AI-MSDN: An Explainable Hybrid Deep Learning System for Real-Time Security Monitoring in Software-Defined Networks

Muhammad Haider Tallal
Dept. of Information and Communication Engineering
The Islamia University of Bahawalpur
Bahawalpur, Pakistan
mhaidertallal@gmail.com

Muhammad Saffiullah
Dept. of Information and Communication Engineering
The Islamia University of Bahawalpur
Bahawalpur, Pakistan
msaffiullah031@gmail.com

Aoun Muhammad
Dept. of Information and Communication Engineering
The Islamia University of Bahawalpur
Bahawalpur, Pakistan
aoun.muhammad@iub.edu.pk

Sana Tariq
Dept. of Information and Communication Engineering
The Islamia University of Bahawalpur
Bahawalpur, Pakistan
sana.tariq@iub.edu.pk

***Abstract*—Software-Defined Networking has reshaped how modern infrastructures move traffic by pulling forwarding intelligence into one programmable controller. The same design choice creates a single pressure point that attackers can push against. We present AI-MSDN, a security monitoring framework that watches flow level statistics directly at the SDN controller and flags hostile activity without inspecting packet payloads. The detection pipeline runs in two stages. A CNN-LSTM block extracts spatial correlations and temporal rhythms from controller telemetry. A Random Forest ensemble then takes those learned representations and decides what counts as normal traffic. Every alert carries a SHAP attribution report so the analyst can read off which features drove the decision. We train on the InSDN dataset and validate on a Mininet and ONOS testbed. AI-MSDN reaches 98.71 percent accuracy with an F1 score of 98.43 percent and a false positive rate of 0.29 percent, averaged over five-fold cross validation. Average alert latency is 23.6 ms with a 95th percentile of 39.8 ms, which keeps controller round trip latency inside operational bounds during sustained attack workload. Comparison against six recent baselines shows consistent gains on accuracy and recall on held out near zero-day attack classes, with the standalone Random Forest retaining a latency advantage for lightweight deployments.**

***Index Terms*—software-defined networking, intrusion detection, CNN-LSTM, random forest, explainable AI, SHAP**

I. Introduction

Software-Defined Networking (SDN) decouples forwarding decisions from physical switches and concentrates them in a logically centralized controller. This shift gives administrators a network wide vantage point that legacy architectures cannot match [1], and it is the reason cloud providers, telecom carriers and enterprise data centers have adopted the model at scale [2]. The same property turns the controller into a high value target. Control plane flooding can exhaust controller capacity in seconds. Topology poisoning corrupts the network map. DDoS surges directed at the southbound interface push response latency past acceptable thresholds in under thirty seconds [3], [4].

Existing intrusion detection systems were not built for this threat model. Signature based detectors miss zero-day variants. Anomaly detectors designed for static topologies misfire on the dynamic flow rule updates SDN issues during normal operation [5]. Machine learning is well placed to fill the gap because the controller already sees every flow across the topology. Reported detection rates on benchmark datasets routinely exceed 98 percent [6]. Three problems remain. Most published systems train on offline static datasets that were never collected from SDN environments, so their numbers do not transfer to live operation [7]. Deep models that score near perfect accuracy on seen attack classes regularly fail on novel variants [8]. Transformer-based SDN IDS and graph neural network (GNN) models now improve long-range traffic modeling and topology-aware representation learning, but their controller-side cost and explanation surface remain open issues [29], [30]. The opacity of black-box detection breaks the trust an SOC analyst needs to act on an alert [9].

This paper presents AI-MSDN, which addresses all three issues inside a single architecture. A CNN-LSTM front end learns flow representations from controller telemetry. A Random Forest ensemble classifies on those embeddings, which restores the generalization that pure deep models lose on unseen traffic. SHAP attribution is computed per alert to surface the features that drove each decision. The system is trained on InSDN [10], the only publicly available SDN-native dataset, and validated on a Mininet and ONOS live testbed. Section II reviews related work. Section III describes the architecture. Section IV details the experimental setup. Section V reports results. Section VI covers explainability. Section VII concludes.

