A Reinforcement Learning Approach to Virtual Network Embedding Problems in 5G Networks

Amir Javadpour, Forough Ja'fari, Tarik Taleb, Chafika Benzaïd
Pedro R. Tomas, Luis Rosa, Jorge Proença, Luis Cordeiro

***Abstract*—5G network slicing is the problem of mapping requested virtual networks on the substrate network resources. Due to resource capacity constraints, the performance of network slicing depends on the number of supported requests. This challenge is a type of Virtual Network Embedding (VNE) problem in which a weighted graph is divided into multiple smaller weighted graphs according to the user's custom requirements. These problems are NP-hard, and most existing solutions have suggested using Reinforcement Learning (RL) models to solve them. However, they do not adequately represent the weighted graph to the learning model. Therefore, their learning rate is limited. This paper proposes TRL-VNE, a Two-stage RL-based VNE solution to overcome these challenges. In the first stage of this solution, an RL model is utilized for mapping the central node of each request. Novel graph-based features (G-features) are used in this model to improve its learning rate. The second stage uses a greedy algorithm to map the other components. The simulation results show that TRL-VNE improves the requests acceptance ratio and maximum supported requests by 21% and 36%, respectively, compared to existing solutions. Moreover, we have proposed a network architecture based on TRL-VNE, and emulated it in Mininet to investigate the feasibility of the proposed solution.**

***Index Terms*—5G networks, Virtual Network Embedding (VNE), Covering network, Reinforcement learning, Software-Defined Networking (SDN).**

I. Introduction

While the 5G networks have emerged to enhance the quality of service in cellular networks, they still require improvement in their functionality, especially in network slicing. 5G networks provide customised services for users according to their specific requirements [1, 2]. These services are also referred to as *slices*, and dividing a physical network into multiple virtual networks to handle user requests is called *network slicing*. Due to the limited CPU and bandwidth capacities of the resources in a 5G network, network slicing must be performed effectively in order to increase the number of supported requests [3].

The mentioned challenge in network slicing can be modelled as a Virtual Network Embedding (VNE) problem. In a VNE problem, there are several Virtual Network Requests (VNRs), equivalent to the virtual networks, that must be mapped onto a substrate network, comparable to the physical network [4, 5]. If Virtual Node (VN) v is mapped on Substrate Node (SN) s, the CPU capacity of s must be equal to or greater than the required CPU capacity of v. Moreover, if v_1 and v_2 are mapped on s_1 and s_2, respectively, the necessary virtual link bandwidth capacity between v_1 and v_2 must be equal to or less than the total available bandwidth capacity between s_1 and s_2. The solution of a VNE problem specifies how the VNs can be mapped on the SNs, satisfying the mentioned constraints [6].

The VNE problems, which are a subset of the Multiway Separator Problem, are NP-Hard [7], so it cannot be solved in polynomial time. Most of the previous works in the VNE field have utilized a Reinforcement Learning (RL) model to solve such problems. RL is a type of machine learning technique in which there is an agent that explores the problem space, performs some actions, and gets rewards and punishments, based on which, it is led toward the optimal solution [8]. The existing RL models for solving VNE problems have some limitations in formulating the substrate networks. Their main limitation is in representing the graph-based features of the network. Representing all the information in a graph, especially when the network size is large, is not efficient due to the difficulty of the agent exploring a big problem space. If a network is modeled as a weighted graph with N nodes, the agent has to process N^3 values, and even if these values are binary, the agent faces 2^{N^3} different situations. For a network with 100 nodes, the agent faces more than a billion situations, and finding the optimal solution in such a space is impossible. As a result, in this paper, we propose an RL model to effectively model different networks and overcome limitations by defining a new type of features, called G-features, and compressing graph data.

This paper proposes a Two-stage RL-based VNE solution (TRL-VNE) for improving 5G network slicing by optimizing the VNE problems. This solution comprises two stages. In the first stage, the appropriate SN for hosting one of the VNs of a VNR is identified, and then, in the second stage, the other VNs and their links are mapped onto the physical resources. The main contributions of this paper are as follows:

- Introducing the concept of *covering networks*, which compresses the information required for representing a graph, and *G-features* for improving the formulation of weighted graphs.
- Proposing a greedy algorithm for mapping the VNs and their links after fixing the location of one of the VNs.
- Proposing an RL model to efficiently find the optimal VNE solution, incorporating G-features into the training process to enhance performance.
- Emulating the proposed solution in a software-defined network to demonstrate the feasibility of the proposed solution.

To improve the structure and presentation, we present the method and the experiments in a clearer and more consistent order. We describe the proposed framework in the same order as it runs in practice. We first introduce the online VNE setting and constraints, then we explain the covering network and the construction of compact global features (G-features), and next we describe the RL major decision with feasibility masking. After that, we present the greedy minor stage that completes node and link mapping. This order reduces back-and-forth explanations and makes it easier to see how each component contributes to the final embedding decision.

We also reorganize the Results section into three clear parts. We first report overall performance and compare against baselines. We then provide parameter sensitivity analysis to show robustness under key settings. Finally, we provide information-loss analysis to explain how covering compression affects performance and an optimality-oriented indicator. This grouping makes the analysis easier to follow and avoids mixing different types of results in one place.

Figure 1 provides a visual overview of the improved paper flow, including the method pipeline and the result grouping. This figure acts as a quick guide for readers and supports a smoother reading experience.

The remainder of this paper is structured as follows. section II reviews the existing VNE solutions and presents their graph representation techniques. In section III the problem we are solving is defined and the notations are presented. section IV explains the proposed solution and the RL model. section V shows how the proposed solution can be implemented in a real network and section VI reports the evaluation results of this solution. Finally, the conclusion of this paper is presented in section VIII.

II. Related Work

In this section, we have reviewed the existing solutions for VNE problems. It is worth noting that several studies have been conducted in this field. Hence, due to space limitations, we will focus on the most recent ones.

Recent years have witnessed significant progress in RL-based VNE algorithms and scalable substrate representations. Zhang et al. [9] proposed a meta-reinforcement learning approach for adaptive VNE, enabling the agent to adapt to varying network conditions. Liu et al. [10] introduced a federated RL framework for multi-domain SDN-based VNE, addressing scalability and privacy in distributed settings. Graph-based abstraction techniques have

Amir Javadpour is Senior Cybersecurity Researcher MOSA!C Lab / ICTFICIAL Oy, Espoo, Finland. **Forough Ja'fari** is with the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran. **Tarik Taleb** is with the Faculty of Electrical Engineering and Information Technology, Ruhr University Bochum, Bochum, Germany. **Chafika Benzaïd** Faculty of Information Technology and Electrical Engineering, University of Oulu, Finland. **Pedro R. Tomas, Luis Rosa, Jorge Proenc¸a, Luis Cordeiro**, are with OneSource, Coimbra, Portugal.

Corresponding author: Amir Javadpour (a.javadpour87@gmail.com)

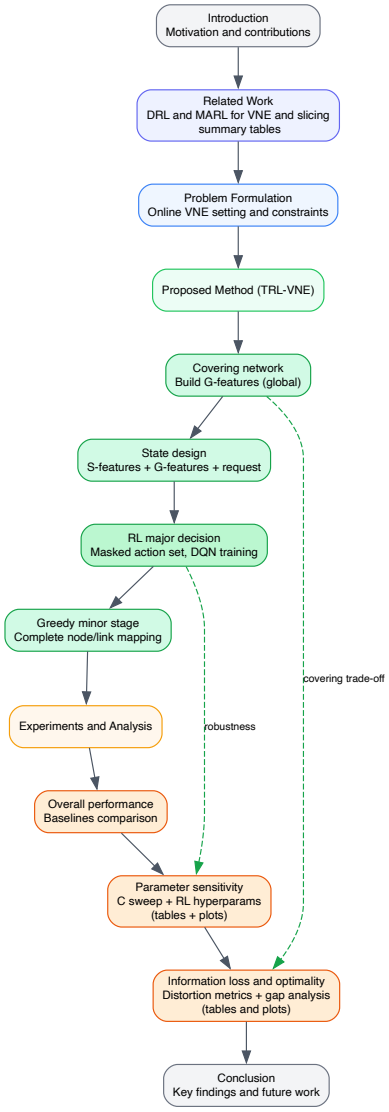

Fig. 1. Paper organization and analysis flow.

