International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 11 | Nov - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

A Next-Generation Adaptive Firewall: Real-Time Anomaly Detection and Kernel-Level Rule Automation Using Deep Reinforcement Learning

Lavanya M
Dept of Artificial Intelligence and Data Science
EAST WEST INSTITUTE OF TECHNOLOGY
Bengaluru, Karnataka

Prof. Dilip S
Dept of Information Science and Engineering
EAST WEST INSTITUTE OF TECHNOLOGY
Bengaluru, Karnataka

Madhu S
Dept of Artificial Intelligence and Data Science
EAST WEST INSTITUTE OF TECHNOLOGY
Bengaluru, Karnataka

Shiva Kumar K
Dept of Artificial Intelligence and Data Science
EAST WEST INSTITUTE OF TECHNOLOGY
Bengaluru, Karnataka

Jyothsna B
Dept of Artificial Intelligence and Data Science
EAST WEST INSTITUTE OF TECHNOLOGY
Bengaluru, Karnataka

***Abstract*—Conventional firewalls depend on static rules and signatures that administrators must update by hand, leaving networks exposed to zero-day exploits and volumetric DDoS attacks during the delay between detection and response. This paper presents the design of an adaptive firewall that closes this loop autonomously. A live packet-sniffing module extracts flow features, a machine-learning classifier compares them with a learned behavioural baseline, and a Proximal Policy Optimization (PPO) agent selects a mitigation action that is translated into nftables or eBPF/XDP rules and enforced in the Linux kernel. A reward-based feedback loop relaxes rules that block legitimate traffic, and temporary rules expire when the threat subsides. We describe the architecture, the Markov decision process formulation, the PPO training objective, and a virtualised testbed built on Ubuntu 22.04 and Kali Linux. Performance goals of over 96% detection accuracy, under 3% false positives, and under 50 ms detection-to-enforcement latency are stated as design targets; experimental validation is planned for the next project phase.**

***Keywords*—adaptive firewall, anomaly detection , deep reinforcement learning, proximal policy optimization, nftables , eBPF , DDoS mitigation, zero-day attacks.**

I. INTRODUCTION

Firewalls remain the first line of defence for enterprise, cloud, and IoT networks, yet most deployments still enforce rule sets written and revised by people. Stateful packet inspection and signature-based intrusion detection work well against known threats, but they cannot recognise traffic that matches no existing signature, and every rule change waits on an administrator. Even next-generation firewalls that add application identification and intrusion prevention remain largely signature-driven. Against automated attacks such as DDoS floods and zero-day exploits, the interval between detection and a deployed rule can be long enough to cause serious damage.

Machine-learning intrusion detectors narrow the detection gap by modelling normal behaviour and flagging deviations [1]–[4]. Most, however, stop at raising alerts; a human must still decide what to block and write the rule. Reinforcement learning (RL) offers a natural way to close this gap, because the choice of firewall action is a sequential decision whose consequences (blocked attacks, blocked legitimate users, added latency) can be expressed as a reward. Existing RL firewall studies are, however, typically evaluated in simulation and do not show rules being enforced on a live kernel [5].

This work designs an adaptive firewall that connects the whole chain: live packet capture, behavioural anomaly detection, PPO-based decision making, and automatic rule injection into the Linux kernel. The main contributions are:

- an end-to-end architecture that turns live traffic into kernel-level filtering rules without human intervention;
- a Markov decision process (MDP) formulation of firewall rule management with a multi-objective reward that balances security against network performance;
- a rule-lifecycle mechanism that relaxes rules causing false positives and expires temporary rules to prevent rule bloat; and
- a reproducible virtualised testbed and evaluation plan with explicit, measurable performance targets.

The remainder of the paper is organised as follows. Section II reviews related work, Section III states the problem and objectives, Section IV presents the architecture, Section V details the learning formulation, Section VI describes the implementation environment, Section VII gives the evaluation plan and targets, and Sections VIII–X discuss applications, limitations, and future work.

