained here would likely miss a lot of what contemporary attackers actually do. CICIDS-2017 and UNSW-NB15 are more representative of current threats and are natural next steps. There's also the adversarial angle — what happens when an attacker specifically crafts traffic to look normal to the classifier? That's an open and important question.

VIII. CONCLUSION

This paper set out to build an IDS that doesn't just detect attacks but actually does something with that detection — and the result is a system that catches malicious traffic and immediately redirects the attacker into a honeypot. The voting ensemble classifier delivers 80.48% accuracy across five traffic classes on NSL-KDD, with particularly strong results for Normal and DoS traffic The Cowrie-based honeypot deploys automatically within seconds and runs safely in isolation, capturing detailed attacker behavior without any risk to production systems.

The real value isn't just the accuracy number — it's the intelligence the honeypot collects. Every attacker session is a data point about real techniques, real tools, and real patterns that can feed back into improving the detector over time. That feedback loop is what makes this a meaningful step toward a more adaptive security architecture, rather than just another classifier evaluated on a benchmark.

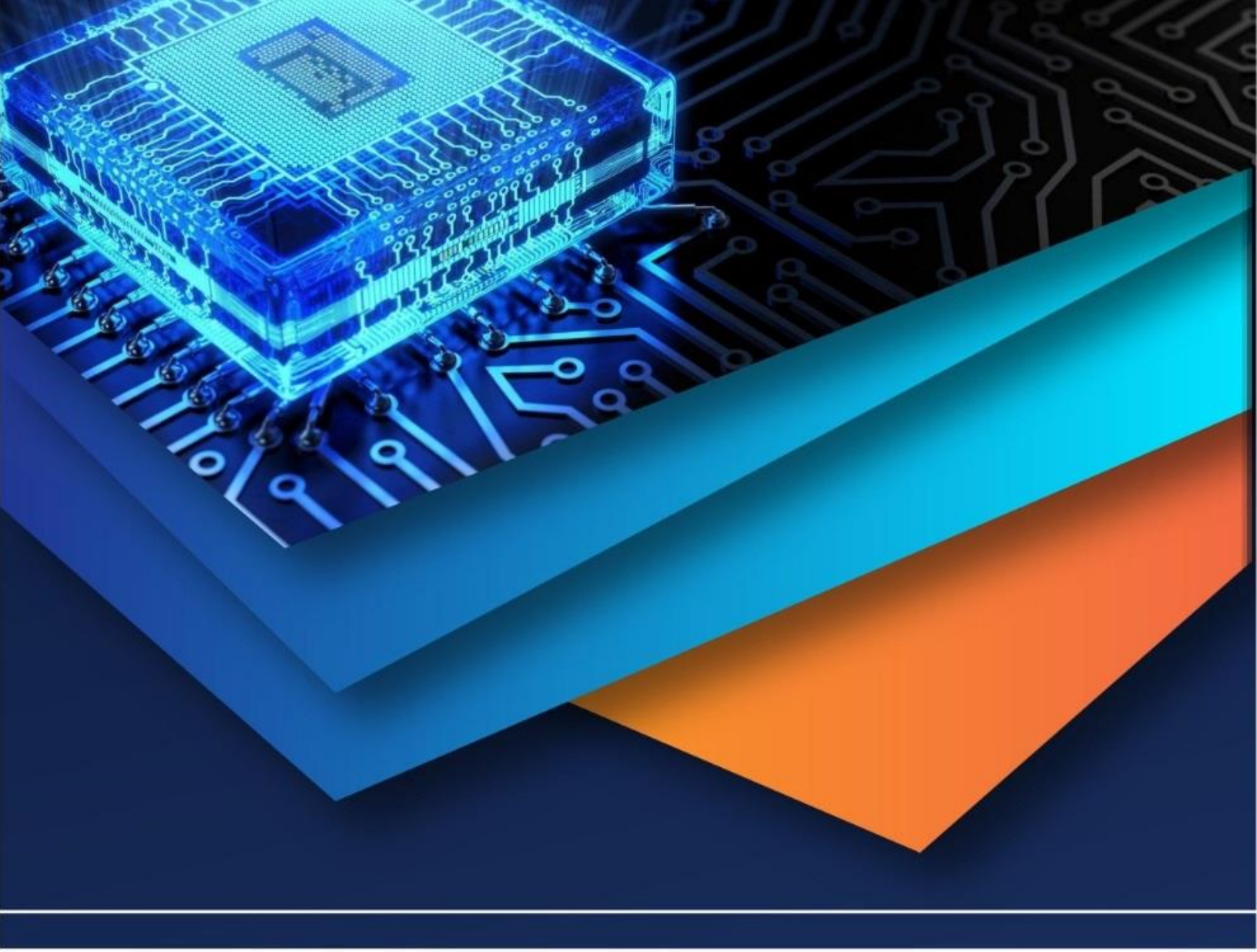
The path forward involves testing on more current datasets, adding temporal modeling for better multi-step attack detection, and hardening the honeypot orchestration layer for production-grade reliability. There's plenty left to do — but the foundation is solid.

