Deep Reinforcement Learning for In-Network Placement of ACL Rules Under Constraints

Wafik Zahwa[†*], Abdelkader Lahmadi[*], Michael Rusinowitch[*], Mondher Ayadi[†]
[*] Université de Lorraine, CNRS, Inria, Loria, F-54000 Nancy, France, {firstname.lastname}@inria.fr
[†] NUMERYX, France, w.zahwa@numeryx.fr

***Abstract*—Efficient distribution of Access Control Lists (ACLs) in modern networks is crucial for ensuring seamless connectivity, robust security, and reliable operations across internal services and hosts. ACLs are typically placed in the switch's Ternary Content-Addressable Memory (TCAM), which is inherently limited in capacity. As communication networks and hosted services expand, the growing complexity and volume of policies require scalable algorithms for effective rule placement. This paper presents a novel approach that combines graph-embedding neural networks (GNN) with deep Q-learning (DQN) to automate optimized ACL distribution across network switches. Our method efficiently manages TCAM utilization while integrating operational constraints and is extensively evaluated on both synthetic and real-world topologies. Results show that it outperforms heuristic and Integer Linear Programming (ILP)-based techniques, offering superior scalability, adaptability, and robustness for ACL rule placement.**

***Index Terms*—ACL, automated rule placement, SDN, graph embedding, reinforcement learning, deep Q-learning, bandwidth constraint, ordering constraint.**

I. Introduction

The increasing complexity, scale, and dynamism of modern networks have led to a rapid growth in Access Control List (ACL) rules [1], [2]. This expansion is driven by rising traffic volumes, evolving security requirements, and the adoption of IoT, BYOD, and cloud services. Application-specific policies, user-based controls, and multitenancy further contribute to ACL proliferation [3], while frequent changes in topology and traffic patterns necessitate constant updates.

Efficient enforcement of these rules requires specialized hardware. Ternary Content Addressable Memory (TCAM) is the dominant choice, as it supports parallel lookups and ternary matching (0, 1, *) needed to encode prefixes, ranges, and priorities. In contrast, SRAM and binary CAM are limited to exact-match operations and lack native prioritization, making them unsuitable for expressive packet filtering. However, TCAM is costly, power-hungry, and capacity-limited: commercial chips typically support only tens of thousands of entries (e.g., 10-20 Mb chips afford 33k IPv6 rules) [4], while large-scale environments such as data centers, cloud-native infrastructures, and 5G/edge systems often require hundreds of thousands. Recent studies confirm that TCAM overflow remains both a critical bottleneck and a vulnerability in production deployments [5].

Although ACL placement and TCAM optimization have been studied since the early SDN generation, the problem remains pressing. High-end TCAM chips (e.g., 144 Mb introduced in 2021 supporting up to 512k entries) provide greater capacity but significantly increase power consumption, sometimes accounting for up to 60% of a switch's budget [6]. The demand for granular policies—including access control, QoS, filtering, and service chaining-continues to intensify this issue. Manual, static ACL management is no longer scalable. Software-Defined Networking (SDN) offers centralized control and dynamic rule distribution, enabling global optimization of TCAM resources, but it requires intelligent algorithms to adapt to changing network conditions.

Recent work has explored splitting and distributing ACLs across multiple switches to reduce per-switch TCAM load. Initial strategies placed rules along single predefined paths [1], [7], which are fragile under path failures or congestion. More robust approaches have proposed multi-path placement over a set of candidate paths [2], [8]. However, these methods assume static routing and fail to generalize under dynamic conditions.

Real-world traffic patterns are typically unbalanced: a small subset of paths carries the majority of traffic due to routing preferences such as shortest-path selection, ECMP (Equal-Cost Multi-Path) balancing, or bandwidth-based decisions. This motivates optimizing rule placement across frequently used paths to avoid redundant replication on underutilized links and switches.

In this work, we propose a learning-based framework for distributed ACL rule placement over a given set of source-destination paths. We couple Deep Q-Learning (DQN) with Graph Neural Networks (GNNs) to learn generalizable strategies that optimize rule placement while satisfying practical constraints. We evaluate our method across synthetic and real-world topologies and compare it to ILP-based and heuristic algorithms under various placement setups and constraints. Our contributions are summarized as follows:

- We propose a novel hybrid GNN-DQN framework for learning optimal ACL rule placements over multiple routing paths.
- We incorporate practical deployment constraints, including rule ordering, bandwidth optimization, and TCAM capacity.
- We extensively evaluate our approach and compare it with ILP and heuristic baselines, highlighting strengths and trade-offs across diverse topologies.

The rest of the paper is structured as follows. Section II provides background on ACLs, RL, and GNNs. Section III discusses related work. Section IV presents the problem formulation and network model. Section V describes our method,

978-3-903176-75-1 ©2025 IFIP

and Section VI outlines the experimental setup. Section VII reports results, and Section VIII concludes.