also been explored, such as the GNN-based substrate modeling of Chen et al. [11], which improves embedding efficiency by leveraging deep graph representations. Hierarchical RL has shown promise for large-scale VNE scenarios, as demonstrated by Xie et al. [12]. For a comprehensive overview of recent advances, Ye et al. [13] provide an extensive survey on deep RL in network resource management. Compared to these methods, our TRL-VNE framework integrates global graph features (G-features) and a two-stage decision process, offering improved scalability and adaptability while maintaining high embedding performance.

Some researchers have proposed VNE solutions without utilizing a learning model. SPSO-VNE (Set-based Particle Swarm Optimization VNE) [14] first randomly mapped the VNs on the SNs as the initial population, and then used heuristic particle swarm optimization to find the optimal mapping solution. A similar solution-finding process is performed in EE-CTA (Efficient, Concurrent, and Topology-Aware) [15] and DYVINE (DYnamic Virtual Network embedding) [16], utilising a non-dominated sorting genetic algorithm and a novel optimisation algorithm, respectively. In DSCD-VNE (Delay Sensitive Cross Domain VNE) [17], the candidate SN is first selected based on the capacity constraints, and then the Kruskal algorithm, a type of greedy method to find the minimum spanning tree, is used for mapping the links. ODEA (Overlapping Divide-and-conquer Evolutionary Algorithm) [18] divides the virtual networks into sub-graphs, maps them based on a sub-graph mapping procedure, which is a type of evolutionary optimizer, to reach sub-solutions, and finally, integrates the sub-solutions. ELECTRE-VNE [19] utilizes ELECTRE, a type of multi-criteria decision-making method, to balance the process of mapping different requests on the SNs.

TABLE I
THE CONSIDERED S-FEATURES IN THE EXISTING VNE SOLUTIONS OR IN THE SOLUTION OF THIS PAPER.

Label	Description
S1	The remaining CPU capacity of each SN
S2	The number of connected links of each SN
S3	The sum of the remaining bandwidth of each SN
S4	The average distance of each SN to the others
S5	The degree centrality of each SN
S6	The closeness centrality of each SN
S7	The betweenness centrality of each SN
S8	The eigenvector centrality of each SN
S9	The min/max available bandwidth
S10	The min/max CPU capacity of the SNs
S11	The min/max sum of the bandwidth of the SNs
S12	The sequence of mapped VNs on each SN
S13	The sum of adjacent SNs CPU capacity for each SN
S14	Being the end point of the highest bandwidth link

On the other hand, most of the researchers have used RL techniques to enhance VNE. The advantage of RL-based VNE solutions over the others is their learning speed. Hence, in this paper, we focus on RL-based solutions. All RL-based approaches have to answer the same fundamental question before finding the VNE solution, and it is *"How to represent the substrate and virtual networks in order to describe it for the VNE solver model completely"*. Since the substrate and virtual networks are weighted graphs, it is essential to specify the features that must be extracted according to two concepts: (1) the representation space must be kept low to avoid huge time consumption, and (2) the representation has to maximize the obtained data from the graphs. The mentioned research has specified the features that are related to a single node or link. We call these features as *S-features*, and those that are considered in the existing VNE solutions or in the solution of this paper are presented in Table I.

Let us discuss the RL-based research, which is also summarised in Table II by specifying their considered features.