II. RELATED WORK

A. Learning-based anomaly detection

Because attack traffic is rare and constantly changing, many studies train models on normal behaviour and treat poorly reconstructed samples as anomalies. Autoencoder-based detectors have been applied to NSL-KDD and related data, combining statistical analysis with deep autoencoders [2], tuning autoencoder performance [3], and adding memory modules to sharpen the boundary between normal and abnormal traffic [4]. These detectors are effective at identifying deviations, but they are offline analysers: their output is a score or an alert, and no mechanism converts it into an enforced policy. Sommer and Paxson also caution that detectors that perform well on benchmark data can behave very differently in

operational networks, which motivates evaluating the proposed system on live traffic [1].

B. Firewall policy management

Formal approaches such as neutrosophic Petri nets model how packet-filtering firewalls process traffic, but the resulting models are static and do not learn from experience [6]. RL has been proposed to tune policies automatically; a recent PPO-based study formulates firewall rule optimisation as an MDP and reports gains over static and value-based baselines on simulated traffic [5]. Such work demonstrates the promise of policy-gradient methods, but leaves open the problems of live packet capture, real-time enforcement, and rule cleanup, which are the focus of the present design.

C. Kernel-level enforcement

Moving filtering into the kernel reduces per-packet cost. Work on faster iptables implementations based on eBPF shows that Linux packet filtering can be made scalable [7], iKern demonstrates intrusion detection and prevention inside the kernel using eBPF [8], and the eXpress Data Path (XDP) enables programmable packet processing at the earliest point in the receive path [9]. These technologies provide the enforcement substrate that an autonomous policy agent needs.

D. Positioning

Table I summarises the gap. The proposed system combines behavioural detection, learned decision making, and live kernel enforcement in one loop.

TABLE I. Qualitative comparison of approaches

Approach	Detects unseen	Learns	Acts alone	Kernel rules
Static / SPI firewall	No	No	No	Yes
Signature IDS / NGFW	Limited	No	Partly	Partly
Autoencoder IDS [2]–[4]	Yes	Offline	No	No
RL policy tuning [5]	Depends	Yes	Simulated	Not shown
Proposed system	Yes	Yes	Yes	Yes

III. PROBLEM STATEMENT AND OBJECTIVES

Static, signature-based firewalls cannot adapt to zero-day exploits or fast-changing DDoS traffic, require continuous human effort, and are prone to configuration errors. Learning-based detectors improve visibility but usually do not act, and published RL firewall studies rarely reach live, kernel-level deployment. The problem addressed here is to build a self-improving firewall that detects behavioural anomalies in real time and enforces mitigation rules in the kernel without human intervention, while keeping false positives low.

The objectives are to (i) establish a behavioural baseline of normal traffic with machine learning; (ii) capture and inspect packets in real time with little added delay; (iii) detect attack patterns, including novel and low-rate DDoS traffic; (iv) convert detections into kernel-level rules that block malicious sources immediately; (v) monitor the effect of each rule and relax it if it blocks legitimate traffic; and (vi) remove temporary rules once the threat has passed.

IV. PROPOSED SYSTEM

A. Architecture

The system is organised as a pipeline with a feedback loop (Fig. 1). Traffic entering the ingress interface is captured and summarised into flow features, scored against a learned baseline, passed to a PPO agent that chooses an action, and converted into a kernel rule. Every decision is logged, and the observed effect of each rule flows back to the agent as a reward.