II. Background

A. Access Control List

ACLs are widely used in enterprise, data center, and WAN networks to enforce security through ordered $\langle match, action \rangle$ rules evaluated top-down, typically with allow/deny semantics [9]. While SDN platforms (e.g., OpenFlow, P4) support advanced actions, this work focuses on traditional allow/deny behavior consistent with firewall semantics. As networks scale, ACL tables rapidly exceed TCAM capacity, motivating strategies such as eviction, compression, and split-and-distribution.Since eviction undermines persistent security policies, split-and-distribution is preferred, partitioning ACLs into subsets constrained by TCAM size and distributing them across switches. Figure 2 illustrates this by dividing the ACL into two sets f_0 and f_1 based on rule actions. The size of each rule set is constrained by TCAM capacity-for example, a 2Mb TCAM stores about 33k 60-bit entries [10]. In this work, we assume ACLs are pre-validated and focus exclusively on optimizing their placement, leaving inconsistency handling and rule generation to future work.

B. Reinforcement Learning

RL models sequential decision-making as a Markov Decision Process (MDP) , where at each step an agent observes a state s_t, selects an action a_t, receives a reward r_{t+1}, and transitions to s_{t+1}. The goal is to learn a policy $\pi : S \to A$ that maximizes the expected return $R(\tau) = \sum_{k=0}^{N} \gamma^k r_{t+k+1}$ over an episode τ, with discount factor γ. RL algorithms refine π through trial-and-error based on observed rewards. In this work, we adopt model-free Deep Q-Learning (DQN) [11], where a neural network $Q(s, a; \theta)$ approximates action-values, enabling scalability to large state spaces.

C. Graph neural network

GNNs learn from graph-structured data by computing node embeddings through iterative message passing. Each node v is initialized with $\mu_v^0 = I(x_v)$ from input features x_v. At layer k, messages from neighbors are aggregated and used to update embeddings: $m_v^{k+1} = M_k(\mu_v^k, \mu_u^k, w_{uv}{}_{u \in N(v)})$, $\mu_v^{k+1} = U_k(\mu_v^k, m_v^{k+1})$. After T layers, a graph-level representation is obtained via a permutation-invariant pooling function (e.g., sum or mean).

III. Related Work

Several split-and-distribution methods have been proposed to tackle the placement problem under different assumptions. Authors in [1] introduce efficient algorithms for rule distribution but are limited to series-parallel network structures. Palette [2] offers a heuristic applicable to arbitrary paths, yet does not handle bandwidth or priority constraints. MaxSMT-based solutions [12], although expressive, lack scalability in dynamic networks due to their recomputation overhead [13]. ILP-based approaches such as Raptor [8] model the problem as a rule minimization task across multi-path networks, with a post-processing heuristic to enforce rule ordering. Authors in [14] extend this model to reduce bandwidth usage by prioritizing rule placement near ingress points, introducing a dual-objective formulation to jointly optimize rule count and placement distance. However, ILP-based methods [8], [14] face two key limitations: (i) scalability: on larger graphs or under additional constraints, the solver frequently exhausts memory and the observed runtime grows sharply (often near-exponentially), leading to practical failures; and (ii) flexibility: incorporating new constraints typically requires expressing them in linear form, otherwise the ILP cannot accommodate the objective or may fail to solve. Our earlier work [15], [16] explored rule distribution across all possible source-destination paths to address routing uncertainty. In this paper, we refine the model by targeting a selected path set and integrating RL to scale across varied topologies and constraints. Recent efforts [17] have explored learning-based approaches and used deep RL for rule eviction optimization. However, many learning-based solutions rely on node-centric actions [18], limiting their suitability for rule placement tasks involving more complex decision spaces. Our method overcomes this by combining deep Q-learning with graph-based encoding to support structured ACL placement over a defined set of paths, enforcing memory, bandwidth, and priority constraints, and generalizing effectively to unseen network topologies.

IV. Preliminaries

This section introduces a motivating example of in-network ACL placement, followed by the problem formulation and its constraints. The notation is summarized in Table I.

Notation	Description
s	single source
t	single destination
P	set of paths
F	set of ACL rules
$m(v)$	memory available on node v, i.e., maximum number of rules that can be installed on v
$\|\|p\|\|$	length of path p, i.e., number of links contained in p (not necessarily st-path)
$paths_v$	subset of paths in P that contain node v
$\|P\|, \|paths_v\|$	number of paths in P, in $\|paths_v\|$
$min(P)$	shortest length of a path in P
P_G	set of all paths between s and t within a graph G
cv	coverage percentage, i.e., ratio of paths included in P (and therefore that should be covered)

TABLE I: Notations for problem formulation

A. Motivating example

Consider the network in Fig.1, with source s, destination t, and intermediate switches a–g (capacity 1 each). The initial ACL table is split into two rule sets, f_0 and f_1 (Tables 2a, 2b), based on their actions. The default action depends on order: the earlier set permits unmatched traffic, while the final set enforces a default deny.