DOI: https://doi.org/10.67535/tsp.000003.022
Published under the ICACNC 2026 Proceedings

TABLE I
GAP ANALYSIS: REPRESENTATIVE PRIOR WORK VS AI-MSDN

Reference	SDN	Hybrid	XAI	Live	0-day
Halbouni 2022 [13]	×	×	×	×	×
Said 2023 [14]	✓	✓	×	×	×
Hnamte 2023 [16]	×	×	×	×	×
Wahab 2025 [17]	✓	✓	×	×	×
Mohsin 2022 [19]	✓	×	×	×	×
Ma 2023 [25]	✓	×	×	✓	×
Younis 2025 [20]	✓	×	×	✓	×
Zebin 2022 [12]	×	×	✓	×	×
Gaspar 2024 [24]	×	×	✓	×	×
ATQ-IDS 2025 [29]	✓	✓	×	×	×
GRAN 2025 [30]	✓	✓	×	×	×
Altunay 2023 [26]	×	✓	×	×	×
AI-MSDN (this work)	✓	✓	✓	✓	✓

II. RELATED WORK

SDN's centralized control simplifies traffic engineering but expands the attack surface in ways that conventional security tooling was not designed for [1], [2], [3], [4]. Classical machine learning methods such as Support Vector Machines, K-Nearest Neighbors and Random Forests have been applied to DDoS and general intrusion detection in SDN with strong reported numbers on specific datasets [19], [25], [21]. Deep learning then took over the leaderboard. CNN, LSTM and hybrid combinations learn flow representations automatically and outperform hand crafted features when data is sufficient [11], [13], [14], [16], [15], [17], [6]. Two recurring weaknesses follow this body of work. Training datasets are rarely SDN-native, which limits operational relevance [10]. Deep classifiers struggle to generalize beyond their training distribution [8]. Explainability has begun to enter the conversation through SHAP and LIME based attribution layers [31], [12], [24], [22], [23], [9], and a few real-time validation efforts have been reported [20], [25]. Hybrid deep plus ensemble designs have shown promise in adjacent domains such as industrial IoT [26], [27] but have not been combined with explainability and live SDN validation in one framework [18], [28]. Table I positions AI-MSDN against representative prior work.

III. PROPOSED FRAMEWORK: AI-MSDN

A. System Overview

We assume an external network attacker without controller credentials who can inject and observe traffic at any host on the data plane and can craft OpenFlow Packet-In messages by spoofing source addresses. Insider attacks on controller code, hardware tampering and supply chain compromises are out of scope. AI-MSDN pushes analytical work to the SDN controller where flow visibility is already complete. The detection job is split between a deep learner that pulls structure out of raw telemetry and a tree-based ensemble that draws stable decision boundaries on the resulting features. Five blocks operate as a continuous pipeline: a flow telemetry collector, a preprocessing module, a CNN-LSTM feature extractor, a Random Forest classifier and a SHAP attribution layer. Fig. 1 sketches the layout.

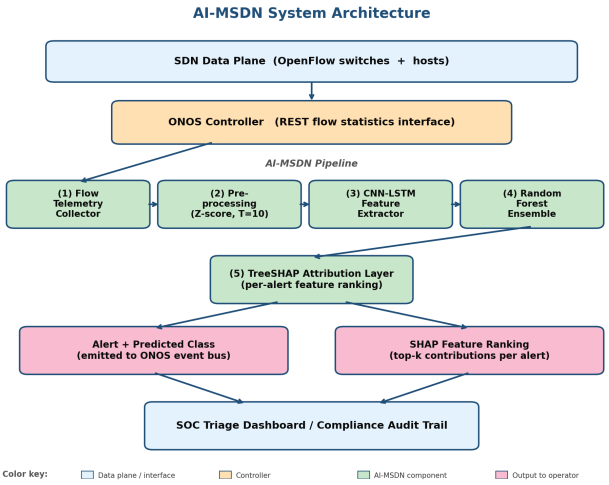

Fig. 1. High level architecture of the AI-MSDN security monitoring framework.