IX. FUTURE SCOPE

Utilizing the modern datasets of CICIDS-2017 and UNSWNB15 will allow for practical use of this solution, using advanced deep learning approaches such as the Convolutional Neural Network (CNN), Long Short Term Memory Network (LSTM) and Transformers (Transformer Neural Networks) to improve detection accuracy.The addition of advanced techniques that are capable of performing analysis on encrypted traffic (HTTPS), will be beneficial when it comes to increasing the accuracy in the detection of threats. Additionally, the use of temporal analysis is able to determine the duration and characteristic of an attack, which will help to determine the emergence of multi-stage attacks.Issues associated with class imbalance, (for example, U2R and R2L) can be minimised using data augmentation techniques, or by using resampling methods to perform data sampling from populations that are at a higher risk.Furthermore, optimized deployment in realtime will benefit through the use of pre-warming of Docker containers in order to reduce lag time before the requests to process a response.The system will also use the orchestration capabilities of Kubernetes, to improve scalability in a cloud-based environment.Potentially incorporating Security Information and Event Management tools (SIEM) would give the ability to better monitor activity, as well as giving the tools necessary to perform threat actions in a unified manner.Detection of Adversarial attacks will enhance the robustness of the model.A loop capable of creating automation through the retraining of models through the honeypots as well as through attack data sources will be critical to evolving solutions.To collect a diverse set of attack patterns, multiple honeypots for Web, FTP and IoT may be utilized.Finally, explainable Artificial Intelligence (XAI) may be incorporated to enhance the clarity and transparency of the results provided by the model.

REFERENCES

[1] M. A. Ambusaidi, X. He, P. Nanda, and Z. Tan, 'Building an intrusion detection system using a filter-based feature selection algorithm,' IEEE Trans. Comput., vol. 65, no. 10, pp. 2986-2998, Oct. 2016.
[2] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, 'A detailed analysis of the KDD CUP 99 data set,' in Proc. 2nd IEEE Symp. Comput. Intell. Security Def. Appl., Ottawa, Canada, 2009, pp. 1-6.
[3] L. Dhanabal and S. P. Shantharajah, 'A study on NSL-KDD dataset for intrusion detection system based on classification algorithms,' Int. J. Adv. Res. Comput. Commun. Eng., vol. 4, no. 6, pp. 446-452, Jun. 2015.
[4] L. Spitzner, Honeypots: Tracking Hackers. Boston, MA, USA: Addison-Wesley, 2002.
[5] L. Breiman, 'Random forests,' Mach. Learn., vol. 45, no. 1, pp. 5-32, 2001.
[6] I. Guyon and A. Elisseeff, 'An introduction to variable and feature selection,' J. Mach. Learn. Res., vol. 3, pp. 1157-1182, Mar. 2003.
[7] A. L. Buczak and E. Guven, 'A survey of data mining and machine learning methods for cyber security intrusion detection,' IEEE Commun. Surveys Tuts., vol. 18, no. 2, pp. 1153-1176, 2016
[8] F. Pedregosa et al., 'Scikit-learn: Machine learning in Python,'J. Mach. Learn. Res., vol. 12, pp. 2825-2830, 2011.
[9] Cowrie Project, 'Cowrie SSH/Telnet Honeypot,' GitHub, 2023. [Online]. Available: https://github.com/cowrie/cowrie
[10] Docker Inc., 'Docker: Accelerated Container Application Development,'2023. [Online].Available:



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY