IAES International Journal of Artificial Intelligence (IJ-AI)
Vol. 15, No. 6, December 2026, pp. 5159~5175
ISSN: 2252-8938, DOI: 10.11591/ijai.v15.i6.pp5159-5175

Queue-sensitive autonomous congestion control using deep reinforcement learning

Osemwegie Omoruyi[1], Moses Opaleye[1], Jane Nkwor[1], Kennedy Okokpujie[1,2]
[1]Department of Electrical and Information Engineering, College of Engineering, Covenant University, Ota, Nigeria
[2]Department of Auditing, College of Accounting Science, University of South Africa, Pretoria, South Africa

Article Info

Article history:

Received Jan 13, 2026
Revised Aug 18, 2026
Accepted Sep 20, 2026

Keywords:

Active queue management
Bufferbloat
Deep reinforcement learning
Double-deep Q-network
Exponential weighted moving average
Network simulator 3
Random early detection

ABSTRACT

Bufferbloat mitigation at modern bottleneck routers is essential for maintaining low queuing delay without sacrificing link utilization. However, the manual parameter tuning requirements of random early detection (RED) remain a significant barrier to deployment. This paper investigates how three value-based deep reinforcement learning (DRL) algorithms, deep Q-network (DQN), double-deep Q-network (DDQN), and double-deep Q-network with prioritized experience replay (DDQN-PER), can autonomously optimizing RED's queue weight parameter (w_q) in latency-sensitive network using network simulator 3 (ns-3). Through systematic zero-shot evaluation under out-of-distribution (OOD) conditions with 10× increase in bottleneck severity, DDQN-PER achieves 32% lower queuing delay than controlled delay (CoDel) at twice the maximum training load, with near-perfect flow fairness (Jain's index 0.997) and full link utilization, without retraining. A universal fairness collapse observed at three times the maximum training load across all algorithms, including passive DropTail, is characterized as a fundamental transmission control protocol with CUBIC congestion control (TCP CUBIC) architectural boundary rather than an algorithmic failure. These findings established that optimized RED can outperform CoDel in latency-sensitive networks.

This is an open access article under the CC BY-SA license.

Corresponding Author:

Moses Opaleye
Department of Electrical and Information Engineering, College of Engineering, Covenant University
Km. 10 Idiroko Road, Canaan Land, Ota, Ogun State, Nigeria
Email: moses.opaleyepgs@stu.cu.edu.ng

1. INTRODUCTION

The emergence of latency-critical applications, including the fifth generation (5G)-enabled tactile internet [1], [2], Industry 4.0 automation systems [3], [4], and real-time interactive services [5], has transformed the performance requirements of modern network infrastructure. These applications necessitate low latency, high reliability, and strong quality of service (QoS) assurance [6]. While these technologies offer fast access, the resulting traffic aggregation places unprecedented stress on the wired transport backhaul and core aggregation points. In these high-throughput environments, raw bandwidth is no longer the sole determinant of QoS, instead, the minimization of end-to-end delay has become the dominant performance objective. At the center of this challenge lies the bottleneck router, where the accumulation of packets in large buffers (a phenomenon also called bufferbloat) induces significant latency, often rendering high-speed links unusable for real-time traffic [1], [2].

The number of internet users rose from 3.1 billion in 2015 to 4.1 billion in 2020 [7]. As of 2025, over 5.5 billion people worldwide use the internet [8], [9]. This increase in growth has continually led to highly unpredictable and dynamic traffic patterns. Active queue management (AQM) serves as the defense against

bufferbloat in routers by proactively dropping or marking packets before the physical buffer reaches saturation [10]. Random early detection (RED), a foundational AQM algorithm, has been widely studied in the literature for its proactive approach to congestion control through early packet dropping or explicit congestion notification (ECN) marking. However, despite its role in addressing bufferbloat, RED's reliance on static parameters limits its ability to adapt to dynamic network conditions [11], [12].

The queue weight (w_q) used in computing the exponential weighted moving average (EWMA) is a standard fixed value of 0.002. While other RED parameters, according to infrastructure requirements, stakeholders, or user-end goals, can be fixed, the w_q itself should be dynamic because it determines the filter sensitivity used to estimate the average queue size (Avg_q). A small w_q makes the filter too slow to detect sudden congestion, leading to buffer overflow, while a large w_q makes the filter react to temporary traffic bursts, causing unnecessary packet drops and low link utilization. Newer standards like controlled delay (CoDel) were created to avoid these tuning problems.

This study is the first to systematically benchmark learned w_q optimization against CoDel, under zero-shot out-of-distribution (OOD) conditions by introducing a double-deep Q-network with prioritized experience replay (DDQN-PER) framework that dynamically adjusts the w_q of RED in a simulated environment, using network simulator 3 (ns-3). The term this a queue-sensitive framework because w_q is treated as a state-dependent control variable rather than a fixed constant.

This study makes five contributions to the field of learning-based AQM. First, present the first systematic comparison of learning-based w_q control against CoDel, demonstrating a 32–33% sustained delay advantage at 2–3× maximum training loads. Second, propose a topology-invariant three-dimensional normalized state representation that is scalable. Third, establish that PER is a critical architectural component for deep reinforcement learning (DRL) in AQM. Fourth, characterize the phenomenon of fairness collapse at 3× maximum training load, where all AQMs degrade, but DDQN-PER maintained a 33% delay advantage. Fifth, by systematically evaluating zero-shot performance at both 2× (high fairness) and 3× (graceful degradation) training loads, we map the scalability boundary to the impact of transmission control protocol with CUBIC congestion control (TCP CUBIC).

2. LITERATURE REVIEW

The literature reviewed recent advances in AQM such as CoDel [13] and proportional integral controller-enhanced (PIE) [14] and learning-based congestion control. These schemes manage queues proactively to limit queuing delay. The review also covered learning-based congestion control.

2.1. Foundational active queue management

RED makes drop decisions based on the computation of the Avg_q, using the EWMA low-pass filter [15], defined in (1).

$$Avg_q = (1 - w_q) \text{x } Avg'_q + w_q \text{ x } q \quad (1)$$

Where q is the instantaneous queue occupancy at time t, and w_q is the queue weight. However, subsequent studies have shown that REDs performance is highly sensitive to static parameter settings. Early attempts to address this limitation introduced reinforcement learning mechanisms, for instance [16] and [17] employed Q-learning to dynamically adjust packet drop probabilities, demonstrating improved performance over RED. Similarly, [18] applied reinforcement learning to adapt ECN marking thresholds in software-defined networks. While several RED-based approaches improve packet marking or dropping decisions, they commonly retain fixed adaptive EWMA queue-weight parameters (w_q), though widely accepted, it has shown to delay congestion response and increase queueing delay under varying load conditions [19].

2.2. Delay-aware and parameter-less standards

To overcome RED's parameter complexity, the community shifted toward delay-aware AQMs such as CoDel and PIE. CoDel mitigates bufferbloat by monitoring the minimum packet sojourn time over a sliding window [13] and is widely adopted due to its simplicity, robustness, and parameter-free design, on the other hand PIE mitigates bufferbloat by periodically calculating a drop probability based on both the current queuing delay and its trend (increasing or decreasing) [14]. In this study, we compare against CoDel (RFC 8289) as the primary delay-based baseline because while PIE remains the standard for fixed-function cable hardware, CoDel represents a state-of-practice for general, parameter-less adoption across the wider internet ecosystem. While advanced variants like FQ-CoDel and CAKE incorporate per-flow queuing,

these operate outside the single-queue architecture this work targets, making CoDel the appropriate iso-architectural baseline.

2.3. Deep reinforcement learning

Recent advances in DRL have enabled more sophisticated congestion management strategies [20]. Approaches such as [21] and [22] leverage neural networks to adapt congestion windows and queuing weights, outperforming traditional TCP variants. Within the AQM domain, [23] and [24] applied deep learning to optimize drop rates and threshold parameters. However, many of these methods rely on standard deep Q-network (DQN) architecture, which is susceptible to overestimation bias [22]. In noisy network environments, it can lead to unstable policies hindering optimal performance. This study employs DDQN-PER to ensure little to no overestimation bias that prioritizes optimal action policies.

2.4. The role of exponential weighted moving average sensitivity

The w_q is the primal parameter for RED optimization because it is used to compute the EWMA Avg_q, a fundamental operation that directly influences all subsequent drop/mark decisions in RED [23]. The w_q defines the temporal sensitivity of congestion detection and directly influences delay stability [25], [26]. Figure 1 illustrates the flow chart of RED, showing how the algorithm manages congestion.

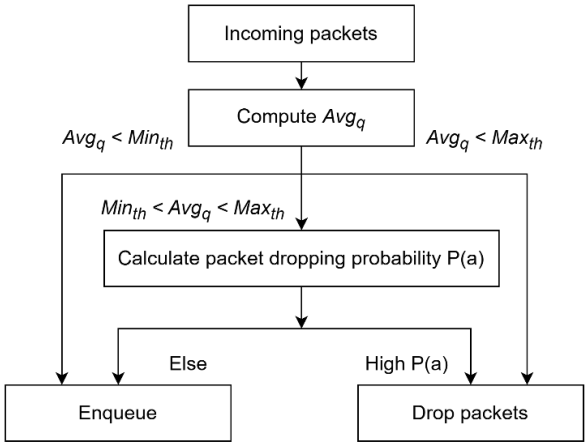

Figure 1. Flow chart RED [27]

In [12] provided a crucial foundation by demonstrating that a DQN-based agent can autonomously adjust w_q to achieve improved throughput (49.9 Mbps), reduced delay (0.949s vs 0.964s for RED), and lower packet loss (8.38% vs. 8.51%). While [12] demonstrated the viability of learning-based w_q control via DQN, achieving good performance over RED, their work left some critical questions unanswered:

i) How does performance distribute across competing flows?
ii) How does it compare to modern delay-based AQMs like CoDel?
iii) Does the learned policy generalize beyond trained conditions?
iv) What are the OOD performance boundaries? This study answered these questions through systematic comparison of value-based DRL algorithms ([28]–[30]), establishing DDQN-PER's superiority in AQM.