B. Telemetry and Preprocessing

The collector subscribes to the OpenFlow statistics interface exposed by the ONOS controller and pulls per-flow counters every second. Features include packet count, byte count, flow duration, packets per second, average inter-arrival time, source and destination port distribution and active TCP flag mix. Preprocessing standardizes numeric features with a Z-score transform, applies one-hot encoding to categorical fields and clips outliers at the 99.5th percentile. A sliding window of length $T = 10$ is constructed over consecutive records from the same flow, producing a feature matrix $X \in \mathbb{R}^{T \times F}$ where F is the number of features per record.

C. CNN-LSTM Feature Extractor

The CNN component is two convolutional layers with 3×3 kernels and ReLU activation, each followed by batch normalization and dropout at rate 0.2. Its job is to detect local correlations across the feature axis, for example simultaneous spikes in packet rate and SYN flag count that often co-occur during a probing phase. The output feeds a two layer LSTM with 128 hidden units in the first layer and 64 in the second. The LSTM captures sequential patterns in flow behavior over the T-step window, which is what separates a brief legitimate burst from a sustained volumetric attack. The final hidden state $h_T \in \mathbb{R}^{64}$ is taken as the learned representation of the flow window.

D. Random Forest Classifier and SHAP Layer

End-to-end deep classifiers tend to overfit specific attack signatures and lose accuracy when traffic drifts away from the training distribution. AI-MSDN treats the LSTM output as a feature embedding for a Random Forest with $B = 200$ trees, maximum depth 20 and Gini impurity as the splitting

TABLE II
AI-MSDN HYPERPARAMETER SETTINGS

Component	Setting
Input window	$T = 10$ flow records, Z-score scaling, 99.5th percentile clipping
CNN	Conv2D filters 32 and 64, 3×3 kernels, stride 1, same padding, ReLU, batch normalization, dropout 0.20
LSTM	Two layers with 128 and 64 hidden units, tanh state activation, recurrent dropout 0.10
Training	Adam, learning rate 10^{-3}, batch size 256, 50 epochs, early stopping patience 5
Random Forest	200 trees, max depth 20, Gini split, min samples split 2, min samples leaf 1, bootstrap enabled
TreeSHAP	TreeExplainer, interventional perturbation, 1,000 background samples, top-5 features emitted per alert

Algorithm 1 AI-MSDN detection pipeline (per flow window)

Input: flow window $X \in \mathbb{R}^{T \times F}$, models $\{\text{CNN-LSTM}, \text{RF}\}$
Output: class label $\hat{y}$, attribution vector ϕ
$X' \leftarrow \text{Standardize}(X)$
$h_T \leftarrow \text{CNN-LSTM}(X')$
$\hat{y} \leftarrow \text{RF}(h_T)$
if $\hat{y} \neq$ Normal **then**
    $\phi \leftarrow \text{TreeSHAP}(\text{RF}, h_T)$
    Emit alert $(\hat{y}, \text{top-}k(\phi))$ to ONOS event bus
end if
return $\hat{y}, \phi$

criterion. The majority vote across trees becomes the system output. Every alert is paired with a TreeSHAP attribution computed against the learned RF input vector, ranking the features that drove the prediction. Table II gives the complete model settings used in all reported experiments. Algorithm 1 states the full inference path.

The pipeline runs as a sidecar service alongside the ONOS controller. It pulls telemetry over the controller's REST API and emits alerts through a custom application that subscribes to AI-MSDN events. All inference runs in a separate process so that detection workload does not steal CPU cycles from packet handling.