With the basic constraint of covering all paths $P = [s, a, b, d, e, t], [s, c, d, g, t], [s, c, d, g, e, t]$ using the minimal number of rules, f_0 is placed at d and f_1 at e, g. Since f_1

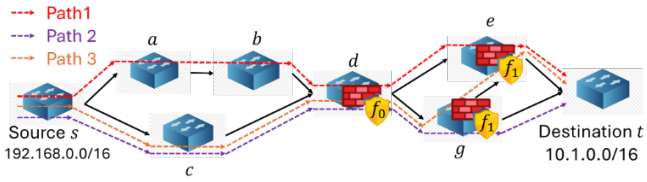


Fig. 1: A placement scenario with topology comprising two rules (f_0, f_1) and three paths: Path 1, Path 2, and Path 3.

includes a more specific deny rule (e.g., blocking 10.1.2.10) than f_0 (allowing 10.1.2.0/24), it must take precedence, leading to swapped placements: f_1 at d and f_0 at e, g. Moreover, f_1 denies ICMP types other than Echo Request, which should be filtered early to save bandwidth. Under this constraint, f_1 moves to a, c while f_0 remains at d. This example illustrates how rule placement adapts to evolving constraints-coverage, ordering, and bandwidth-underscoring the need for flexible, constraint-aware strategies in practice.

Source IP	Source Port	Dest IP	Dest Port	Protocol	Action
192.168.1.10	Any	10.1.20.16	80	TCP	Allow
192.168.0.0/16	Any	10.1.2.0/24	Any	Any	Allow
192.168.1.10	Any	10.1.10.50	Any	ICMP	Allow
192.168.1.10	53	10.1.1.30	Any	UDP	Allow
default rule					

(a) First subtable: f_0

Source IP	Source Port	Dest IP	Dest Port	Protocol	Action
192.168.3.30	Any	10.1.2.10	443	TCP	Deny
192.168.3.30	Any	10.1.3.40	Any	ICMP Type!=8	Deny
default rule					

(b) Second subtable: f_1

Fig. 2: Example of a simplified ACL table divided into two subtables f_0 and f_1 based on the action of the rules.

B. Network representation and problem statement

We model the network as a directed acyclic graph (st-dag) $G(V, E)$ with a single source s and destination t. Nodes represent switches and edges links; any path from s to t is an *st-path*, and all nodes belong to at least one such path.

Each node $v \in V$ has a TCAM capacity $m(v)$, i.e., the maximum number of rules it can store. Let $F = f_1, \ldots, f_k$ be the rule sets (or "rules") and $P = p_1, \ldots, p_f$ the selected st-paths where policies must be enforced. The goal is to place rules on nodes so that every path in P is covered by all rules in F, while minimizing TCAM usage and respecting capacity limits. Redundant placement along the same path is allowed only when unavoidable due to capacity or path overlap, and should be minimized.

Problem formulation: Given $G(V, E)$, memory bounds $m(v)$, rule set F of size k, and path set P, the placement problem $PLACE(G, m, F, P)$ seeks a mapping of rules to nodes such that:

1) **Resource minimization:** total rule placements across all nodes are minimized;
2) **Capacity constraints:** no node hosts more than $m(v)$ rules;
3) **Path coverage:** every path $p \in P$ encounters every rule $f \in F$ at least once.

Let $paths_v$ be the subset of P passing through node v, and P_G the set of all st-paths in G. A path p is covered by rule f if f is placed on at least one of its nodes. The problem is infeasible if the shortest path provides fewer slots than rules, i.e., $\min(P) \times m < k$. Even in simple cases ($m = 1$, $k = 1$), $PLACE$ is NP-hard.

C. Rule placement problem with bandwidth constraint

Minimizing bandwidth wastage is essential for efficient network performance, as bandwidth is consumed along the links between switches. A practical strategy is to place deny rules as close to the source as possible [14], allowing early filtering of unwanted traffic. This prevents unnecessary data from traversing multiple links, reducing congestion and improving overall bandwidth utilization.

D. Rule placement problem with ordering constraint

Rule ordering is critical in ACL placement to ensure that specific or high-priority rules are enforced before general ones, thereby preserving correct filtering and preventing security breaches. For example, a rule blocking 192.168.1.5/32 must precede one allowing 192.168.1.0/24 [1]. In SDN, ACLs may also encode advanced actions such as *forward_to_port*, distinguishing endpoint policies (e.g., access control, QoS) from routing policies (e.g., forwarding). Endpoint policies must take precedence-for instance, a port-blocking security rule should override any routing directive-ensuring both secure and correct operation [8], [19].

V. Deep Reinforcement Learning Approach for Rule Placement

A. Method overview

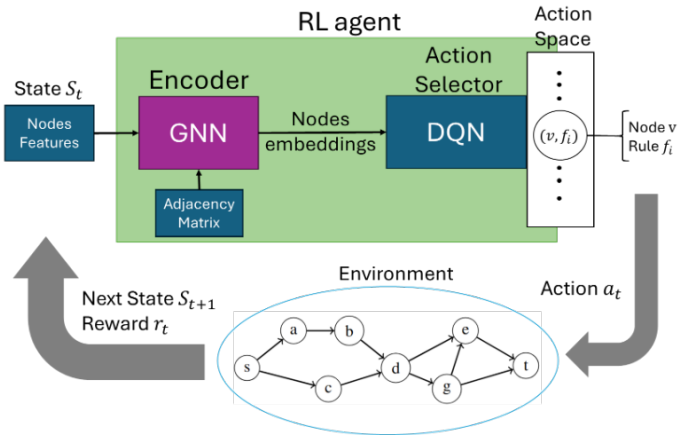


Fig. 3: GNN-DQN architecture dedicated to rule placement.