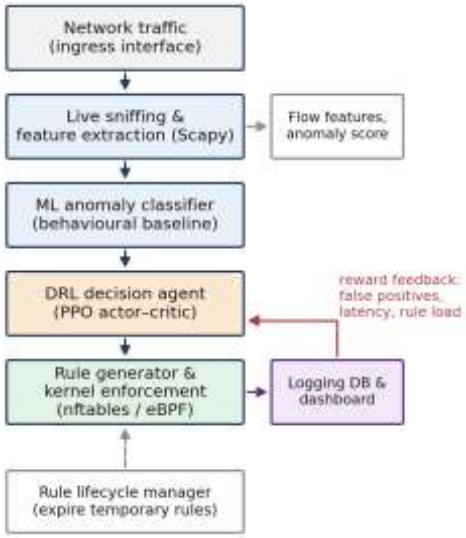

Fig. 1. Architecture of the proposed adaptive firewall.

B. Modules

Live data reader. Uses Scapy to capture packets on an interface in promiscuous mode without interrupting connections, and aggregates them into flow records with features such as source and destination address, protocol, packet size, inter-arrival time, and flow duration.

Smart threat finder. A machine-learning classifier compares live flow features with a baseline learned from historical and live traffic, and outputs an anomaly score or class label covering DDoS surges and zero-day behaviour.

Intelligent decision maker. A PPO actor–critic agent receives the flow state and anomaly evidence and selects a mitigation action: allow, drop, or rate-limit.

Automatic rule enforcer. Translates the chosen action into kernel syntax, for example an nftables rule that drops traffic from an offending source address, and installs it in the kernel packet-filtering tables. eBPF/XDP is the planned path for earlier, faster drops.

Rule lifecycle manager. Tracks every automatically created rule, relaxes rules that are found to block legitimate traffic, and deletes temporary rules when the threat subsides so that the rule table stays small and fast.

Logging and dashboard. Stores detected threats, blocked sources, rule changes, and performance metrics, and presents them to administrators, who may override automated decisions or trigger retraining.

C. Operational workflow

(1) Packets arrive at the ingress interface. (2) Scapy captures them and extracts flow features. (3) The classifier scores the flows against the baseline. (4) The PPO agent selects an action for anomalous flows. (5) The enforcer generates and installs the corresponding kernel rule. (6) The system observes the effect on legitimate traffic and returns a reward, adjusting or removing the rule if it causes disruption. (7) All events are logged and shown on the dashboard.

V. LEARNING FORMULATION

A. Markov decision process

Firewall control is modelled as a Markov decision process [17] (Fig. 2). At each step t the agent observes a state s_t, chooses an action a_t, and receives a reward r_t.

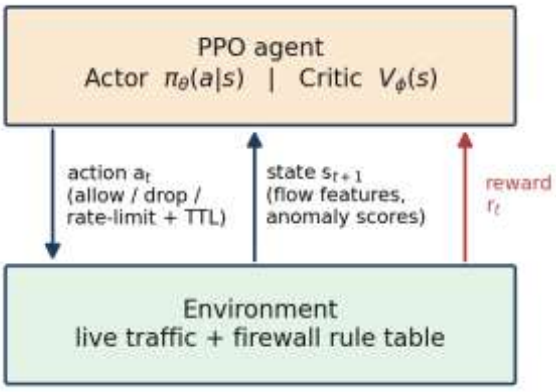

Fig. 2. MDP interaction between the PPO agent and the network.

State. A normalised vector of flow features (for example rate, packet size, inter-arrival time, flow duration, protocol and port encodings), the classifier's anomaly score, and summary statistics of the current rule table.

Action. A discrete choice among allow, drop, and rate-limit for a flow or source, together with a rule lifetime (time-to-live) for temporary rules.

Reward. A weighted sum that rewards blocking malicious traffic and penalises blocking legitimate traffic, missed attacks, added latency, and rule-table growth:

$$r_t = w_1 B_t - w_2 F_t - w_3 M_t - w_4 L_t - w_5 N_t \quad (1)$$

where B_t is malicious traffic blocked, F_t legitimate traffic wrongly blocked, M_t malicious traffic missed, L_t added decision latency, N_t the number of active rules, and w_i are tunable weights. The weights determine the trade-off between security and availability and will be tuned during Phase II.

B. Proximal Policy Optimization

PPO is an on-policy actor–critic method that keeps each policy update close to the previous policy [10]. It was preferred over value-based methods such as DQN [12] for its training stability and its ability to handle large, structured action spaces. Let π_θ be the actor and V_φ the critic. The probability ratio and the clipped surrogate objective are

$$\rho_t(\theta) = \pi_\theta(a_t|s_t) / \pi_{\theta old}(a_t|s_t) \quad (2)$$

$$L^{CLIP}(\theta) = E_t[\, \min(\, \rho_t \hat{A}_t,\ clip(\rho_t,\ 1-\varepsilon,\ 1+\varepsilon)\ \hat{A}_t\,)\,] \quad (3)$$

where ε is the clipping range. The advantage $\hat{A}_t$ is computed with Generalized Advantage Estimation [11]:

$$\hat{A}_t = \Sigma_{l \geq 0} (\gamma\lambda)^l \delta_{t+l}, \quad \delta_t = r_t + \gamma V_\varphi(s_{t+1}) - V_\varphi(s_t) \quad (4)$$

The critic is trained by minimising the squared error between $V_\varphi(s_t)$ and the observed return. Clipping prevents a single update from changing the policy drastically, which matters in a firewall because an over-aggressive policy could block legitimate traffic.

C. Safeguards

Because rules are injected into a running kernel, the design includes several safeguards: each automatic rule carries an expiry time; the feedback loop relaxes a rule when legitimate flows are affected; every change is logged for audit; and administrators can override any decision from the dashboard. We also plan a dry-run mode in which the agent's actions are recorded but not enforced, and a protected list of critical addresses that may never be blocked.

VI. IMPLEMENTATION ENVIRONMENT

The prototype is developed and tested in an isolated virtual network managed with Oracle VirtualBox. An Ubuntu 22.04 LTS machine hosts the firewall and all AI components, while a Kali Linux machine acts as the attacker, generating scans and exploit traffic with Nmap, Hping3, and Metasploit. Table II lists the software stack.

TABLE II. Software stack

Function	Tool
Language	Python 3.10+
Packet capture	Scapy [14]
Rule enforcement	nftables/iptables [15]; eBPF/XDP planned [9]
RL agent	Stable-Baselines3 (PPO) with a Gym-style environment [13]
Deep learning	TensorFlow or PyTorch [20]
Data preparation	Pandas, scikit-learn [19]
Virtualisation	Oracle VirtualBox

Minimum hardware is a quad-core CPU with 8 GB of RAM (16 GB recommended for DRL training), 40 GB of storage (100 GB SSD recommended for logging and datasets), and a network interface that supports promiscuous mode.

VII. EVALUATION PLAN AND TARGETS

The classifier will first be trained and validated offline on a public intrusion-detection benchmark such as NSL-KDD [16] or CICIDS2017 [18], and the complete system will then be exercised on live traffic in the virtual testbed under port scans, exploit attempts, and volumetric floods from the attacker machine. Reported metrics will include accuracy, precision, recall, F1-score, false-positive rate, detection-to-enforcement latency, and the maximum number of concurrent connections handled. Table III lists the design targets. These are goals derived from the project requirements, not measured results; the final column is reserved for Phase II measurements.

TABLE III. Design targets and measured results

Metric	Target	Measured
Detection accuracy	> 96%	To be measured
False-positive rate	< 3%	To be measured
Detection-to-rule latency	< 50 ms	To be measured
Concurrent connections	≥ 50,000	To be measured
Stability under DDoS	Remains operational	To be measured

VIII. APPLICATIONS

The design targets settings in which manual rule maintenance does not scale: enterprise networks and data centres facing DDoS and scanning campaigns; cloud and hybrid environments that need consistent policy across tenants; healthcare and financial systems where both protection and availability are critical; IoT and 5G deployments with many heterogeneous devices; and critical infrastructure where fast, unattended response is valuable.