3. SYSTEM MODEL

3.1. Network environment and topology

The dumbbell topology is utilized in ns-3 as shown in Figure 2. A systematic evaluation of simulation platforms was conducted, ns-2 [31], ns-3 [32], Cooja simulator [33] and Mininet [18]. Following the review by [34] this study used ns-3, following the internet engineering task force (IETF) topology standards for AQM implementation [10]. The network is made up of a bottleneck of 5 Mbps data rate and 10 ms propagation delay. This study utilized 50 Mbps access links data rate during training, validation, and testing, then increased to 500 Mbps during the zero-shot generalization validation on OOD. The physical buffer limit is set to 100 packets by employing the bandwidth-delay product (BDP). Given an aggregate round-trip time (RTT) of approximately 40 ms and a 5 Mbps bottleneck capacity, the BDP is calculated to be 17.26 packets, the set buffer size is

approximately 5.8 times the BDP. This buffer design provides sufficient operational space for bufferbloat to occur. Full-duplex links are used to maintain high realistic fidelity. TCP CUBIC (RFC 8312) is selected as the transport protocol because user datagram protocol (UDP) transmits continuously regardless of queue state, it unsuitable for evaluating the effectiveness of drop or mark-based AQMs. This study utilized three traffic categories: Forward traffic, reverse traffic, and bursty web-like traffic. Reverse traffic ensure that the forward flows are subjected to natural acknowledgment and acknowledgment (ACK) compressions, forming a stochastic feedback loop that establishes a rigorous test for the agent's stability. Table 1 contains the traffic model setups. This multi-category traffic model is designed to simulate the complex dynamic behavior of the internet, often characterized by chaotic user patterns that complicated queue stabilization.

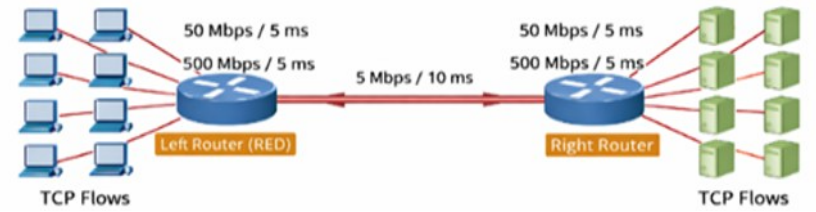

Figure 2. The proposed topology using IETF standard AQM topology

Table 1. The ns-3 environment and baseline configuration

Category	Parameters	Value/configuration
Topology	Network architecture	Dumbbell topology
	Bottleneck capacity/delay	5 Mbps/10 ms
	Access link capacity (train/OOD)	50 Mbps/500 Mbps (OOD)
	Access link propagation delay	5 ms
	Simulation time	100 s
Traffic model	TCP protocol/segment size	TCP CUBIC (RFC 8312)/1448 bytes
	Traffic composition	BulkSend (90%) + exponential OnOff (10%)
DropTail (baseline)	Queue size	100 packets (including RED and CoDel)
CoDel	Target delay/interval	5 ms/100 ms
RED	Minimum/maximum threshold	10, 30 packets
	w_q/max drop probability	0.002/0.02
Enhanced RED for DRL	Dynamic parameter	EWMA weighting factor (w_q)

3.2. Markov decision process formulation

Markov decision process **(**MDP) was defined by the tuple $M = (S,A,P,R,\gamma)$ where S denotes the state, A represents the action space, P is the state transition probability, R defines the reward and $\gamma \in (0,1]$ is the discount factor [17]. The state vector $S_t \in S$ provides the agent with a scale-invariant perception of the bottleneck congestion. Scale invariance is achieved by normalizing raw metrics (e.g., queue size) into a range [0, 1] by dividing by the upper RED threshold (Max_{th}). This allows the learned policy to generalize across different link capacities and buffer configurations. The proposed state representation is a three-dimensional normalized vector as shown in (2). Where, $\frac{q_{avg}}{Max_{th}}$: normalized exponentially-weighted moving average queue size, $\frac{q_{inst}}{Max_{th}}$: normalized instantaneous queue size, loss term: normalized packet loss rate (the divisor 10.0 scales loss percentage).

$$S_t = [\frac{q_{avg}}{Max_{th}}, \frac{q_{inst}}{Max_{th}}, \min\left(\frac{Loss}{10.0}, 1.0\right)] \quad (2)$$

The action space $A_t \in A$ focuses exclusively on the w_q as the primal control variable. This study utilized a discrete action space consisting of 16 levels mapped via a log-hybrid distribution. A log-hybrid distribution combines logarithmic and linear spacing to provide specific control authority across different congestion scales. This study employ sensitivity levels ranging from 0.0002 to 0.150. The logarithmic range (0.0002 to 0.002) provides high-precision sensitivity for noise filtering. The higher linear range (0.003 to 0.150) provides the high-reactivity control needed to prevent buffer saturation during acute traffic surges. The reward function $R_t \in R$ is designed as a weighted composite of three fundamental network performance metrics,

utilization, delay, and packet loss, to capture the inherent trade-offs in AQM. The weights reflect the relative priority of each metric in the target deployment context. Reward is calculated as (3).

$$R_t = (0.9 \cdot \boldsymbol{U}) - (0.5 \cdot \boldsymbol{L}) - (0.5 \cdot \boldsymbol{D}) \quad (3)$$

Where U is link utilization; L is packet loss; D is end-to-end delay. This study clipped the final reward to the interval [−1,1]. This study implements an online reinforcement learning control loop by directly coupling the DRL agent with the ns-3 simulator via a synchronous ZeroMQ (REQ–REP) interface. This design enables real-time interaction. More complex reward formulations risk instability and poor convergence due to varying reward scales, a concern addressed in foundational DRL work [29], where reward clipping was critical to ensuring stable gradients and consistent training across different tasks. Utilization is weighted most strongly (0.9) because maintaining high link utilization during congestion is the primary AQM objective. Delay and packet loss are assigned equal weights (0.5) to prevent the agent from disproportionately optimizing one metric at the expense of the other. At each control interval (τ = 0.2 s), ns-3 transmits the current state vector to the agent, which performs a forward inference pass and returns the updated queue sensitivity parameter. ns-3 then injects the new weight into the bottleneck router and resumes simulation. This synchronous step-wise interaction is shown in Figure 3.

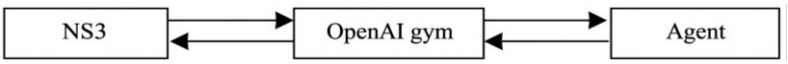

Figure 3. General setup flow

4. THE DDQN-PER FRAMEWORK

Figure 4 illustrates the three-phase framework design. This work follows a progressive architectural evaluation, starting from DQN, to DDQN, and DDQN-PER, as detailed in Table 2. Three-phase experimental design in Table 3 pool setups. The agent trains on stochastically sampled 30/40/60 flow scenarios (Rngrun 1-20). Every 50 episodes, performance is evaluated on 60-flow scenarios (Rngrun 26-30) for checkpoint selection. The best model undergoes in-distribution testing (Rngrun 31-50, 60 flows) and zero-shot OOD testing (120/180 flows, 500 Mbps access link data rate) without retraining. The proposed systematic comparison of DQN, DDQN, and DDQN-PER under identical network conditions shows which DRL performs best when trained on the same reproducible setup.

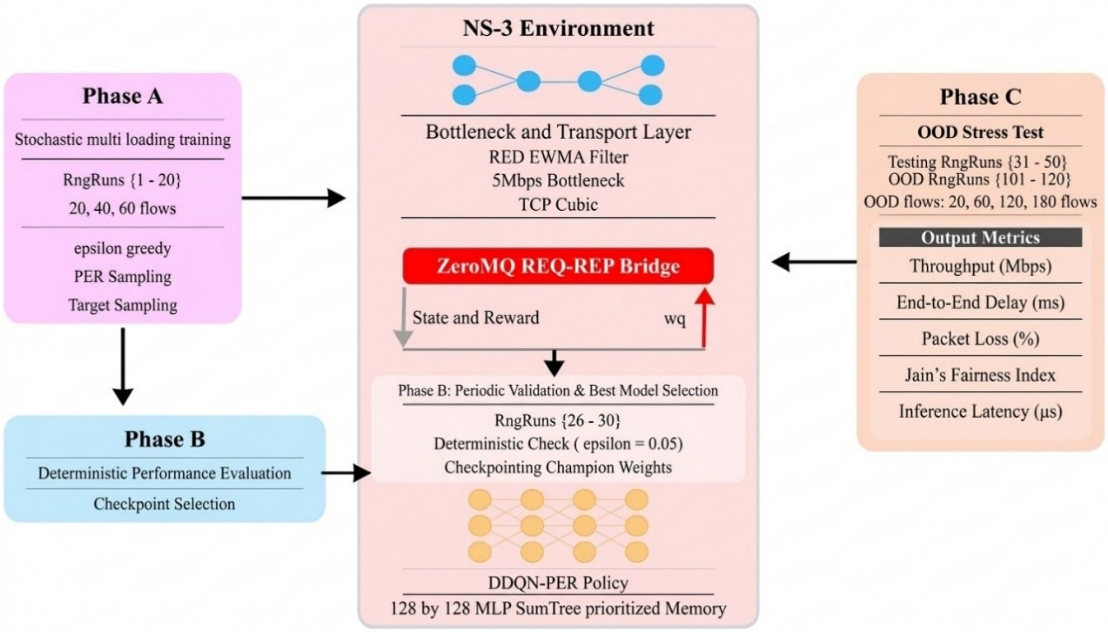

Figure 4. Model design setup

4.1. Reproducibility

To address reproducibility Table 2 provides deterministic and stochastic simulation parameters. This study employs ns-3's MRG32k3a generator with fixed Rngseed = 42 to ensure all experiments sample from the same master random stream, guaranteeing reproducibility. The Rngrun parameter selects the starting position within independent sub-streams of this master generator. A common concern is whether sequential Rngrun values (e.g., 21, 22, and 23) produce statistically independent results or merely correlated variations. The MRG32k3a generator addresses this through mathematical guarantees: each Rngrun accesses a distinct independent stream rather than simply incrementing a seed value. Firstly, the full period is calculated as $2^{191} \approx 3.1 \times 10^{57}$ (length before the master sequence repeats) secondly the independent streams are calculated as $2^{127} \approx 1.7 \times 10^{38}$ available (far exceeding practical needs) and thirdly the per-stream period is calculated as $2^{64} \approx 1.8 \times 10^{19}$ random numbers per Rngrun. Thus, Rngrun = 21 and Rngrun = 22 access mathematically independent streams with zero correlation, not sequential positions in the same stream [35]. This ensures all Rngrun are statistically independent while remaining reproducible. The choice of linear ε-greedy decay as the exploration strategy is to maintain the hyperparameter configuration established by [28], [29] ensuring performance between DQN, DDQN, and DDQN-PER are solely architectural.

4.2. Neural network architecture

The policy and target network utilize a multi-layer perceptron (MLP) with two hidden layers of 128 neurons each. Rectified linear unit (ReLU) to prevent gradient vanishing and ensure stable backpropagation. The 128-neuron width provides optimal balance between approximation capacity and computational efficiency. The target Q-value uses the DDQN update rule to eliminate overestimation bias as formalized in (4).

$$Y_t^{DDQN} = R_{t+1} + \gamma Q\left(s_{t+1}, arg\max_a Q\ (s_{t+1}, a; \theta_t); \theta_t^-\right) \quad (4)$$

Where θ and θ_t^- represents the online and target network weights, respectively. The online network selects the action with maximum Q-value, while the target network evaluates that action's value, decoupling selection from estimation. To prioritize learning from critical congestion events, we implement PER using a SumTree data structure. Transition sampling probability is proportional to absolute TD error as shown in (5).

$$P(i) = p^{\propto}{}_i / \sum_{\mathrm{k}} p^{\propto}{}_k \quad (5)$$

This study applies importance sampling (IS) weights during the optimization step. This combination of DDQN evaluation and prioritized sampling enables stable policy convergence despite inherent dynamic network traffic.

Table 2. Deep reinforcement learning

Feature	DQN (baseline)	DDQN	DDQN-PER
Network architechture	128 by 128 MLP	128 by 128 MLP	128 by 128 MLP
Target calculation	Standard bellman	DDQN (decoupled)	DDQN (decoupled)
Replay strategy	Uniform random	Uniform random	SumTree
Sampling bias	none	none	Important sampling
Loss function/Epsilon decay	Huber loss/linear	Huber loss/linear	Huber loss/Linear
Priority exponent (alpha)/IS annealing (beta)	n/a	n/a	0.6/0.4 → 1.0
Learning rate/discount factor	0.00025/0.99	0.00025 /0.99	0.00025/0.99
Batch size/buffer capacity	64/100,000	64/100,000	64/100,000
Activation/optimizer	ReLU/Adam	ReLU/Adam	ReLU/Adam

Table 3. Deterministic seeding and statistical protocol

Protocol components	Value/pool	Purpose
ns-3 global seed	Rngseed = 42	Fixed world geometry and traffic layout
Training/validation pool	Rngrun = 1 – 20/26 - 30	Stochastic learning/periodic “best-models” checkpoint selection
Testing/OOD stress pool	Rngrun = 31 – 50/101 - 120	Testing/out of distribution robustness validation
Agent initialization (seed)	{100, 200, 300, 400, 500}	Fixed PyTorch weights and epsilon choices
Load tiers (training)/OOD	{30, 40, 60}/{20, 60, 120, 180}	Multi-load policy discovery/scalability testing

4.3. Evaluation design

This study utilized a periodic validation strategy where, every 50 episodes, the agent is evaluated in deterministic mode ($\epsilon = 0.05$) on the held-out validation pool. This allows for a "best-model selection" strategy. The final performance is evaluated using the best model weights and not weights from the final training episode which might not be the best-performing model.

4.4. Unified deep reinforcement learning-random early detection optimization algorithm

The unified algorithm is summarized in Algorithm 1. This study introduced a secondary evaluation protocol and assessment procedure, as detailed in Algorithm 2, which represents the test phase. During this phase, the trained best model, frozen in its deterministic state (ϵ = 0), is subjected to entirely unseen environmental realizations (Runs 101–120).

Algorithm 1. Load invariant RED sensitivity optimization (DDQN-PER)

1: Init online net θ, target net θ^-, SumTree buffer D, hyperparams: k = 64, η = 0.00025, training_frequency = 4, target_update = 1000, α = 0.6 and β = 0.4 → 1.0, episodes = 1000.
2: For episode e = 1 to 1000 do: sample traffic load L ∈ {30, 40, 60}, RngRun ∈ {1 - 20}; reset env, observe state s_0. (L denotes the aggregate number of TCP flows across both Left and Right access links (applied symmetrically, L/2 per side).
3: While not done:
4: Select a_t via ε-greedy: $a_t = \underset{a}{\text{argmax}} Q_\theta(s_t,\text{a})$ with probability 1 – ε; Execute a_t in ns-3 → get r_t, s_{t+1}
5: Store transition (s_t, a_t, r_t, s_{t+1}) in D with maximum priority P_{max}
6: If steps % training_frequency == 0 then: sample k transitions using priority P(i) = ${p^\propto}_i / \sum_k {p^\propto}_k$; compute IS weights w_i and DDQN target Y_i using θ^-; Update priorities $P(i) = |Y_i - Q_\theta(s_i, a_i)|$; gradient descent step on weighted Huber Loss.
7: If steps % target_update_frequency == 0 then $\theta^- \leftarrow \theta$
8: Anneal ε and PER β.
9: Every 50 episodes: validate on RngRun 26-30 (ε = 0.05); If $R_{avg,val} > R_{peak}$, save weights θ as best_model.pth.
10: Final evaluation: load best_model.pth and evaluate on Test Pool {RngRun 31 – 50}.

Crucially, this assessment includes traffic intensities (20, 60, 120, and 180 flows) which include flows outside the training distribution (Lmax = 60). ε = 0 for theoretical limit assessment. At this point, epsilon isolates the learned capabilities of the agents without exploratory contamination. To avoid policy-specific bias, each DRL configuration is evaluated across five independent neural network initialization seeds. This separation of environmental randomness (Rngrun) from policy randomness (agent seeds) enables fair, statistically robust comparisons between traditional AQM and DRL-optimized AQM.

Algorithm 2. The zero-shot OOD generalization and scalability assessment

1: Inputs: best_models (DQN, DDQN, DDQN_PER), Baseline Suite {RED, CODEL, DROPTAIL}, OOD topology: 500 Mbps access links, 5 Mbps bottleneck; load tiers L ∈ {20, 60, 120, 180}, evaluation RngRun ∈ {101 – 120}, agent seeds: {100, 200, 300, 400, 500}.
2: For each algorithm A in {Baseline Suite U DRL suite}, each load tier L, each agent seed S (DRL only), each evaluation seed R:
3: Initialize ns-3 with L flows, seed R and OOD topology.
4: If A is DRL: Load weights from best_models; Set ε = 0; Initialize inference latency buffer $\Delta T_{inf} \leftarrow []$
5: At each decision step t: Start timer; Select a_t; Record ΔT_{inf}.append(elapsed); Execute a_t in ns-3, observe s_{t+1}.
6: Else: execute fixed baseline logic (RED or CODEL).
7: At simulation end: compute Jains fairness index J = $(\sum_i x_i)^2$ / n$\sum_i x^2{}_i$; aggregate throughput delay and packet loss.
8: End loops.
9: Output: aggregate statistics (compute mean and std) and computational overhead.

5. EXPERIMENTAL RESULTS

5.1. Value estimation and reward plot

Figure 5 illustrates the Q-value plot. DQN exhibits a significantly higher predicted Q-value, stabilizing at approximately 20, which shows the overestimation bias inherent in noisy networking environments. In contrast, both DDQN and the proposed DDQN-PER maintain a lower and more stable value estimation, stabilizing at approximately 13.5. This plot averages across five independent seeds. The subtle shaded region indicates tight variance and consistent convergence behavior across runs. In Figure 6, the reward plot demonstrates a consistent upward trend for all three agents. However, DDQN-PER reaches higher reward at scattered points across the plots much faster than the other algorithms, particularly visible in episodes 0 to 200, demonstrating better sample efficiency. The shaded region represents the standard deviation, with wider

variance reflecting natural training variability across runs. This matters in real-world applications because each episode might be expensive, training time may be limited, and faster convergence means quicker deployment.

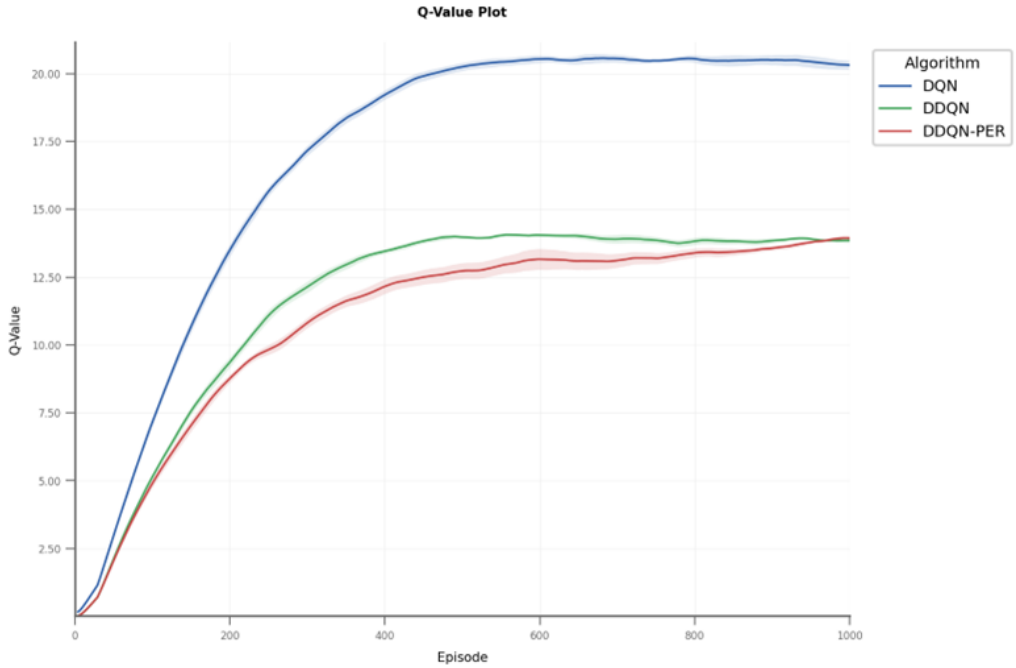

Figure 5. Q value



Figure 6. Reward plots

5.2. Test result

Table 4 shows the test result done on the highest traffic flow (60) on the training topology (50 Mbps access links) with ε = 0.05. All algorithms achieve high throughput showing high link utilization. Differences emerge in delay and loss behavior. DDQN-PER achieves the lowest delay at 153.5 ± 0.26 ms,

delivering a 29.6% reduction relative to CoDel and a 10.2% reduction relative to RED. This latency improvement is obtained with only a modest increase in packet loss. The low delay variance further indicates stable and well-regulated control under peak load. CoDel achieved the lowest packet loss 5.77 ± 0.15 because unlike RED it manages the network based on time rather than capacity. By monitoring how long packet actually stay in the queue, CoDel can react precisely to bad congestion without excessive random dropping that occurs in RED.

Table 4. Testing results

Algorithm	Throughput (Mbps)	Delay (ms)	Loss (%)
DropTail	4.96 ± 0 .004	265.2 ± 3.60	6.00 ± 0.16
CODEL	4.96 ± 0.004	218.4 ± 2.55	5.77 ± 0.15
RED	4.95 ± 0.005	171.0 ± 1.26	6.50 ± 0.16
DQN	4.91 ± 0.001	154.0 ± 0.27	6.83 ± 0.08
DDQN	4.91 ± 0.000	153.7 ± 0.93	6.84 ± 0.08
DDQN-PER	4.91 ± 0.001	153.5 ± 0.26	6.90 ± 0.09

5.3. Queuing latency under out-of-distribution conditions

Table 5 and Figure 7 show that the DDQN-PER agent achieves the lowest mean end-to-end delay across all evaluated load tiers under OOD conditions (10× more severe bottleneck ratio than training). At 60 flows, DDQN-PER maintains 149.8 ± 2.01 ms, representing a 32% reduction versus CoDel (221.1 ms) and a 14% improvement over RED (173.8 ms). Most remarkably, at 120 flows, double the maximum load used to train, DDQN-PER delivers 156.0 ± 2.32 ms, achieving 32% lower delay than CoDel (229.3 ms) and 12% lower than RED (177.4 ms), while simultaneously maintaining high fairness (0.997) and full throughput utilization (4.95 Mbps). At the extreme 180-flow load (3× training maximum), DDQN-PER maintains the lowest delay (157.0 ms) among all methods—33% lower than CoDel (234.4 ms) and 12% lower than RED (179.3 ms).

5.4. Bandwidth equity out-of-distribution conditions

Table 5 and Figure 8 report bandwidth equity index (Jain's fairness index), under OOD conditions. Agents were tested on a 500 Mbps access link with the same bottleneck at 20, 60, 120, and 180 flows. Up to 120 flows, all algorithms sustain near-identical throughput (≈ 4.95–4.96 Mbps) and high fairness. At 120 flows, DRL agents achieve fairness of 0.997 ± 0.004 (DQN), 0.998 ± 0.002 (DDQN), and 0.997 ± 0.003 (DDQN-PER), comparable to or exceeding traditional AQMs. At 180 flows, fairness crashed across all methods. CoDel and RED exhibit complete fairness collapse (0.000 ± 0.000), all three DRL agents show statistically negligible fairness at 180 flows, DQN and DDQN-PER report 0.020 ± 0.141 and DDQN reports 0.000 ± 0.000, differences that fall entirely within the noise margin given the large standard deviation, confirming the collapse is universal and architectural rather than algorithmic. Despite this crash, DRL agents maintain substantially lower delay (157.0–159.4 ms) than traditional AQMs (179.3–270.1 ms), consistently outperforming CoDel.

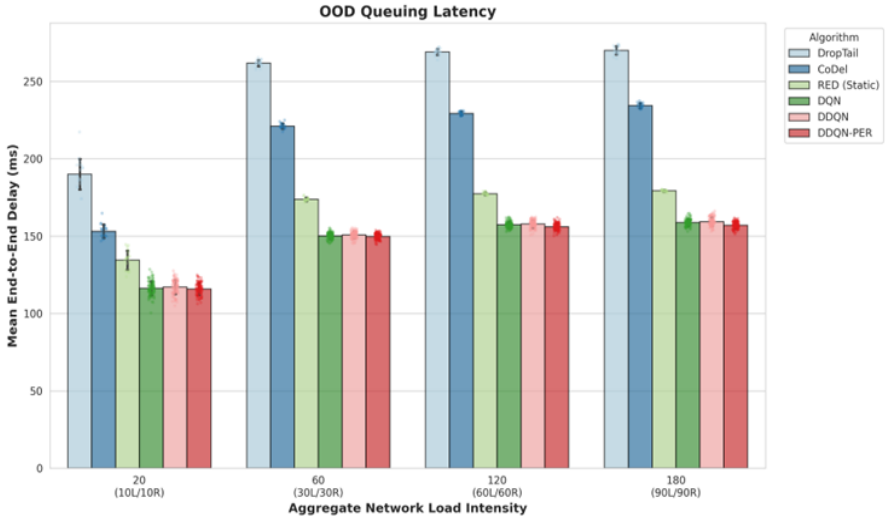

Figure 7. End to end delay plot

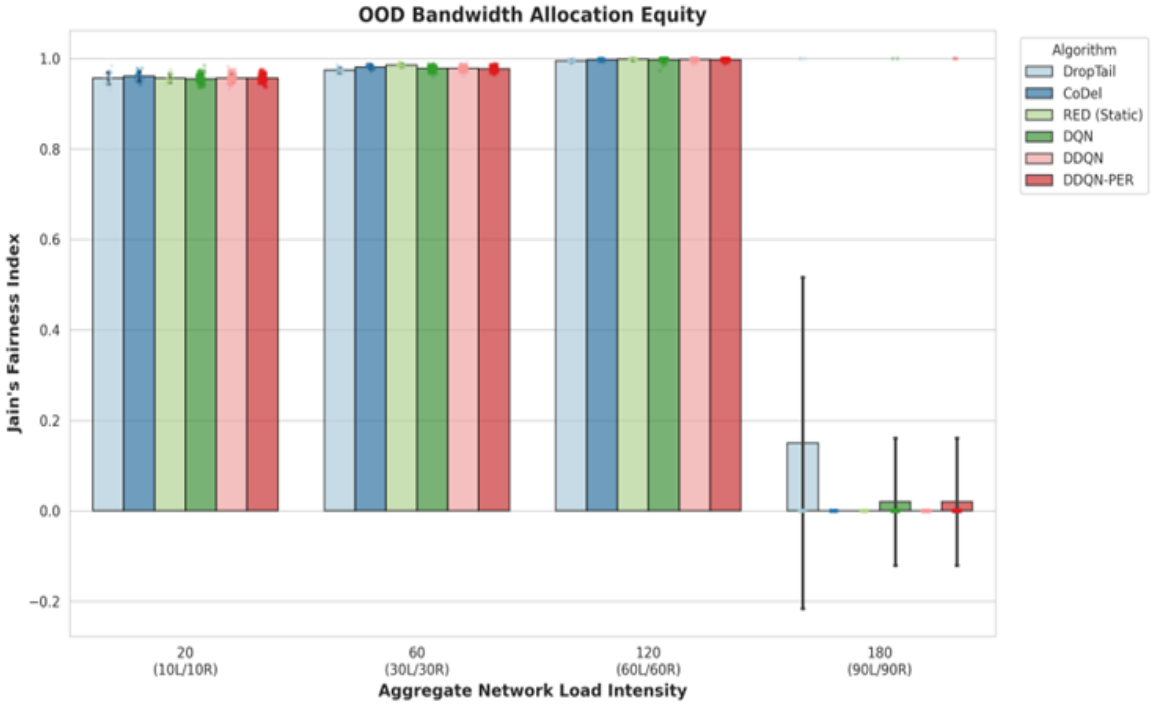

Figure 8. Jains fairness index

This behavior reflects the fundamental trade-off between fairness, delay, and loss under extreme congestion. Under extreme OOD conditions (3× maximum training load, 10× access link speed), DDQN-PER maintained a 33% delay advantage while experiencing a severe fairness crash, a behavior that may be acceptable in latency-critical deployments where delay minimization is paramount and some unfairness can be tolerated to maintain low latency for all flows.