GNN-DRL design. As shown in Fig.3, the RL environment is the network graph $G(V, E)$. At each time step t, the agent observes a state s_t encoding the topology and current placements,

then selects an action a_t (placing a rule on an uncovered path in P). States are encoded using a Message Passing Neural Network (MPNN) [18], which produces node embeddings $\mu_u^{(T)}$ capturing structural and policy features. These are fed into a DQN that estimates Q-values from global and local information, with the action maximizing $\hat{Q}(s_t, a_t; \Theta)$. After executing a_t, the environment updates and returns a reward r_t, enabling the agent to learn optimized placement strategies.

Role of GNNs. Applying DQN directly to graphs is difficult because input/output dimensions vary with topology. Padding or resizing introduces inconsistencies, since node roles differ across graphs, thereby harming generalization. GNNs overcome this by embedding each node with structural features reflecting its role in the current graph, enabling accurate, scalable, and topology-aware rule placement across diverse instances.

Learning algorithm. Each episode simulates a full interaction cycle between the agent and the environment, starting with a graph sampled from dataset $\mathbb{S}$, initializing features, and proceeding through message passing to compute states. Actions are chosen via ϵ-greedy exploration, while rewards iteratively update Q-values toward the optimal policy. An episode ends when all paths in P are covered ($done_t = True$) or the step limit ($MaxTry$) is reached. Generalization is evaluated by periodically testing the agent on unseen validation graphs every val_freq episodes.

B. Agent state, action, and reward overviews

State: A state encodes the current rule distribution over $G(V, E)$. Each node v is associated with k feature vectors $x_v^1, \ldots, x_v^k$, one per rule f_i, where:

- $x_v^i[1]$ Node degree (number of incident edges);
- $x_v^i[2]$ Boolean: whether f_i is installed on v;
- $x_v^i[3]$ Number of paths through v that include f_i;
- $x_v^i[4]$ Number of paths through v that exclude f_i;
- $x_v^i[5]$ Remaining capacity of v (available memory);
- $x_v^i[6]$ Boolean: whether v belongs to any path in P.

Local features ($x_v^i[2], x_v^i[5], x_v^i[6]$) assess action validity, while global features ($x_v^i[1], x_v^i[3], x_v^i[4]$) guide selection by reflecting structure and episode context.

Action: In our placement problem, an action $a(v, f_i)$ installs rule f_i on node v and is classified as valid, redundant, or invalid. It is redundant if all paths through v already contain f_i, and invalid if v is the source/destination, lacks memory, already hosts f_i, or is not on any path in P. An action is valid when neither redundant nor invalid, i.e., installing f_i on v respects constraints and covers new paths.

Reward: The goal is to cover all paths in P while minimizing memory usage. Installing f_i on v yields:

$$reward(a(v, f_i)) = \begin{cases} -penalty & \text{if } a(v, f_i) \text{ is invalid or redundant,} \\ paths_{v,f_i} - occ & \text{otherwise.} \end{cases}$$

where $paths_{v,f_i} \in [0, 1]$ is the ratio of newly covered paths through v, and $occ \in [0, 1]$ is memory occupancy, i.e., installed rules divided by total capacity (participating nodes $\times$ m).

To handle the bandwidth constraint-placing deny rules close to the source s- the reward for valid actions is extended to: $paths_{v,f_i} - occ - dist_{s,v,f_i}$.
where

$$dist_{s,v,f_i} = \frac{1}{|paths_v|} \sum_{p \in paths_v} \frac{||p_i||}{||p||}$$

and p_i being the prefix of p from s to the first node containing f_i. Since packets are processed at the first switch where f_i appears, only this occurrence is considered. The prefix length $||p_i||$ is normalized by $||p||$, and $dist_{s,v,f_i}$ is the average across all paths in P containing v.

C. Ordering algorithm

To enforce ordering constraints, we keep the same state, action, and reward definitions but adjust validity conditions. For two rules f_i and f_j with f_i prioritized, $a(v, f_i)$ is valid only if f_j is not placed before v on any path in $paths_v$, and $a(v, f_j)$ only if f_i is not placed after v. This induces non-Markovian behavior, since validity depends on prior actions, violating the Markov property. We address this with a two-phase strategy: the DQN agent first solves the base problem, then a post-processing phase adjusts placements to respect priorities.