IX. CHALLENGES AND LIMITATIONS

Several challenges must be managed. False positives can interrupt legitimate services, so the reward weights and safeguards are central to safe operation. Adversaries may attempt to poison training data or evade the model, so robustness to adversarial machine learning is required. Running deep-learning inference on every packet can add latency, which motivates flow-level features and kernel offload. Encrypted traffic limits payload inspection, so the system relies on behavioural metadata. Automated kernel-rule injection carries the risk of instability if the automation logic is faulty. Finally, DRL training demands significant compute and data. The current work is a design and prototype description: no experimental results are reported yet, and the targets in Table III have not been validated.

X. SNAPSHOTS

This section presents snapshots captured from the running prototype dashboard. They document the implemented interface; the values shown come from the prototype in simulation mode.



Fig. 3. Main dashboard of the prototype, showing live traffic statistics, SLA benchmark checks and the baseline profiler.

Fig. 3 shows the main dashboard of the prototype (Nexus Adaptive Firewall), running in simulation mode. The top cards report live flow ingestion, threats intercepted, active mitigations, mitigation latency and self-healing corrections. The benchmark panel compares detection accuracy, false positive rate, response time, connection scalability and forwarding overhead against their targets. The lower panels summarise the autoencoder-based baseline profiler, the self-healing engine and the integrated threat-intelligence feeds.



Fig. 4. Active kernel mitigations panel with model status and per-source enforcement details.

Fig. 4 shows the active kernel mitigations. The summary bar reports the training status of the models and the accuracy, precision, execution overhead and reconstruction error shown by the dashboard. Each row of the table lists a source address, the enforcement action (for example rate limit), the threat class (zero-day anomaly, port scan, SYN flood or UDP burst), model confidence, an explainable-AI attribution, the rule age against its time-to-live, and an Unblock control for manual override.



Fig. 5. Network port activity and service sentinel listing open ports, bound processes and risk levels.

Fig. 5 shows the port sentinel, which inspects listening sockets on the host. For each port it lists the state, the service and its function, the bound process, a risk level (high exposure, sensitive or core service) and an enforcement action to close the port. The filters separate all, open, closed and quarantined ports, which helps the administrator find risky services such as exposed Windows networking and file-sharing ports.



Fig. 6. Real-time anomaly score chart, attack vector distribution and a quarantined port.

Fig. 6 shows the rolling flow anomaly scores plotted against the rate-limit and drop thresholds, together with the distribution of detected attack vectors. Below them, the port sentinel shows a port that has been dropped at the kernel and quarantined, with a Re-Open Port action that lets the administrator reverse the decision.

XI. CONCLUSION AND FUTURE WORK

We presented the design of an adaptive firewall that links live packet sniffing, behavioural anomaly detection, PPO-based decision making, and kernel-level rule automation in a single self-correcting loop. By treating rule management as a sequential decision problem and adding rule relaxation and expiry, the design aims to reduce administrative effort and configuration errors while responding at machine speed to zero-day and DDoS attacks.

Phase II will implement and evaluate the full prototype against the targets in Table III, tune the reward weights, and add explainable AI so that administrators can see why a flow was blocked. Planned extensions include LLM-based policy generation, multi-cloud policy synchronisation, and predictive threat intelligence.

ACKNOWLEDGMENT

The authors thank the Department of Artificial Intelligence and Data Science, East West Institute of Technology, Bengaluru, for guidance and support.

REFERENCES

[1]R. Sommer and V. Paxson, “Outside the closed world: On using machine learning for network intrusion detection,” in Proc. IEEE Symp. Security and Privacy, 2010, pp. 305–316.

[2]C. Ieracitano, A. Adeel, F. C. Morabito, and A. Hussain, “A novel statistical analysis and autoencoder driven intelligent intrusion detection approach,” Neurocomputing, vol. 387, pp. 51–62, 2020.