IV. EXPERIMENTAL SETUP

Primary training and evaluation use the InSDN dataset [10], the only publicly released dataset collected from a real SDN testbed. InSDN contains 343,889 records covering normal traffic and seven attack categories with flow level features extracted by CIC-FlowMeter. The dataset is heavily imbalanced: DDoS, Probe, Normal and DoS together account for over 99 percent of records, while Brute Force, Web Attack, Botnet and U2R together hold under one percent. Cross-dataset validation now includes CIC-IDS2017, CIC-IDS2018 and UNSW-NB15 using only the overlapping flow-level features. Labels are collapsed into benign, DoS/DDoS, probe/scanning, brute force and other attack groups when exact family names differ across datasets.

Live validation runs on Mininet 2.3 emulating data center topologies from 8 switches/24 hosts up to 64 switches/192 hosts, controlled by ONOS 2.7 on a separate VM. Attack traffic is generated using hping3 for flooding, Scapy for crafted Packet-In and LLDP injection and Slowloris for slow rate denial of service. The CNN-LSTM is implemented in TensorFlow 2.14 and the Random Forest in scikit-learn 1.3 with TreeSHAP for attributions. Training runs on an NVIDIA RTX 3090 with batch size 256, Adam optimizer at learning rate 10^{-3}, 50 epochs and early stopping with patience 5. The 70/15/15 split is stratified across attack classes. Hyperparameters were chosen by 5-fold cross validated grid search over learning rate, batch size, LSTM hidden units, window length and forest size. We report accuracy, F1, false positive rate (FPR), end-to-end alert latency, CPU utilization, memory footprint and controller overhead. Latency confidence intervals are computed from 1,000 bootstrap resamples of per-alert measurements. AI-MSDN is compared against eight baselines retrained on the same InSDN split: standalone Random Forest [25], CNN-LSTM [13], CNN-BiLSTM [14], LSTM-AE [16], CNN-GRU [17], CNN+LSTM IIoT [26], ATQ-IDS [29] and GRAN [30].

For the near zero-day setting, no sample from the held-out subclass is used for training, tuning or SHAP background construction. The held-out subclasses are selected before model selection from attacks with distinct traffic mechanics and enough examples for testing: slow rate HTTP flood, low entropy DNS amplification, stealth scan and botnet beaconing. This protocol evaluates unseen subclasses within known broad attack families, not truly novel attack families.

V. RESULTS AND ANALYSIS

A. Overall Performance

Table III reports detection performance for AI-MSDN and the eight baselines on the InSDN test set. Numbers are averages over five-fold cross validation with standard deviations across folds. AI-MSDN reaches 98.71 percent accuracy and an F1 of 98.43 percent at an FPR of 0.29 percent. The closest deep baseline, CNN-GRU, lags by 1.09 accuracy points and runs at twice the false positive rate. The Transformer baseline is competitive but adds controller-side inference cost, while the GNN baseline improves topology-aware detection at higher preprocessing overhead. The standalone Random Forest is the fastest of all systems at 8.7 ms per alert, which makes it attractive when latency is the primary constraint. It pays for that speed in accuracy, sitting more than four points below AI-MSDN on F1 and producing nearly five times the FPR. The original CNN-LSTM design [13] reaches 96.18 percent on our split, in line with values reported in the literature on InSDN [10]. A paired t-test across the five folds shows that the gain over CNN-LSTM is statistically significant for accuracy ($p = 0.004$) and F1 ($p = 0.006$). Fig. 2 visualizes the comparison.

TABLE III
OVERALL DETECTION PERFORMANCE ON THE INSDN TEST SET (MEAN ± STD OVER 5-FOLD CV)