The universality of this collapse across all algorithms, including DropTail, which applies no AQM logic whatsoever, reveals that the root cause is not algorithmic. At 180 flows competing for the 5 Mbps bottleneck, the theoretically fair per-flow bandwidth share is approximately 27.8 Kbps. However, TCP CUBIC enforces a minimum congestion window of 1 maximum segment size (MSS) (1,448 bytes), which at the network RTT of approximately 40 ms yields a minimum achievable sending rate of approximately 289 Kbps, approximately 10× the fair share.

This means that even a maximally backed-off TCP CUBIC flow is physically incapable of transmitting at its fair share rate, regardless of the AQM policy applied. AQM mechanisms can only signal congestion through packet drops or marks; they cannot override TCP's minimum window floor. This also explains the progressive fairness trajectory observed across load tiers, fairness increases monotonically from 20 to 120 flows because in these regimes the per-flow fair share remains above TCP CUBIC's minimum window, allowing AIMD convergence to operate correctly. The transition to 180 flows cross a phase boundary beyond which mathematical fairness is unachievable under TCP CUBIC, irrespective of the AQM policy deployed.

5.5. Packet loss

The packet loss results highlighted in Table 5 and Figure 9 show the deliberate loss–delay trade-off learned by the DRL-based DDQN-PER agent. At lower loads, DDQN-PER incurs moderately higher packet loss than traditional AQMs, but this gap narrows as congestion increases while enabling substantial latency reduction. At 60 flows, there was a slight increase in packet loss (7.43% vs CoDel's 6.13%).

DDQN-PER achieves 12.23 ± 0.21% loss, closely matching RED (12.27 ± 0.13%) and remaining within 0.72 percentage points of CoDel (11.51 ± 0.14%), while maintaining lower delay (156.0 ± 2.32 ms vs. 229.3 ± 1.22 ms for CoDel and 177.4 ± 0.87 ms for RED). At 180 flows, all algorithms converge to similar loss levels (15.57–16.17%) as the network approaches its capacity limit. Despite this convergence, DDQN-PER sustains substantially lower delay (157.0 ± 2.26 ms) than CoDel (234.4 ± 1.57 ms) and DropTail (270.1 ± 2.70 ms). Loss variance remains comparable across methods, indicating stable dropping behavior

under extreme congestion. Overall, DDQN-PER identifies a favorable operating point on loss–delay, accepting small increases in packet loss at lower loads to prevent queue buildup and achieve consistently lower delay.

Table 5. OOD performance results

Load	Algorithm	Throughput (Mbps)	Delay (ms)	Loss (%)	Jain Index
20	DropTail	4.95 ± 0.01	190.0 ± 9.79	1.62 ± 0.12	0.956 ± 0.013
20	CoDel	4.94 ± 0.01	153.2 ± 4.30	1.61 ± 0.10	0.961 ± 0.011
20	RED (Static)	4.87 ± 0.03	134.5 ± 6.16	2.22 ± 0.16	0.956 ± 0.009
20	DQN	4.88 ± 0.03	116.2 ± 4.51	2.37 ± 0.17	0.955 ± 0.010
20	DDQN	4.89 ± 0.02	117.1 ± 4.53	2.39 ± 0.17	0.956 ± 0.009
20	DDQN-PER	4.89 ± 0.03	115.8 ± 3.93	2.38 ± 0.20	0.956 ± 0.010
60	DropTail	4.96 ± 0.00	261.8 ± 2.02	6.74 ± 0.14	0.974 ± 0.006
60	CoDel	4.96 ± 0.00	221.1 ± 1.75	6.13 ± 0.15	0.981 ± 0.005
60	RED (Static)	4.95 ± 0.00	173.8 ± 1.10	6.82 ± 0.13	0.985 ± 0.004
60	DQN	4.95 ± 0.00	150.2 ± 2.02	7.30 ± 0.17	0.978 ± 0.006
60	DDQN	4.95 ± 0.00	150.9 ± 2.20	7.32 ± 0.17	0.978 ± 0.005
60	DDQN-PER	4.95 ± 0.00	149.8 ± 2.01	7.43 ± 0.21	0.977 ± 0.006
		Zero-shot generalization results			
120	DropTail	4.96 ± 0.00	269.1 ± 2.04	11.07 ± 0.14	0.995 ± 0.003
120	CoDel	4.96 ± 0.00	229.3 ± 1.22	11.51 ± 0.14	0.997 ± 0.002
120	RED (Static)	4.96 ± 0.00	177.4 ± 0.87	12.27 ± 0.13	0.998 ± 0.002
120	DQN	4.95 ± 0.01	157.3 ± 2.04	12.16 ± 0.17	0.997 ± 0.004
120	DDQN	4.95 ± 0.01	157.8 ± 2.37	12.16 ± 0.15	0.998 ± 0.002
120	DDQN-PER	4.95 ± 0.01	156.0 ± 2.32	12.23 ± 0.21	0.997 ± 0.003
180	DropTail	4.96 ± 0.00	270.1 ± 2.70	14.28 ± 0.18	0.150 ± 0.366
180	CoDel	4.96 ± 0.00	234.4 ± 1.57	15.35 ± 0.18	0.000 ± 0.000
180	RED (Static)	4.96 ± 0.00	179.3 ± 0.58	16.17 ± 0.12	0.000 ± 0.000
180	DQN	4.95 ± 0.01	158.8 ± 2.25	15.59 ± 0.24	0.020 ± 0.141
180	DDQN	4.95 ± 0.01	159.4 ± 2.79	15.61 ± 0.23	0.000 ± 0.000
180	DDQN-PER	4.95 ± 0.01	157.0 ± 2.26	15.57 ± 0.25	0.020 ± 0.141

Note: all results use OOD conditions (500 Mbps access links vs. 50 Mbps training same topology). Rows 20 and 60 test bottleneck severity generalization with seen flow counts. Rows 120 and 180 (under separator) test both bottleneck severity and flow count generalization

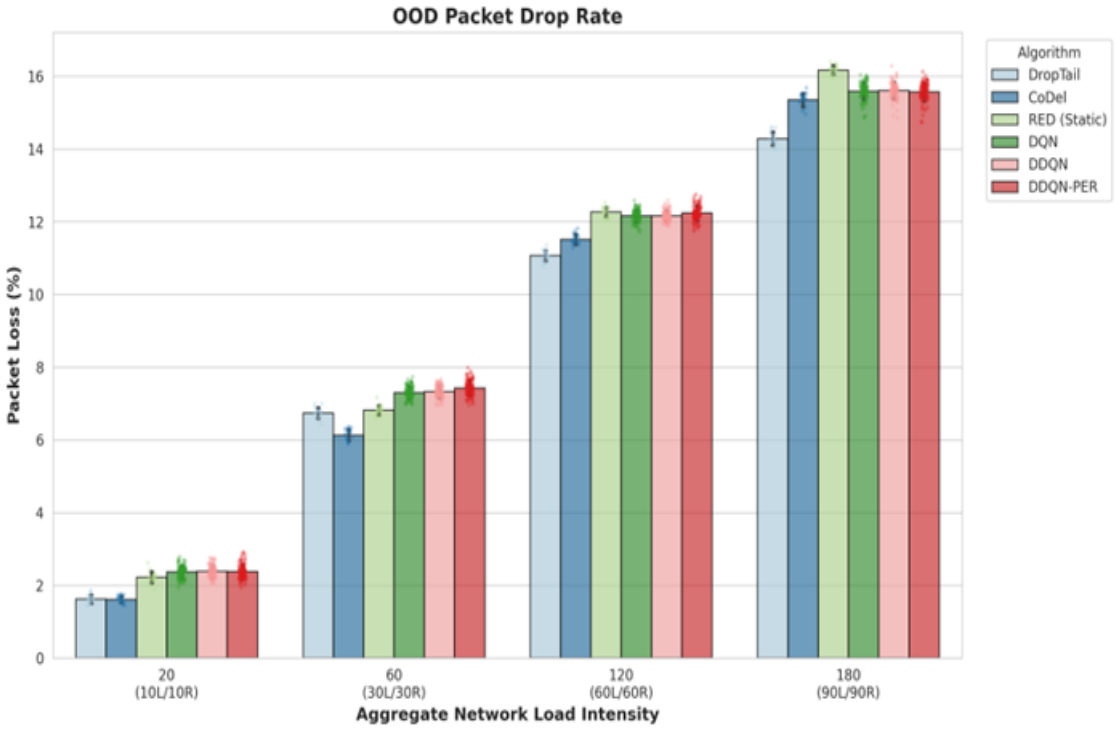

Figure 9. Packet loss

5.6. Effect of exploration rate on performance

Table 6 isolates the impact of the evaluation protocol by comparing DDQN-PER at 60 flows, the only configuration evaluated under both in-distribution testing (ε = 0.05) and OOD stress testing (ε = 0). Despite operating on a topology with a 10× more severe bottleneck ratio (500→5 Mbps vs. 50→5 Mbps), deterministic OOD evaluation achieves lower delay (149.8 ± 2.01 ms) than noisy evaluation on the training topology (153.5 ± 0.26 ms), corresponding to a 3.7 ms (2.4%) reduction.