CDRL (Continuous-Decision VNE scheme relying on RL) [20] uses an RL agent to extract the network features, and then passes them to a sequence-to-sequence learning model, which converts a sequence of inputs to a sequence of outputs to find the mapping solution. In DRL-VNE (Double-layer RL VNE) [21], first the VNs are mapped on the SNs based on both local (e.g., degree centrality) and global (e.g., eigenvector centrality) features, and then an RL model is used for mapping the links. A3C+GCN (Asynchronous Advantage Actor-Critic + Graph Convolutional Network) [22] uses graph convolutional networks to extract the features, and then trains an RL model to find the solution. In RDAM (Reinforcement learning based Dynamic Attribute Matrix) [23], the VNs are mapped on the SNs according to the dynamic updates of the substrate network features that are passed to the RL model. DeepViNE (Deep Virtual Network Embedding) [24] encodes the substrate network graph as an image, which helps the RL model to perceive the network features properly. PNVNE (Pointer Network VNE) [25] uses an action mechanism, a neural network technique that improves some parts of the input data to focus on the most suitable SN. A multi-layer VNE problem is considered by MLRL (Multi-Layer RL) [26], where a request is mapped on the SNs and another upper-layer request is mapped on it. An RL model is used to map the VNs, and another auxiliary algorithm is used to reassign the previously mapped VNs to other SNs to improve its real-time performance. Recent advances have established reinforcement learning as a key method for network optimization in SDN and 5G. Early Q-learning models enabled dynamic resource allocation with low overhead, but had limited generalization and slow convergence in complex scenarios [27].

To address these issues, deep RL frameworks introduced constraint-aware training, embedding resource and QoS requirements into the reward function [28]. These models adapt faster and generalize better, though with increased computational costs.

Improvements were made by extending decision-making to the network edge, reducing response latency and enhancing locality-aware orchestration through edge-based reinforcement learning. Actor-critic architectures supported real-time exploration and exploitation, benefiting ultra-low-latency applications. However, decentralization posed challenges in coordinating distributed learners and maintaining state consistency [29, 30].

Multi-agent RL architectures enhance scalability and fairness in hierarchical networks, though agent synchronization remains challenging [31, 32].

Federated reinforcement learning addresses privacy and bandwidth, yet

TABLE II
DIFFERENT RL-BASED VNE SOLUTIONS, THEIR MODEL CHARACTERISTICS, AND FEATURE COVERAGE.

Solution	**Model Summary (RL Type / Architecture / Focus / Security)**	**S-feature Coverage (S1–S14)**	**G-feature**
CDRL [20]	DQN / MLP / Embedding / No Security	S1, S2, S3	✗
DRL-VNE [21]	DDPG / MLP / Embedding / No Security	S1, S3, S5, S6, S7, S8	✗
A3C+GCN [22]	A3C / GCN / Embedding / No Security	S1, S3, S10, S11, S12	✗
RDAM [23]	DQN / MLP / Routing / No Security	S1, S2, S3, S4	✗
DeepViNE [24]	DQN / CNN / Embedding / No Security	S1, S12	✗
PNVNE [25]	PPO / MLP / Placement / No Security	S1, S3, S9	✗
MLRL [26]	DDPG / LSTM / Routing / No Security	S1, S3, S5, S6, S7	✗
TRL-VNE (this paper)	DQN+Transformer / Transformer / Joint VNE / Yes Security	S1, S2, S3, S13, S14	✓

struggles with convergence under non-i.i.d. data [33].

QoS/QoE-aware RL optimizes user satisfaction, requiring extensive labeled data for consistent results under congestion [34, 35].

In non-terrestrial networks with satellites or UAVs, RL algorithms employing probabilistic routing and delay-aware rewards offer robust performance under link failures, but require longer training to stabilize [55].

Security-aware RL for virtual network embedding incorporates trust into the environment, enhancing isolation and adversarial resilience, though at higher computational cost [56, 57].

Recent innovations also include meta-reinforcement learning and dueling deep Q-network enhancements to accommodate preference changes and real-time routing under latency/jitter constraints [51, 52]. Furthermore, knowledge transfer and graph-based learning architectures were developed for dynamic environments and wireless virtualization [53, 54].