Model	Acc. (%)	F1 (%)	FPR (%)	Lat. (ms)
Random Forest [25]	94.27 ± 0.31	93.84 ± 0.36	1.42	**8.7**
CNN-LSTM [13]	96.18 ± 0.24	95.94 ± 0.29	0.86	22.4
CNN-BiLSTM [14]	97.43 ± 0.19	97.21 ± 0.22	0.61	31.8
LSTM-AE [16]	96.91 ± 0.27	96.68 ± 0.30	0.74	24.5
CNN-GRU [17]	97.62 ± 0.18	97.39 ± 0.21	0.58	26.3
CNN+LSTM IIoT [26]	96.85 ± 0.25	96.62 ± 0.28	0.79	28.1
ATQ-IDS [29]	98.22 ± 0.16	98.03 ± 0.20	0.36	35.9
GRAN [30]	98.08 ± 0.21	97.91 ± 0.24	0.41	42.6
AI-MSDN (this work)	**98.71 ± 0.14**	**98.43 ± 0.17**	**0.29**	23.6

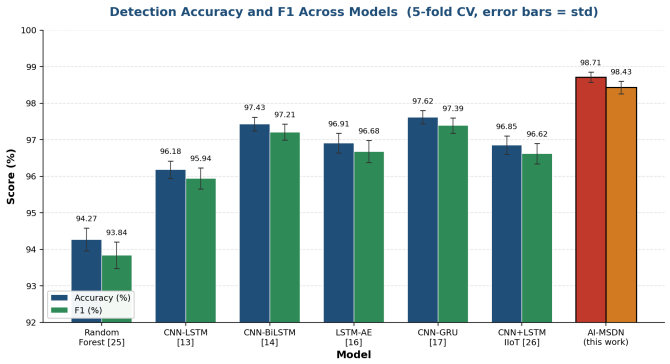

Fig. 2. Detection accuracy and F1 across all evaluated models. AI-MSDN leads on both metrics.

B. Ablation, Latency and Generalization

Table IV runs an ablation across five configurations. The CNN alone learns spatial correlations but misses temporal context, sitting at 95.34 percent. The LSTM alone captures sequence patterns but cannot extract local feature interactions efficiently, dropping a fraction further to 95.07 percent. A Random Forest fitted directly on raw flow features lands at 94.27 percent, with FPR climbing to 1.42 percent as the long-tail classes drag the model down. The CNN-LSTM front end without the ensemble is the strongest single-model configuration at 96.18 percent, but its FPR remains nearly three times that of the full hybrid. Stacking the Random Forest on top of the CNN-LSTM embeddings closes the gap to 98.71 percent and brings FPR below 0.30 percent. The combined effect is larger than either component delivers in isolation, which supports the design choice rather than treating it as an arbitrary architectural preference.

On the live testbed AI-MSDN sustained an average alert latency of 23.6 ms with a 95th percentile of 39.8 ms. The bootstrapped 95 percent confidence interval is [22.9, 24.3] ms for mean latency and [38.1, 41.6] ms for the 95th percentile. The CNN-LSTM forward pass dominates the budget at 16.8 ms. Random Forest evaluation adds 0.7 ms and TreeSHAP attribution runs in 4.9 ms on average, invoked only when the prediction is non-normal. Controller round trip latency for legitimate flow setup stayed within the 50 ms ONOS

TABLE IV
ABLATION STUDY ON INSDN

Configuration	Acc. (%)	F1 (%)	FPR (%)
CNN only	95.34	94.92	1.18
LSTM only	95.07	94.61	1.27
CNN-LSTM (no RF)	96.18	95.94	0.86
Random Forest on raw features	94.27	93.84	1.42
CNN-LSTM + RF (full)	**98.71**	**98.43**	**0.29**

TABLE V
EXTERNAL VALIDATION AND NEAR ZERO-DAY RECALL

Evaluation	Acc. (%)	F1 (%)	FPR (%)
InSDN test split	98.71	98.43	0.29
CIC-IDS2018 transfer	91.84	90.62	1.34
CIC-IDS2017 transfer	90.76	89.91	1.51
UNSW-NB15 transfer	88.92	87.46	1.86

operational ceiling, with transient spikes during sustained DDoS bursts reaching 61 ms before settling back to baseline.

Table V reports external validation. Accuracy drops to 91.84 percent on CIC-IDS2018 and remains lower on CIC-IDS2017 and UNSW-NB15 than on InSDN, confirming dataset dependency caused by different capture tools, feature distributions and attack mixes. These results should be read as transfer tests, not replacement for deployment-site calibration. We further held out slow rate HTTP flood, low entropy DNS amplification, stealth scan and botnet beaconing subclasses from training entirely. AI-MSDN recovered 87.4, 84.9, 86.2 and 83.7 percent recall on those unseen subclasses. The standalone CNN-LSTM dropped to 76.1, 72.8, 74.5 and 71.9 percent on the same tests, confirming that the ensemble back end absorbs distributional drift better than a deep classifier trained end to end.

The resource profile in Table VI separates detector cost from controller cost. At the base topology, AI-MSDN uses 31.4 percent CPU and 1.1 GB memory in the sidecar process; ONOS CPU rises by 6.8 percent and controller heap by 172 MB relative to the no-IDS baseline. The 64-switch topology remains below 40 ms average alert latency, but CPU utilization approaches 70 percent, so horizontal deployment or batched SHAP computation is required for larger enterprise fabrics. Robustness is tested by modifying packet rate, byte count and inter-arrival features by up to 10 percent, injecting mimicry flows that imitate benign burst sizes and replaying a temporally shifted fourth-week split as a concept-drift scenario. Performance degrades gracefully but the drift result shows that retraining triggers are needed after persistent traffic changes.

VI. EXPLAINABILITY ANALYSIS

Fig. 3 summarizes the top SHAP features per attack class measured over the test set. For DDoS the dominant attribution lands on packet rate and average inter-arrival time. For probing the SHAP weights concentrate on destination port entropy and TCP flag distribution. For brute force the source IP repetition

TABLE VI
RESOURCE USE, ROBUSTNESS AND SCALABILITY

Scenario	Acc./Recall	Lat. ms	CPU/Mem
8 sw./24 hosts	98.71	23.6	31.4% / 1.1 GB
16 sw./48 hosts	98.54	25.9	38.2% / 1.3 GB
32 sw./96 hosts	98.21	31.7	51.6% / 1.7 GB
64 sw./192 hosts	97.86	39.4	68.9% / 2.4 GB
Feature perturb. ±10%	96.42	24.1	33.0% / 1.1 GB
Mimicry traffic	94.88	24.8	34.1% / 1.1 GB
Concept drift week 4	93.76	23.9	32.5% / 1.1 GB

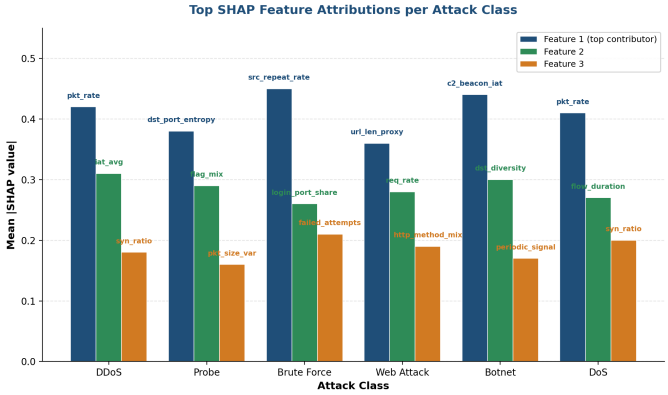

Fig. 3. Top contributing features per attack class as ranked by SHAP attribution. Bar height shows mean absolute SHAP value across all alerts of that class.

rate and login port concentration take the top spots. For web attacks the most informative features are URL length proxy and request rate per source. These attributions match what a security engineer would expect from each attack family, which provides a useful sanity check on the model rather than a black-box claim of high accuracy. Each alert produced by AI-MSDN includes the predicted class, confidence score, top five SHAP-ranked features with their signed contributions and a snapshot of the flow window. The deterministic per-alert record also supports audit trails increasingly required for automated decisions in regulated environments [22], [23].