This result highlights the dominant role of exploration during evaluation. Under ε = 0.05, random action selection disproportionately samples conservative sensitivity values from the logarithmic action space, allowing deeper queue buildup before drops occur and increasing queuing delay. Eliminating exploration (ε = 0) enables consistent selection of aggressive, state-aligned actions, yielding a proactive dropping regime optimized for low-latency.

The latency improvement comes at a modest cost in packet loss: 7.43% under ε = 0 versus 6.90% under ε = 0.05 (+0.5%). This shift reflects a controlled movement along loss–delay, favoring early intelligent drops. From a quality-of-experience perspective, this tradeoff is favorable, as modern transport protocols readily absorb small increases in loss, whereas even a few milliseconds of added delay directly impact interactive applications. Overall, this comparison validates the dual-epsilon evaluation strategy. ε = 0.05 testing assesses robustness under realistic noise, while ε = 0 evaluation reveals the intrinsic generalization capability of the learned policy. The fact that deterministic evaluation on a 10× harder bottleneck severity outperforms noisy evaluation on the training configuration provides strong evidence that the agent learned topology-invariant congestion control principles.

Table 6. Impact of evaluation protocol on 60-flow performance

Context	Algorithm	ε	Throughput (Mbps)	Delay (ms)	Loss (%)
Testing (in distribution)	DDQN-PER (50 Mbps/5 Mbps bottleneck)	0.05	4.91 ± 0.001	153.5 ± 0.26	6.90 ± 0.09
OOD stress test	DDQN-PER (500 Mbps/5 Mbps bottleneck)	0.00	4.95 ± 0.00	149.8 ± 2.01	7.43 ± 0.21

5.7. Learned policy analysis

Figure 10 is the learned w_q policy evolution heatmaps averaged across five independent seeds for the algorithms DQN, DDQN, and DDQN-PER. Figure 10(a) shows DQN's early bias toward w_q = 0.150, Figure 10(b) shows DDQN correcting this and Figure 10(c) shows DDQN-PER's balanced split, averaged across five seeds, revealing that all three agents independently learned to rely almost entirely on two w_q values —the smallest (w_q = 0.0002) and the largest (w_q = 0.150). While this may appear to underutilise the 16-level action space, it is in fact shows that intermediate w_q values offer no practical advantage as they are neither slow enough for precise noise filtering nor fast enough for acute congestion response all three agents correctly learned to discard them through experience. Color intensity reflects normalised action selection frequency across 50 training blocks, where each block represents 20 training episodes.

The key difference lies in how each agent balances these two dominant values. DQN leans too heavily on w_q = 0.150 from early training, reflecting its tendency to overestimate Q-values and respond too aggressively to congestion. DDQN corrects this but swings too far the other way, favouring w_q = 0.0002 and underreacting when congestion demands a stronger response. DDQN-PER strikes the most balanced split between both values consistently across all 50 training blocks, switching appropriately between slow precision filtering and fast reactive dropping as network conditions demand. This progression directly corresponds to the performance improvements reported in Tables 4 and 5. The early training blocks show scattered selections across all action levels, which gradually concentrate into this two-value pattern as training advances, confirming that the behaviour is a product of genuine reward-driven learning across all five seeds rather than a coincidental artefact.

5.8. Computational feasibility and real-time viability

A common aspect of DRL-based network control is that inference overhead may negate the latency benefits achieved through intelligent policy learning. The proposed computational overhead analysis directly addresses this concern as shown in Figure 11. Across all evaluated load intensities (20–180 flows), the mean inference latency remains remarkably stable: DQN averages 606 ± 67 µs, DDQN 560 ± 52 µs, and the proposed DDQN-PER 564 ± 62 µs. While individual operations exhibit occasional outliers reaching 700–800 µs due to OS-level scheduling and cache effects, the mean execution times remain tightly clustered and statistically indistinguishable across the three architectures. These measurements, obtained on a 12th Gen Intel Core i7-1260P CPU, demonstrate that DRL controllers are computationally viable for deployment in modern

edge routers. With sub-millisecond inference at line rate, the DDQN-PER framework introduces no meaningful bottleneck to packet processing pipelines.

5.9. Hyperparameter sensitivity analysis

The canonical hyperparameter values, learning rate η = 0.00025, PER priority exponent α = 0.6, IS annealing start β = 0.4, and replay buffer capacity of 100,000 transitions follow the foundational search of [29], [30]. To validate their transferability to the AQM domain, a domain-specific sensitivity screening was conducted across eight configurations using a one-factor-at-a-time approach [36], varying a single hyperparameter per configuration while holding all others at canonical values. This design isolates the effect of each hyperparameter on performance, so that any change can be attributed to the parameter that was varied.

Four groups were evaluated: learning rate η (C1: 0.0001, C2: 0.010), PER exponent α (C3: 0.4, C4: 0.8), IS annealing start β (C5: 0.6, C6: 0.8), and replay buffer capacity (C7: 50,000 transitions), with C0 as the canonical reference. Each configuration was trained for 200 episodes across three independent seeds, with epsilon decay adjusted to anneal fully within this budget. The 60-flow condition represents the most demanding training load; stability here implies stability at lighter loads. Train, validation, and test phases use RNG pools 1–20, 26–30, and 31–50 respectively, consistent with the main experimental protocol. Results are drawn from the test pool and presented in Table 7.

Throughput remains stable at 4.90 Mbps across all configurations. queuing delay varies within a 2.47 ms range (152.02–154.49 ms). C2 (η = 0.010) exhibits the highest delay standard deviation of ± 3.24 ms, more than double any other configuration, confirming that high learning rates introduce policy instability in the AQM domain and validating η = 0.00025 as the stable operating point. C3 (α = 0.4) produces the highest mean delay of 154.49 ± 0.89 ms, demonstrating that reduced prioritisation weakens focus on high TD-error congestion transitions and validating the canonical α = 0.6. C4 (α = 0.8) produces 152.02 ± 1.19 ms, within the noise margin of canonical, confirming α = 0.6 balances prioritisation strength against overfitting risk. C5 and C6 show negligible β sensitivity (0.14 ms and 0.26 ms deviations), consistent with reduced IS correction influence under dense AQM reward signals. C7 (50,000 buffer) produces 153.34 ± 1.03 ms, 0.88 ms above canonical, confirming the 100,000-transition buffer provides a meaningful advantage within the training budget.

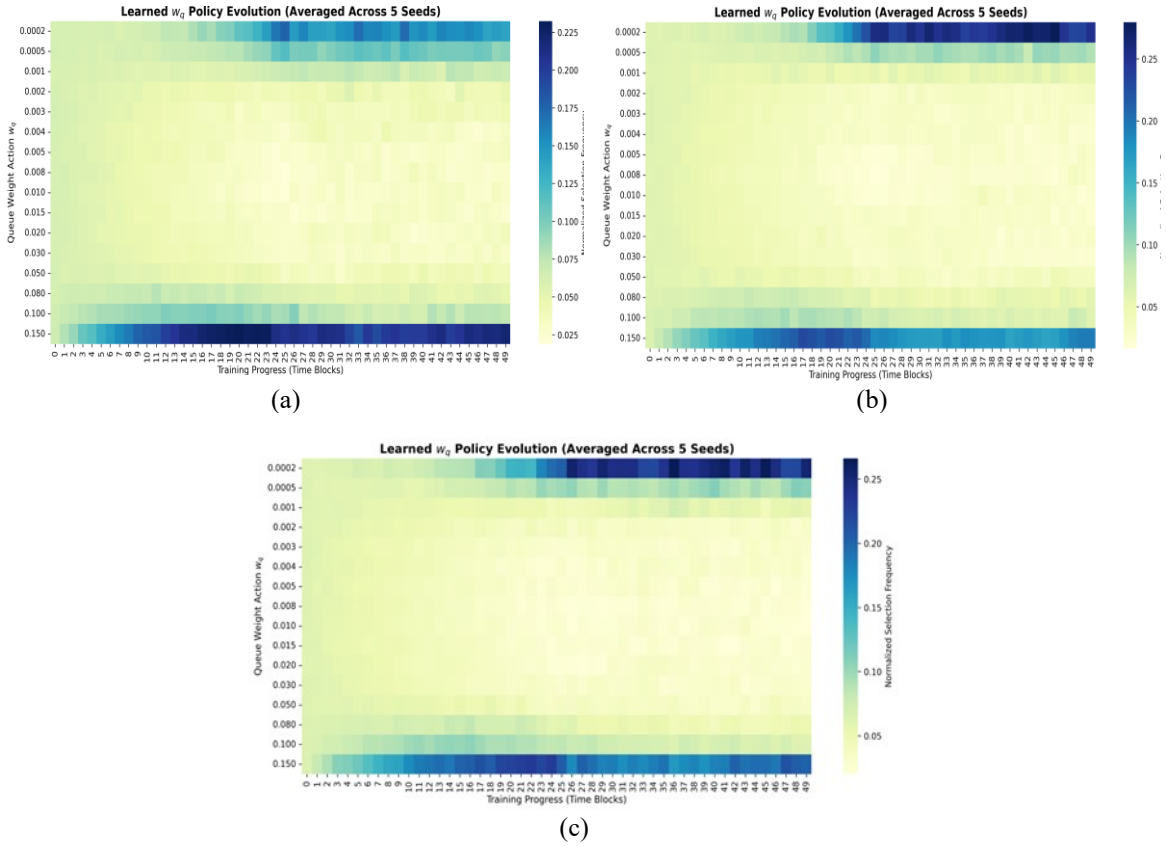

Figure 10. Learned w_q policy evolution heatmaps averaged across five independent seeds for (a) DQN, (b) DDQN, and (c) DDQN-PER