[3]W. Xu, J. Jang-Jaccard, A. Singh, Y. Wei, and F. Sabrina, “Improving performance of autoencoder-based network anomaly detection on NSL-KDD dataset,” IEEE Access, vol. 9, pp. 140136–140146, 2021.

[4]B. Min, J. Yoo, S. Kim, D. Shin, and D. Shin, “Network anomaly detection using memory-augmented deep autoencoder,” IEEE Access, vol. 9, pp. 104695–104706, 2021.

[5]Y. R. Bhagwat, “Adaptive firewall strategy generation and optimization based on reinforcement learning,” Int. J. Multidisciplinary Research (IJFMR), vol. 7, no. 6, Nov.–Dec. 2025.

[6]J. K. Madhloom, Z. H. Noori, S. K. Ebis, O. A. Hassen, and S. M. Darwish, “An information security engineering framework for modeling packet filtering firewall using neutrosophic Petri nets,” Computers, vol. 12, no. 10, p. 202, 2023.

[7]S. Miano, M. Bertrone, F. Risso, M. Vásquez Bernal, Y. Lu, and J. Pi, “Securing Linux with a faster and scalable iptables,” ACM SIGCOMM Comput. Commun. Rev., vol. 49, no. 3, 2019.

[8]H. J. Hadi, M. Adnan, Y. Cao, F. B. Hussain, and N. Ahmad, “iKern: Advanced intrusion detection and prevention at the kernel level using eBPF,” Technologies, vol. 12, no. 8, art. 122, 2024.

[9]T. Høiland-Jørgensen et al., “The eXpress Data Path: Fast programmable packet processing in the operating system kernel,” in Proc. ACM CoNEXT, 2018, pp. 54–66.

[10]J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” arXiv:1707.06347, 2017.

[11]J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, “High-dimensional continuous control using generalized advantage estimation,” arXiv:1506.02438, 2015.

[12]V. Mnih et al., “Playing Atari with deep reinforcement learning,” arXiv:1312.5602, 2013.

[13]A. Raffin et al., “Stable-Baselines3: Reliable reinforcement learning implementations,” J. Mach. Learn. Res., vol. 22, no. 268, pp. 1–8, 2021.

[14]Scapy Project, “Scapy: Packet manipulation program,” documentation. [Online]. Available: https://scapy.net

[15]Netfilter Project, “nftables.” [Online]. Available: https://netfilter.org/projects/nftables/

[16]M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the KDD CUP 99 data set,” in Proc. IEEE Symp. Comput. Intell. Security and Defense Applications (CISDA), 2009, pp. 1–6.

[17]R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[18]I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, “Toward generating a new intrusion detection dataset and intrusion traffic characterization,” in Proc. 4th Int. Conf. Information Systems Security and Privacy (ICISSP), 2018, pp. 108–116.

[19]F. Pedregosa et al., “Scikit-learn: Machine learning in Python,” J. Mach. Learn. Res., vol. 12, pp. 2825–2830, 2011.

[20]A. Paszke et al., “PyTorch: An imperative style, high-performance deep learning library,” in Advances in Neural Information Processing Systems (NeurIPS), vol. 32, 2019.

[21]F. Valenza, D. Bringhenti, and G. Marchetto, “An optimized approach for assisted firewall anomaly resolution,” IEEE, 2023.

[22]D. Bringhenti, G. Marchetto, and R. Sisto, “Automated firewall configuration in virtual networks,” IEEE, 2023.

[23]Q. Zhang, X. Gong, and L. Guo, “All-optical QPSK pattern recognition in high-speed optoelectronic firewalls,” IEEE, 2023.

[24]F. Valenza, J. Yusupov, D. Bringhenti, and G. Marchetto, “An optimized approach for assisted firewall anomaly resolution,” IEEE Access, 2023.

[25]D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, “Automated firewall configuration in virtual networks,” 2023.