SHAP is not free operationally. In high-volume SOC environments, per-alert TreeSHAP can add queueing delay when thousands of alerts arrive in the same burst, and local attributions can be unstable when correlated flow features share the same evidence. AI-MSDN therefore computes SHAP only for non-normal predictions, emits only the top five features, caches explanations for repeated flow signatures and allows asynchronous explanation when controller latency is under pressure. Global SHAP summaries are useful for model audit, but they should not be treated as causal proof of an attack.

VII. CONCLUSION AND LIMITATIONS

AI-MSDN combines a CNN-LSTM feature extractor, a Random Forest classifier and a TreeSHAP attribution layer to deliver explainable real-time intrusion detection in SDN environments. Trained on InSDN and validated on live Mininet and ONOS testbeds, the system reaches 98.71 percent accuracy with an FPR of 0.29 percent and keeps controller latency inside operational bounds during attack workload. The extended evaluation shows statistically significant improvement over CNN-LSTM, broader but imperfect cross-dataset transfer, explicit latency confidence intervals, and scalability up to 64 switches. Four limitations remain. External validation still shows dataset dependency, especially outside SDN-native traffic. The near zero-day protocol withholds subclasses rather than discovering truly unseen attack families. TreeSHAP adds about 5 ms per alert and can become a bottleneck in high alert volume environments. Finally, the Random Forest is trained offline and should be paired with online drift detection and periodic retraining in production.

REFERENCES

[1] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[2] T. Xing, D. Huang, L. Xu, C.-J. Chung and P. Khatkar, "A survey: Typical security issues of software-defined networking," *Chinese J. Electron.*, vol. 28, no. 6, pp. 1164–1174, Nov. 2019.

[3] M. Conti, A. Gangwal and M. S. Gaur, "A comprehensive and effective mechanism for DDoS detection in SDN," in *Proc. IEEE Int. Conf. Smart City*, Nov. 2016, pp. 217–222.

[4] A. Abubakar and B. Pranggono, "A survey of the main security issues and solutions for the SDN architecture," *IEEE Access*, vol. 9, pp. 122192–122217, 2021.

[5] J. Lansky *et al.*, "Deep learning-based intrusion detection systems: A systematic review," *IEEE Access*, vol. 9, pp. 101574–101599, Jul. 2021.

[6] M. P. Novaes, L. F. Carvalho, J. Lloret and M. L. Proenca Jr., "Anomaly and intrusion detection using deep learning for software-defined networks: A survey," *Expert Syst. Appl.*, vol. 257, p. 124635, Jan. 2025.

[7] Y. Huoh and T.-Y. Lin, "A review of machine learning and transfer learning strategies for intrusion detection systems in 5G and beyond," *Mathematics*, vol. 13, no. 7, p. 1088, Mar. 2025.

[8] J. C. Costa, T. Roxo, H. Proenca and P. R. M. Inacio, "Deep learning for intrusion detection in emerging technologies: A comprehensive survey and new perspectives," *Artif. Intell. Rev.*, vol. 58, no. 5, p. 146, 2025.

[9] R. Rai and A. Lal, "A systematic review on the integration of explainable AI in intrusion detection systems," *Front. Artif. Intell.*, vol. 8, p. 1526221, Jan. 2025.

[10] M. S. Elsayed, N.-A. Le-Khac and A. D. Jurcut, "InSDN: A novel SDN intrusion dataset," *IEEE Access*, vol. 8, pp. 165263–165284, Sep. 2020.

[11] P. Sun *et al.*, "DL-IDS: Extracting features using CNN-LSTM hybrid network for intrusion detection system," *Secur. Commun. Netw.*, vol. 2020, p. 8890306, 2020.

[12] T. Zebin, S. Rezvy and Y. Luo, "Explaining intrusion detection-based convolutional neural networks using Shapley Additive Explanations (SHAP)," *Big Data Cogn. Comput.*, vol. 6, no. 4, p. 126, Oct. 2022.