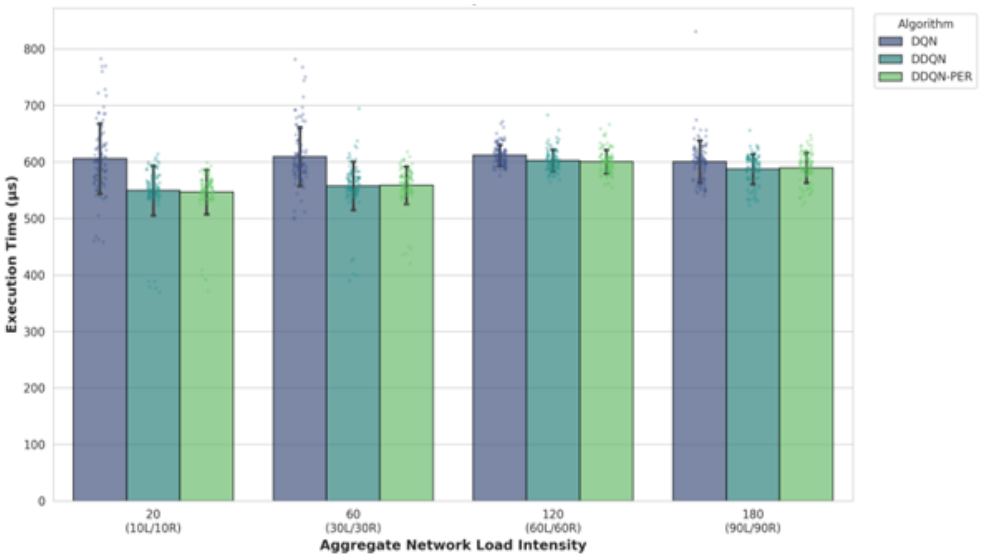

Figure 11. Computational overhead

Table 7. Hyperparameter sensitivity screening results (DDQN-PER, 60 flows, test pool Rngrun 31–50)

Config	lr	alpha	Beta	Buffer	Throughput (Mbps)	Delay(ms)	Packet loss (%)	Advantage over CoDel
C0	0.00025	0.6	0.4	100,000	4.90 ± 0.00	152.46 ± 1.48	6.97 ± 0.11	Yes (30.2%)
C1	0.0001	0.6	0.4	100,000	4.90 ± 0.00	152.62 ± 0.84	6.77 ± 0.05	Yes (30.1%)
C2	0.01	0.6	0.4	100,000	4.90 ± 0.01	153.63 ± 3.24	7.00 ± 0.28	Yes (29.7%)
C3	0.00025	0.4	0.4	100,000	4.90 ± 0.00	154.49 ± 0.89	6.76 ± 0.12	Yes (29.3%)
C4	0.00025	0.8	0.4	100,000	4.90 ± 0.00	152.02 ± 1.19	6.88 ± 0.06	Yes (30.4%)
C5	0.00025	0.6	0.6	100,000	4.90 ± 0.00	152.32 ± 1.18	7.02 ± 0.01	Yes (30.3%)
C6	0.00025	0.6	0.8	100,000	4.90 ± 0.00	152.72 ± 0.91	6.84 ± 0.09	Yes (30.1%)
C7	0.00025	0.6	0.4	50,000	4.90 ± 0.00	153.34 ± 1.03	6.83 ± 0.13	Yes (29.8%)

5.10. System requirements and training infrastructure

Training was conducted on a 12th Gen Intel Core i7-1260P CPU (2.1 GHz, 12 cores, 16 logical processors) with 16 GB RAM. The experiments ran on a Linux virtual machine (Kernel 6.12.20-amd64) in VMware Workstation hosted on Windows 11 25H2. The implementation used Python 3.13.2 with PyTorch 2.6.0+cu124.

5.11. Deployment guidelines

The experimental results of this study provide guidance for network operators. Three scenarios are identified directly from the results: i) scenario 1: DDQN-PER strongly recommended: latency-sensitive single-bottleneck deployments, including industrial automation backhaul, tactile internet gateways, and real-time control system networks operating under TCP CUBIC dominated, ii) scenario 2: DDQN-PER preferred where fairness degradation is tolerable: delay-critical deployments under extreme overload at up to 3× maximum training load, where DDQN-PER retains a 33% delay advantage over CoDel despite universal fairness collapse—a behavior acceptable where delay minimization is paramount, iii) scenario 3: CoDel remains appropriate: deployments requiring zero-configuration plug-and-play operation without ML inference infrastructure, or environments where the operational overhead of maintaining a trained DRL agent is not justifiable relative to the performance margin. CoDel provides adequate delay control with no deployment cost.

6. CONCLUSION

This study demonstrates that dynamically optimizing RED's w_q through value-based DRL is a viable and effective strategy for latency-sensitive network environments. DDQN-PER consistently outperforms CoDel across both in-distribution and zero-shot OOD conditions, sustaining a 29–33% delay advantage with near-perfect fairness up to 2× training load. The universal fairness collapse observed at 3× load reflects a fundamental TCP CUBIC architectural boundary rather than an algorithmic failure—a limitation shared by all evaluated AQMs, including DropTail. Future work should explore deploying the proposed framework on

physical programmable data plane hardware, such as P4-enabled switches or eBPF-instrumented routers, with energy consumption profiling to provide cost-of-ownership insights for large-scale adoption. Evaluating DDQN-PER under diverse traffic profiles including UDP, QUIC, and BBR-driven flows would further increase testing coverage. Extending the framework to multi-agent, multi-bottleneck topologies is the most critical step toward internet-scale deployment, where multiple AQM agents must coordinate across bottleneck nodes to preserve end-to-end latency bounds. Evaluation was conducted on a single dumbbell topology, extension to multi-path and datacenter topologies integrating a per-queue or per-host methodology into DDQN-PER this would create an expanded baseline comparison to include FQ-CoDel, and CAKE providing clearer impact on multi-queue structure queue management. Ultimately, this research demonstrates the promise of DRL in adaptive queue management and lays the groundwork for more robust, fair, and scalable AQM solutions in next-generation networks.

ACKNOWLEDGMENTS

The Authors acknowledge Pastor Kadiri Opaleye for providing the personal financial support to cover the article processing charge for the publication of this manuscript. We thank Covenant University and the Department of Electrical and Information Engineering for providing the institutional environment to conduct this research.

FUNDING INFORMATION

Authors declares no funding involved.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Osemwegie Omoruyi	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Moses Opaleye	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓
Jane Nkwor						✓	✓			✓				
Kennedy Okokpujie	✓	✓		✓	✓	✓	✓			✓	✓			

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject administration
Fu : **Fu**nding acquisition

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

DATA AVAILABILITY

The data that support the findings of this study are available from the corresponding author, [MO], upon reasonable request.

REFERENCES

[1] I. Al Ridhawi, M. Aloqaily, F. Karray, Y. Otoum, and Y. Jararweh, "Realizing the tactile internet through intelligent zero touch networks," *IEEE Network*, vol. 36, no. 4, pp. 278–284, 2022, doi: 10.1109/MNET.001.2200016.
[2] R. W. L. Coutinho and A. Boukerche, "Design of edge computing for 5G-enabled tactile internet-based industrial applications," *IEEE Communications Magazine*, vol. 60, no. 1, pp. 82–88, 2022, doi: 10.1109/MCOM.001.21261.
[3] D. Mourtzis, J. Angelopoulos, and N. Panopoulos, "Smart manufacturing and tactile internet based on 5G in industry 4.0: Challenges, applications and new trends," *Electronics*, vol. 10, no. 24, 2021, doi: 10.3390/electronics10243175.
[4] J. Ansari *et al.*, "Performance of 5G trials for industrial automation," *Electronics*, vol. 11, no. 3, 2022, doi: 10.3390/electronics11030412.
[5] K. A. Tychola, K. Voulgaridis, and T. Lagkas, "Tactile IoT and 5G & beyond schemes as key enabling technologies for the future metaverse," *Telecommunication Systems*, vol. 84, no. 3, pp. 363–385, Nov. 2023, doi: 10.1007/s11235-023-01052-y.

[6] M. U. A. Siddiqui, H. Abumarshoud, L. Bariah, S. Muhaidat, M. A. Imran, and L. Mohjazi, "URLLC in beyond 5G and 6G networks: an interference management perspective," *IEEE Access*, vol. 11, pp. 54639–54663, 2023, doi: 10.1109/ACCESS.2023.3282363.