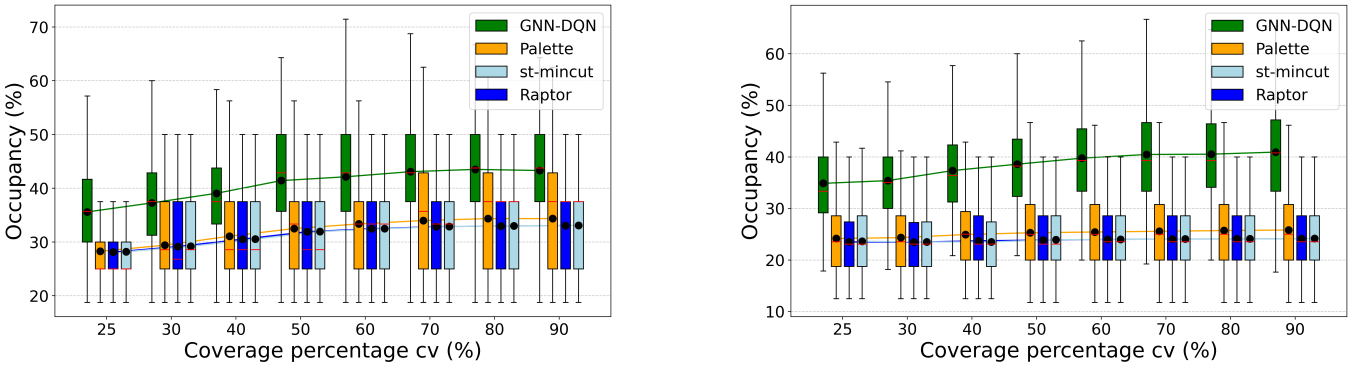
Given a solution S from $PLACE(G, m, F, P)$, we define a pivot set $\mathcal{S}$: the minimal set of nodes containing all placements of a rule. We compute shortest-path distances from s to $\mathcal{S}$ and from $\mathcal{S}$ to t. If $d(\mathcal{S}, t) \times m \geq |F| - 1$, a virtual source s' is connected to $\mathcal{S}$, high-priority rules are placed on the pivot, and DQN handles remaining rules downstream. If $d(s, \mathcal{S}) \times m \geq |F| - 1$, a virtual destination t' is created after the pivot, low-priority rules are placed on $\mathcal{S}$, and DQN handles remaining rules upstream. If neither holds, a larger pivot is selected. This procedure guarantees that rule priorities are preserved whenever the algorithm completes successfully.

VI. Experiments setup

We applied the GNN-DQN algorithm (Algorithm V-A) to the rule placement problem, training on graphs generated by our tool [16] emulating Internet Topology Zoo topologies [20]. Baselines include Palette [2], st-mincut [15], Raptor [8], and Raptor++ for bandwidth constraints [14]. Experiments were implemented in Python and run on a GPU server with an Intel Xeon Gold 6258R, 503GB RAM, and an NVIDIA RTX A6000 (CUDA 12.1) under Ubuntu 22.04.

Performance is evaluated with three metrics: *Occupancy*, percentage of switch memory used; *Success Rate*, valid placements on unseen graphs; and *dist*, which measures the average distance of rules from the source s over all paths in P, extended to the average across k rules. This third metric assesses the solution quality of the placement problem with the bandwidth constraint.

Training uses $\gamma = 0.8$, a learning rate of 10^{-5}, mini-batches of 32, and a -10 penalty for invalid or redundant actions. We run 2 episodes per graph, update the target network every 1000 steps, validate every 50 episodes, and use three layers of message passing ($T = 3$) in the GNN



(a) Occupancy distribution with graphs of size 10

(b) Occupancy distribution with graphs of size 20

Fig. 4: Occupancy distribution of the four approaches across four graph sizes with varying cv.

VII. Results and Discussion

We generated 1,000 training, 200 validation, and 500 testing instances across graph sizes of 10–25 nodes. The coverage ratio $cv \in [0, 1]$ determines the fraction of st-paths selected for policy enforcement ($|P| = |P_G| \times cv$). From this, $|P|$ paths are randomly selected. Models were trained once at $cv = 0.5$ with a fixed number of rules (3 for sizes 10 and 15, 4 for sizes 20 and 25) and evaluated on unseen graphs with varying cv. Due to space limits, we report representative results; additional findings will appear in an extended version.

A. Placement results with underutilized capacity

We evaluated the method on unseen graphs of sizes 10–25 with varying cv from 25–90%. Higher cv values increase the number of selected paths in P, altering the graph structure. The number of rules was fixed to half the slots of the shortest path (3 rules for sizes 10 and 15, and 4 for sizes 20 and 25). Comparisons were made against Palette, Raptor, and st-mincut.

Figures 4a and 4b show occupancy distributions for graph sizes 10 and 20. Occupancy grows with cv since more paths require coverage. Raptor and st-mincut achieve the lowest occupancy, followed by Palette. For GNN-DQN, occupancy is comparable on smaller graphs and tends to be higher on larger graphs, reflecting the increased number of paths and placement decisions. This is expected: heuristics apply predefined strategies that quickly identify effective placements, whereas GNN-DQN relies on exploration and is more sensitive to hyperparameters. Despite this, GNN-DQN generalizes well. Trained once at $cv = 50\%$, it achieved over 95% success across all graph sizes and cv values. Figure 5 shows that GNN-DQN maintains high success rates on larger graphs (e.g., size 25), while Raptor declines with higher coverage. To further assess scalability, we tested graphs of sizes 30 and 40 (30 samples each, $cv = 25\%$). As reported in Table II, Raptor failed, whereas GNN-DQN produced feasible placements.

Conclusion: ILP-based methods fail to scale as graph size increases. In contrast, GNN-DQN, without retraining, adapts to larger graphs and consistently generates valid placements.

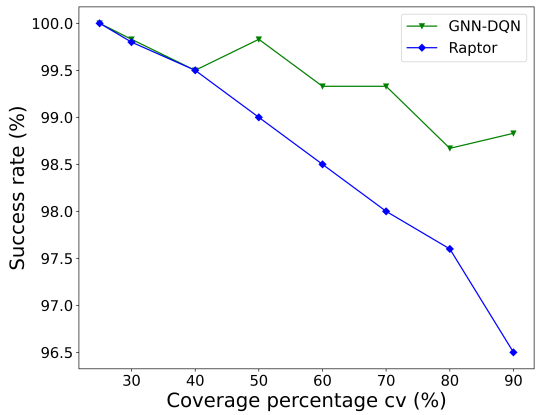


Fig. 5: Success rates of Raptor and GNN-DQN on 500 size-25 graphs with varying cv.

Graph size-rules	Avg. paths	GNN-DQN	Palette	st-mincut	Raptor
30-5	408	74	32	26	-
40-7	930	80.64	20.97	20.97	-

TABLE II: Average occupancy of four algorithms on 30 graphs per size ($cv = 25\%$).

B. Placement results at near maximum capacity

We further evaluated GNN-DQN under stricter constraints by increasing the number of rules to near the available slots in the shortest path ($|F| = (\min(P) \times m) - 1$), leaving little placement flexibility.

Graph size-rules	Avg. paths	GNN-DQN	Palette	st-mincut	Raptor
10-5	9.552	71.79 ± 8.6	61.31 ± 11.15	59.59 ± 10.21	57.25 ± 10.96
15-5	55.892	72.14 ± 8.85	53.82 ± 9.95	50.44 ± 8.26	44.85 ± 8.64
20-7	27.45	76.02 ± 7.26	56.72 ± 9.48	53.7 ± 8.48	47.35 ± 9.06

TABLE III: Average occupancy of four algorithms with $|F| = (\min(P) \times m) - 1$ at $cv = 50\%$.

Table III shows average occupancy at $cv = 50\%$: Raptor performs best, followed by st-mincut and Palette, while GNN-DQN yields higher occupancy due to exploration and problem difficulty. Nevertheless, Fig. 6 shows that GNN-DQN achieves

a higher success rate than Palette, demonstrating robustness under capacity limits.

These results reflect two factors: the sensitivity of GNN-DQN to hyperparameter tuning and the simplicity of the placement problem under a single constraint. Adding rules did not increase problem complexity but required more global and strategic placement. As rule counts approach capacity, algorithms must anticipate future placements to avoid blocking solutions. Palette installs rules incrementally by prioritizing high-coverage junction nodes. However, as these nodes get saturated, it struggles to complete all paths, lowering its success rate. In contrast, GNN-DQN leverages long-term rewards and a global view to learn strategies that better handle such constrained settings and anticipate future limitations.

Conclusion: Heuristic methods perform worse in constrained scenarios requiring global awareness, while GNN-DQN maintains superior success without retraining.

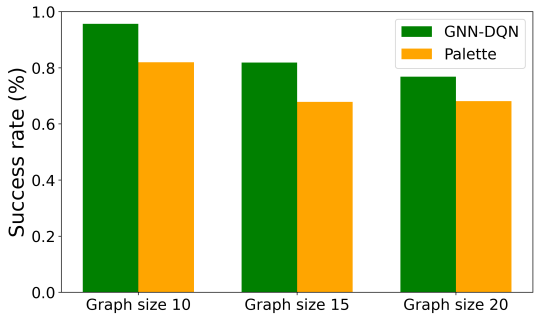


Fig. 6: Success rate of GNN-DQN vs. Palette with $|F| = (\min(P) \times m) - 1$ and $cv = 50\%$.

C. Placement results with a bandwidth constraint

We further evaluated GNN-DQN under a multi-constrained setting by adding the bandwidth constraint that minimizes the distance between the source and rule placements. Four models (graph sizes 10–25) were trained with $|F| = 0.5 \times (\min(P) \times m)$ using the same graphs as before. As deny-rule placement was not fixed, all rules were optimized for proximity to the source. Heuristic methods such as Palette and st-mincut could not be adapted to this constraint, as they are tailored to specific objectives and require redesign to handle new ones, while GNN-DQN and Raptor++ (an ILP-based extension with distance minimization) were evaluated.

	GNN-DQN			Raptor++		
Graph size-rules	Avg. occ(%)	Avg. time(s)	Avg. dist(link)	Avg. Occ(%)	Avg. Time(s)	Avg. dist(link)
10-3	40.65	0.0147	2.24	43.49	0.0241	1.352
15-3	35.76	0.0247	2.51	31.51	0.0898	1.33
20-4	34.27	0.0566	2.95	31.37	0.1549	1.51
25-4	36.24	0.0701	2.96	25.55	0.1231	1.508

TABLE IV: Performance of GNN-DQN and Raptor++ on placement with bandwidth constraint ($cv = 50\%$).

Table IV reports average occupancy, execution time, and distance across 500 unseen graphs at $cv = 50\%$. GNN-DQN was consistently faster than Raptor++, achieved lower occupancy on small graphs, and comparable results on larger ones. Raptor++ obtained shorter distances by greedily filling nodes close to the source. In contrast, GNN-DQN balances multiple objectives via its reward function (see Section V-B), promoting path coverage, occupancy reduction, and proximity to the source. By selecting junction nodes close to the source that cover all paths in P, it optimizes trade-offs among competing goals. Figure 7 shows that GNN-DQN achieves higher success rates than Raptor++, demonstrating scalability and flexibility in handling complex, multi-objective placement problems on unseen topologies, with similar results at $cv = 25\%$.

Conclusion: Heuristics lack flexibility since even minor changes to objectives require redesign. ILP-based approaches can adapt when constraints are linear but suffer from poor scalability. In contrast, GNN-DQN incorporates new objectives directly into the reward function, achieving superior flexibility, scalability, and success rates.

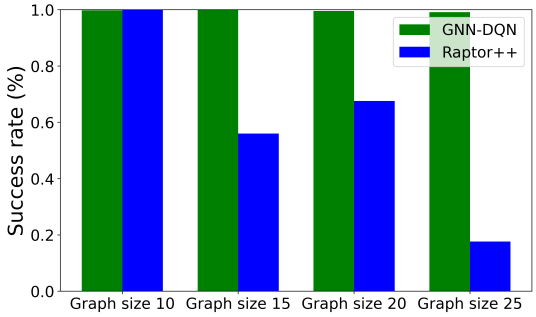


Fig. 7: Success rates of GNN-DQN vs. Raptor++ under bandwidth constraint and $cv = 50\%$.

D. Practical considerations and discussion

The above experiments under various constraints were repeated on five real-world topologies from the Internet Topology Zoo dataset [20], yielding consistent results.

Mininet emulation. We emulated the topology in Fig. 1 using Mininet with a RYU controller pre-installing rules. Each switch had TCAM capacity $m = 2$, and forwarding paths were $P =$ Path 2, Path 3, $[s, a, b, d, g, t]$, $[s, c, d, e, t]$. The initial ACL, derived from an enterprise firewall, was split into five sets: f_1 Ingress Filtering, f_2 Transport/Port Filtering, f_3 Context-Aware Filtering, f_4 Reverse Path Protection, and f_5 Last Switch Enforcement. Custom traffic was generated to match each rule, and evaluation used (i) *path coverage* (percentage of rules on all paths) and (ii) *correctness* (allowed packets delivered, denied packets dropped). Since unmatched packets are dropped by default, correctness is determined solely by false negatives-allowed packets being dropped. Palette placed only four rule sets (f_1:[d], f_2:[d], f_3:[c,g], f_4:[c,g]), yielding 80% coverage and variable correctness (f_1:66%, f_2:56%, f_3:92.7%, f_4:88%, f_5:97.3%). In contrast, GNN-DQN placed all sets (f_1:[d], f_2:[c,e], f_3:[c,g], f_4:[g,e], f_5:[d]), achieving 100% coverage and correctness.

Discussion. This work does not aim to surpass ILP-based or heuristic approaches in solution quality (e.g., rule occupancy), as GNN-DQN is a neural approximator meant to emulate their

Placement with	Underutilized capacity $\lvert F\rvert = 0.5 \times min(P) \times m$			Near maximum capacity $\lvert F\rvert = min(P) \times m - 1$			Bandwidth constraint			Ordering constraint
Graph parameters	size ≤ 15	density ≥ 0.5	size ≥ 20	size ≤ 15	density ≥ 0.5	size ≥ 20	size ≤ 15	density ≥ 0.5	size ≥ 20	
Palette	+	+	+	+	-	-	-	-	-	+
st-mincut	+	+	+	+	-	+	-	-	-	+
Raptor	+	+	-	+	-	-	+	-	-	+
GNN-DQN	+	+	+	+	+	+	+	+	+	+

TABLE V: Summary of algorithm strengths and weaknesses across placement setups and constraints.

behavior; rather, it highlights the distinctive strengths of the GNN-DQN framework across diverse deployment scenarios.

Scalability: GNN-DQN scales to large topologies where ILP-based methods (Raptor, Raptor++) fail due to exponential growth in decision variables.

Generalization ability and robustness: Trained once, the agent generalized to 500 unseen graphs of varying size, density, path configurations, and rule counts, achieving a 95% success rate with 0.14s inference time.