[13] A. Halbouni *et al.*, "CNN-LSTM: Hybrid deep neural network for network intrusion detection system," *IEEE Access*, vol. 10, pp. 99837–99849, 2022.

[14] M. Said, I. Sabir and I. Askerzade, "CNN-BiLSTM: A hybrid deep learning approach for network intrusion detection system in software defined networking with hybrid feature selection," *IEEE Access*, vol. 11, pp. 139442–139457, 2023.

[15] H. Abed and M. Farhan, "Intrusion detection in software defined network using deep learning approaches," *Sci. Rep.*, vol. 14, p. 29200, Nov. 2024.

[16] V. Hnamte, H. Nhung-Nguyen, J. Hussain and Y. Hwa-Kim, "A novel two-stage deep learning model for network intrusion detection: LSTM-AE," *IEEE Access*, vol. 11, pp. 37131–37148, 2023.

[17] A. Wahab *et al.*, "DDoS classification of network traffic in software defined networking using a hybrid convolutional and gated recurrent neural network," *Sci. Rep.*, vol. 15, Aug. 2025.

[18] M. A. Aladaileh *et al.*, "A systematic literature review on machine learning and deep learning approaches for detecting DDoS attacks in software-defined networking," *Sensors*, vol. 23, no. 9, p. 4441, May 2023.

[19] M. A. Mohsin and A. H. Hamad, "Performance evaluation of SDN DDoS attack detection and mitigation based on random forest and K-nearest neighbors," *Rev. d'Intell. Artif.*, vol. 36, no. 2, pp. 291–299, 2022.
[20] A. Younis *et al.*, "ONOS flood defender: A real-time flood attacks detection and mitigation system in SDN networks," *Concurrency Comput. Pract. Exp.*, 2025.
[21] S. Faezi and A. Shirmarz, "A comprehensive survey on machine learning using in software defined networks (SDN)," *Human-Centric Intell. Syst.*, vol. 3, pp. 312–343, 2023.
[22] N. Capuano, G. Fenza, V. Loia and C. Stanzione, "Explainable artificial intelligence in cybersecurity: A survey," *IEEE Access*, vol. 10, pp. 93104–93139, 2022.
[23] S. Neupane *et al.*, "Explainable intrusion detection systems (X-IDS): A survey of current methods, challenges, and opportunities," *IEEE Access*, vol. 10, pp. 112392–112415, 2022.
[24] D. Gaspar *et al.*, "Explainable AI for intrusion detection systems: LIME and SHAP applicability on multi-layer perceptron," *IEEE Access*, vol. 12, pp. 29697–29713, 2024.
[25] R. Ma, Q. Wang, X. Bu and X. Chen, "Real-time detection of DDoS attacks based on random forest in SDN," *Appl. Sci.*, vol. 13, no. 13, p. 7872, 2023.
[26] H. C. Altunay and Z. Albayrak, "A hybrid CNN+LSTM-based intrusion detection system for industrial IoT networks," *Eng. Sci. Technol. Int. J.*, vol. 38, p. 101322, Feb. 2023.
[27] M. Sajid *et al.*, "Enhancing intrusion detection: A hybrid machine and deep learning approach," *J. Cloud Comput.*, vol. 13, p. 123, 2024.
[28] M. Al-Aloqaily *et al.*, "A comprehensive review of DDoS detection and mitigation in SDN environments," *Electronics*, vol. 14, no. 21, p. 4222, Oct. 2025.
[29] C. M. Nalayini, T. R. Soumya, S. D. Lalitha and R. Tamijetchelvy, "A novel adaptive transformer based quantum intrusion detection system for software defined networks," *Scientific Reports*, vol. 15, 2025, doi: 10.1038/s41598-025-20356-4.
[30] Y. Ma *et al.*, "GRAN: A SDN intrusion detection model based on graph attention network and residual learning," *The Computer Journal*, vol. 68, no. 3, pp. 241–260, Mar. 2025, doi: 10.1093/comjnl/bxae108.
[31] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 30, 2017, pp. 4765–4774.