[7] S. Sunassee, A. Mungur, S. Armoogum, and S. Pudaruth, "A comprehensive review on congestion control techniques in networking," in *Proceedings of the 5th International Conference on Computer Methodologies and Communication (ICCMC)*, 2021, pp. 305–312, doi: 10.1109/ICCMC51019.2021.9418329.

[8] World Population Review, "Internet users by country 2025" 2025. Accessed: Aug. 21, 2025. [Online]. Available: https://worldpopulationreview.com/country-rankings/internet-users-by-country

[9] Statista, "Internet and social media users worldwide 2025" 2025. Accessed: Aug. 21, 2025. [Online]. Available: https://www.statista.com/statistics/617136/digital-population-worldwide/

[10] M. P. Toopchinezhad and M. Ahmadi, "Machine learning approaches for active queue management: a survey, taxonomy, and future directions," *Computer Networks*, vol. 262, 2025, doi: 10.1016/j.comnet.2025.111174.

[11] K. Okokpujie, C. Emmanuel, E. N.-Osaghae, and I. Okokpujie, "Novel active queue management scheme for routers in wireless networks," *International Journal of Communications Antenna and Propagation*, vol. 8, no. 1, pp. 53–61, 2018, doi: 10.15866/irecap.v8i1.13408.

[12] M. Q. Matrood and M. H. Ali, "Adaptive random early detection using deep reinforcement learning for enhanced congestion control in dynamic TCP/IP networks," *Al-Salam Journal of Engineering and Technology*, vol. 4, no. 1, pp. 135–149, 2024, doi: 10.55145/ajest.2025.04.01.013.

[13] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar, "Controlled delay active queue management," 2018, doi: 10.17487/RFC8289.

[14] R. Pan, P. Natarajan, F. Baker, and G. White, "Proportional integral controller enhanced (PIE): a lightweight control scheme to address the bufferbloat problem," *RFC Editor*, Feb. 2017, doi: 10.17487/RFC8033.

[15] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, 1993, doi: 10.1109/90.251892.

[16] Y. Su, L. Huang, and C. Feng, "QRED: a Q-learning-based active queue management scheme," *Journal of Internet Technology*, vol. 19, no. 4, pp. 1169–1178, 2018, doi: 10.3966/160792642018081904019.

[17] F. Jiang, C. Feng, C. Zhu, and Y. Sun, "Performance analysis of an active queue management algorithm based on reinforcement learning," *Journal of European Systems Automation*, vol. 53, no. 5, pp. 637–644, 2020, doi: 10.18280/jesa.530506.

[18] J. E.-S. Shahzad and H. S. Kim, "RLECN---a learning based dynamic threshold control of ECN," *ICT Express*, vol. 9, no. 5, pp. 905–911, 2023, doi: 10.1016/j.icte.2023.10.005.

[19] A. A. Mahawish and H. J. Hassan, "Developed the EX-RED algorithm to find best parameters based on load connection factor," *Iraqi Journal of Computers, Communications, Control and Systems Engineering*, vol. 23, no. 4, pp. 84–94, 2023, doi: 10.33103/uot.ijccce.23.4.8.

[20] W. Wei, H. Gu, and B. Li, "Congestion control: a renaissance with machine learning," *IEEE Network*, vol. 35, no. 4, pp. 262–269, Jul. 2021, doi: 10.1109/MNET.011.2000603.

[21] Y. Wang, L. Wang, and X. Dong, "An intelligent TCP congestion control method based on a deep Q network," *Future Internet*, vol. 13, no. 10, 2021, doi: 10.3390/fi13100261.

[22] H. Fawaz, D. Zeghlache, T. A. Pham, J. Leguay, and P. Medagliani, "Deep reinforcement learning for smart queue management," *Electronic Communications of the EASST*, vol. 80, 2021, doi: 10.14279/TUJ.ECEASST.80.1139.

[23] A. B. Yousif, H. J. Hassan, and G. Muttasher, "Applying reinforcement learning for a random early detection algorithm in adaptive queue management systems," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 26, no. 3, pp. 1684–1691, 2022, doi: 10.11591/ijeecs.v26.i3.pp1684-1691.

[24] D. A. AlWahab, G. Gombos, and S. Laki, "On a deep Q-network-based approach for active queue management," in *Proceedings of the Joint European Conference on Networks and Communications and 6G Summit (EuCNC/6G Summit)*, 2021, pp. 123–128, doi: 10.1109/EuCNC/6GSummit51104.2021.9482588.

[25] K. Kato, H. Kato, H. Asahara, D. Ito, and T. Kousaka, "Effects on random early detection of the packet drop probability function with an adjustable nonlinearity," *Nonlinear Theory and Its Applications, IEICE*, vol. 14, no. 2, pp. 193–206, 2023, doi: 10.1587/nolta.14.193.

[26] A. A. A.-Shareha, M. M. Abualhaj, M. A. Alsharaiah, and Q. Y. Shambour, "A new framework for evaluating random early detection using Markov modulate Bernoulli process stationary distribution," *International Journal of Intelligent Engineering Systems*, vol. 17, no. 4, pp. 790–802, 2024, doi: 10.22266/ijies2024.0831.72.

[27] S. Madipelli, D. R. Gillella, and S. Devaraya, "The RED algorithm–averaged queue weight modeling for non linear traffic," Master of Thesis, Master of Science Program in Electrical Engineering, School of Computing, Blekinge Institute of Technology Blekinge Institute of Technology, Karlskrona, Sweden, 2009.

[28] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.

[29] H. V. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, 2016, pp. 2094–2100.

[30] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[31] J. Liu and D. Wei, "Active queue management based on Q-learning traffic predictor," in *2022 International Conference on Cyber-Physical Social Intelligence (ICCSI)*, Nov. 2022, pp. 399–404, doi: 10.1109/ICCSI55536.2022.9970698.

[32] P. Tam, S. Math, A. Lee, and S. Kim, "Multi-agent deep Q-networks for efficient edge federated learning communications in software-defined IoT," *Computers, Materials & Continua*, vol. 71, no. 2, pp. 3319–3335, 2022, doi: 10.32604/cmc.2022.023215.

[33] P. K. Donta, S. N. Srirama, T. Amgoth, and C. S. R. Annavarapu, "iCoCoA: intelligent congestion control algorithm for CoAP using deep reinforcement learning," *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, no. 3, pp. 2951–2966, Mar. 2023, doi: 10.1007/s12652-023-04534-8.

[34] M. Q. Matrood and M. H. Ali, "Enhancing network congestion control: a comparative study of traditional and AI-enhanced active queue management techniques," *Journal of Cybersecurity and Information Management*, vol. 15, no. 1, pp. 179–196, 2024, doi: 10.54216/JCIM.150114.

[35] P. L'Ecuyer, "Good parameters and implementations for combined multiple recursive random number generators," *Operations Research*, vol. 47, no. 1, pp. 159–164, 1999, doi: 10.1287/opre.47.1.159.

[36] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI-18)*, 2018, pp. 3207–3214, doi: 10.1609/aaai.v32i1.11694.

BIOGRAPHIES OF AUTHORS



Dr. Osemwegie Omoruyi had his Bachelors of Engineering in Computer Engineering from Covenant University, Ota in 2008. He then studied for a Master's of Science degree in Electronic and Electrical Engineering from the Obafemi Awolowo University, Ile-Ife in 2014. He is obtained his doctoral degree from Covenant University in 2022. He currently lectures in Covenant University, Ota, Ogun State, Nigeria. His research interests include reinforcement learning, distributed systems, and artificial intelligence. He is a member of the Nigeria Society of Engineers. He can be contacted at email: osemwegie.omoruyi@covenantuniversity.edu.ng.



Moses Opaleye holds a bachelor of engineering (B.Eng.) in Mechatronics Engineering and is currently a master's student at Covenant University studying Computer Engineering. He is a graduate member of the Nigerian Society of Engineers (NSE). His research interests encompass artificial intelligence, cyber security, and networking. He can be contacted at email: moses.opaleyepgs@stu.cu.edu.ng.



Jane Nkwor had her bachelor's degree in Electrical and Computer Engineering from Lagos State University, Ojo, Lagos State. She is studying for her master's degree in Computer Engineering at Covenant University, Ota, Ogun State. Her research interests include reinforcement learning, artificial intelligence, gaming, and renewable energy. She can be contacted at email: jane.nkwor@covenantuniversity.edu.ng.



Dr. Kennedy Okokpujie holds a Bachelor of Engineering (B.Eng.) in Electrical and Electronics Engineering, a Master of Science (M.Sc.) in Electrical and Electronics Engineering, a Master of Engineering (M.Eng.) in Electronics and Telecommunication Engineering, a Master of Business Administration (MBA), and a Ph.D. in Information and Communication Engineering, besides several professional certificates and skills. He is currently an associate professor in the Department of Electrical and Information Engineering at Covenant University, Ota, Ogun State, Nigeria. He is a member of the Nigerian Society of Engineers and a senior member of the Institute of Electrical and Electronics Engineers (SMIEEE). He is also a research fellow at the Department of Auditing, College of Accounting Science, University of South Africa, Pretoria, South Africa. His research areas of interest include biometrics, artificial intelligence, and network security and management. He can be contacted at email: kennedy.okokpujie@covenantuniversity.edu.ng or kenjie451@gmail.com.