Recent work shows RL is becoming more context-aware and distributed. Deep and federated models scale well but face challenges in deployment. Hybrid approaches are robust yet resource-intensive, highlighting ongoing trade-offs.

Another research study adopted a hierarchical deep reinforcement learning model for resource allocation in radio access network slicing. This architecture divides the control logic into upper-layer policies for inter-slice coordination and lower-layer agents for per-user adaptation. The model achieved faster convergence and stronger isolation between slices, providing better quality of service compared to non-hierarchical reinforcement learning frameworks [41]. A recent AutoML-based solution using LazyPredict automated model selection for 5G sub-slice orchestration, reducing configuration overhead and maintaining QoS. However, it showed reduced flexibility in dynamic service chaining scenarios [42].

In cloud-native wireless architectures, deep reinforcement learning was applied to dynamically allocate radio resources, improving spectral efficiency and reducing packet delay. Performance dropped in scenarios with unpredictable user mobility [43].

In Open Radio Access Network systems, multi-agent reinforcement learning was employed for decentralized radio access slicing. The agents used localized observations from the base stations and shared their policies through a lightweight consensus mechanism. This multi-agent approach reduced over-provisioning and improved user fairness metrics compared to centralized decision-making strategies. Coordination among agents, however, posed new challenges in terms of synchronization overhead [44].

To enhance reliability in mission-critical networks, a study introduced a deep reinforcement learning approach that activated backup nodes and rerouted traffic during failures. This restored functionality in 0.3 seconds on average, reducing packet loss, though it required extensive training on fault simulation data and increased runtime computational load [45].

A comparative analysis reveals that hybrid models with attention-based mechanisms are more effective in adapting to user-centric metrics, such as quality of experience. Hierarchical and multi-agent reinforcement learning architectures offer better modularity and scalability, especially in dense and layered deployment environments. Automated orchestration using machine learning pipelines reduces manual tuning but may limit runtime flexibility. Reliability-oriented models are essential for industrial and safety-critical use cases, although they demand high computing power and robust training datasets.

Jeong and Lee proposed a decentralized federated reinforcement learning model to coordinate multiple mobile edge computing nodes without centralized data transfer. This model improved privacy and robustness in dynamic environments and outperformed traditional centralized learning methods in terms of scalability [46].

Shao et al. presented a semantic-aware resource allocation method in vehicle-to-everything heterogeneous networks. Their approach enhanced semantic transmission quality and maintained spectrum efficiency under co-channel interference using a reinforcement learning-based optimization scheme [47].

Researchers at Ericsson developed an intent-driven platform that converts network-level intents into policy actions using a reinforcement learning backend. This framework, known as HYDRA, enabled continuous adaptation of radio access parameters in real time [48].

Another study implemented deep reinforcement learning for massive multiple-input multiple-output non-orthogonal multiple access systems in millimetre-wave communication. The authors applied enhanced clustering, duelling deep Q-networks, and policy gradients to boost convergence and spectral efficiency [49].

In virtual network embedding, a dynamic approach integrated reinforcement learning with graph convolutional neural networks to reduce fragmentation and increase embedding success under dynamic demand [50].

Safe-slicing was introduced to manage mobile network resources with deep reinforcement learning while adhering to service-level agreements. This model optimized orchestration policies in highly variable mobile environments [28].

A security-aware embedding framework was also developed using reinforcement learning to avoid insecure links and improve trust-based resource mapping strategies. Long-term reward improvements and reduced embedding rejections were observed [37].

An innovative management and orchestration model for virtual network functions employed a parameterized action twin deterministic policy gradient approach. The framework autonomously launched, scaled, or migrated VNFs depending on traffic and service metrics [38].

For service chaining, a quality of service-aware deep reinforcement learning model deployed lightweight measurement agents to support chaining decisions and maintain experience quality [34].

Another contribution, ISO-DRLVNE, addressed isolation in virtual network embedding. The model preserved network security by incorporating isolation metrics in its reward function while achieving effective embedding [36].