Adaptability and Flexibility: Heuristics such as st-mincut and Palette, limited by greedy local optimization, fail under strict constraints, while GNN-DQN leverages a global view to maintain high success. In multi-constrained settings, heuristics require reformulation and ILPs become infeasible (17% success), whereas GNN-DQN integrates new constraints directly into the reward function, sustaining strong performance (99%). Table V consolidates all results. In summary, while ILP-based approaches suffer from scalability issues, and heuristics lack flexibility, GNN-DQN offers a versatile, generalizable, and robust solution for ACL rule placement in complex, dynamic network environments.

VIII. Conclusions & Future Work

We proposed a GNN-DQN approach for ACL rule placement that combines graph embeddings with deep Q-learning to generalize across diverse topologies. Experiments against Palette [2], Raptor [8], and Raptor++ [14] showed superior scalability, flexibility, and adaptability under constraints such as rule ordering, prioritization, and bandwidth minimization. These results demonstrate the method's suitability for real deployments. As future work, we plan to extend the approach to continuous action spaces to further enhance generalization and scalability.

Acknowledgments

This work is supported by a CIFRE convention between the ANRT (National Association of Research and Technology) and the company NUMERYX Technologies.

References

[1] Ahmad Abboud, Rémi Garcia, Abdelkader Lahmadi, Michaël Rusinowitch, and Adel Bouhoula. Efficient distribution of security policy filtering rules in software defined networks. In *Proceedings IEEE NCA*, 2020.

[2] Yossi Kanizo, David Hay, and Isaac Keslassy. Palette: Distributing tables in software-defined networks. In *Proceedings IEEE INFOCOM*, 2013.

[3] Changhun Jung, Sian Kim, Rhongho Jang, David Mohaisen, and Dae-Hun Nyang. A scalable and dynamic acl system for in-network defense. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022.

[4] Curtis Yu, Cristian Lumezanu, Harsha V Madhyastha, and Guofei Jiang. Characterizing rule compression mechanisms in software-defined networks. In *Passive and Active Measurement (PAM): 17th International Conference,Heraklion, Greece, March 31-April 1*. Springer, 2016.

[5] Ankur Mudgal, Abhishek Verma, Munesh Singh, Kshira Sagar Sahoo, Erik Elmroth, and Monowar Bhuyan. Flora: Flow table low-rate overflow reconnaissance and detection in sdn. *IEEE Transactions on Network and Service Management*, 2024.

[6] Chad R Meiners, Alex X Liu, and Eric Torng. Bit weaving: A non-prefix approach to compressing packet classifiers in TCAMs. *IEEE/ACM Transactions on Networking*, 2011.

[7] Masoud Moshref, Minlan Yu, Abhishek Sharma, and Ramesh Govindan. Scalable rule management for data centers. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, 2013.

[8] Pravein Govindan Kannan, Mun Choon Chan, Richard TB Ma, and Ee-Chien Chang. Raptor: Scalable rule placement over multiple path in software defined networks. In *2017 IFIP Networking Conference (IFIP Networking) and Workshops*. IEEE, 2017.

[9] Shuyuan Zhang, Franjo Ivancic, Cristian Lumezanu, Yifei Yuan, Aarti Gupta, and Sharad Malik. An adaptable rule placement for software-defined networks. In *2014 44th annual IEEE/IFIP international conference on dependable systems and networks*. IEEE, 2014.

[10] Kalapriya Kannan and Subhasis Banerjee. Compact tcam: Flow entry compaction in tcam for power aware sdn. In *International conference on distributed computing and networking*. Springer, 2013.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

[12] Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Fulvio Valenza, and Jalolliddin Yusupov. Automated firewall configuration in virtual networks. *IEEE Transactions on Dependable and Secure Computing*, 2022.

[13] Haifeng Sun, Xingjian Liao, Jingyu Wang, Qi Qi, Zirui Zhuang, Jianxin Liao, and Dapeng Oliver Wu. Fast and scalable acl policy solving under complex constraints with graph neural networks. *IEEE/ACM Transactions on Networking*, 2024.

[14] Yu-Wei Chang and Tsung-Nan Lin. An efficient dynamic rule placement for distributed firewall in SDN. In *IEEE Global Communications Conference*. IEEE, 2020.

[15] Wafik Zahwa, Abdelkader Lahmadi, Michael Rusinowitch, and Mondher Ayadi. Automated placement of in-network ACL rules. In *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*, 2023.

[16] Wafik Zahwa, Abdelkader Lahmadi, Michael Rusinowitch, and Mondher Ayadi. In-network ACL rules placement using deep reinforcement learning. In *2024 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*. IEEE, 2024.

[17] Manuel Jiménez-Lázaro, Javier Berrocal, and Jaime Galán-Jiménez. Deep reinforcement learning based method for the rule placement problem in software-defined networks. In *NOMS, IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2022.

[18] Thomas Barrett, William Clements, Jakob Foerster, and Alex Lvovsky. Exploratory combinatorial optimization with reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[19] Nanxi Kang, Zhenming Liu, Jennifer Rexford, and David Walker. Optimizing the" one big switch" abstraction in software-defined networks. In *Proceedings of the ninth ACM Conference on Emerging Networking Experiments and Technologies*, 2013.

[20] Simon Knight, Hung X Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan. The internet topology zoo. *IEEE Journal on Selected Areas in Communications*, 2011.