Finally, in software-defined networking flow management, reinforcement learning was utilized to determine which rules should persist in flow tables. The goal was to reduce unnecessary controller switches and prolong flow-table lifespan [39].

None of these studies have focused on representing the topological features that are not assigned to a single node or link. Hereafter, we call these features as *G-feature*. One of the goals of this paper is to consider G-features for enhancing graph representation.

Table V presents a detailed feature-based comparison of recent RL-based solutions not previously covered in earlier summaries. The comparison leverages a set of node- and link-level substrate features (S1 to S14) and a topological graph-level feature (G-feature), collectively representing the complexity and abstraction of the embedding environment. Each method is assessed on whether it utilizes these features in its learning model or decision process.

Among the evaluated methods, GNN-VNE [50] and Graph-RL Allocation [53] incorporate a diverse set of substrate features, including bandwidth- and CPU-based metrics, and demonstrate notable consideration for global graph structures, enabling improved resource allocation decisions. Conversely, Dueling-DQN Routing [52] emphasises node centrality metrics such as degree and betweenness but lacks broad feature diversity or G-level awareness.

Meta-RL VNE [51] showcases the adaptability of meta-learning in VNE, particularly through its use of a richer feature space and its ability to generalize across unseen request patterns. Adaptive SFC RL [54], although primarily focused on service function chaining, uniquely leverages transfer learning in combination with select features to enable responsiveness in dynamic service contexts.

TABLE III
COMPREHENSIVE FEATURE COMPARISON OF BENCHMARK RL-BASED VNE ALGORITHMS

Solution	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	G-feature
CDRL [20]	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
DRL-VNE [21]	✓	✗	✓	✗	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗
A3C+GCN [22]	✓	✗	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✗	✗	✗
RDAM [23]	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
DeepViNE [24]	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗
PNVNE [25]	✓	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
MLRL [26]	✓	✗	✓	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗
TRL-VNE (this paper)	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓
SafeSlicing [28]	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
ISO-DRL-VNE [36]	✓	✗	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗
Security-Aware VNE [37]	✓	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✓	✗
MANO-VNF-RL [38]	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓
SFC-QoE [34]	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗
RL-FlowMgmt [39]	✗	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗
Hybrid DRL-5G [40]	✓	✗	✓	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗	✗	✓

TABLE IV
EMERGING RL-BASED VNE AND 5G NETWORK MANAGEMENT SOLUTIONS

Solution	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	G-feature
Hierarchical DRL-RAN [41]	✓	✓	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
AutoML Sub-slice [42]	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗
Cloud-Native RL [43]	✓	✗	✓	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✓
O-RAN MARL [44]	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
RL-Reliability [45]	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓
Federated RL 5G [46]	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
Semantic-Aware RL [47]	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
HYDRA (Ericsson) [48]	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓
mmWave-MIMO RL [49]	✓	✗	✓	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✓	✗

TABLE V
RECENT ADVANCED RL-BASED VNE METHODS WITH SPECIALIZED ARCHITECTURES

Solution	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	G-feature
GNN-VNE [50]	✓	✗	✓	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✓
Meta-RL VNE [51]	✓	✓	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓
Dueling-DQN Routing [52]	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
Graph-RL Allocation [53]	✓	✗	✓	✗	✗	✓	✓	✗	✗	✗	✗	✗	✓	✗	✓
Adaptive SFC RL [54]	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓

This comparison reveals that advanced RL techniques increasingly benefit from integrating both localised substrate attributes and abstract graph-level representations to improve decision quality and system scalability.

The above tables (Table III–Table V) provide a comprehensive feature-based comparison of 27 reinforcement learning-based solutions in the context of Virtual Network Embedding (VNE), network slicing, service function chaining, and 5G management. These solutions are evaluated across a consistent set of node-level (S1–S14) and graph-level (G-feature) attributes that significantly influence embedding accuracy and system efficiency.

From the extended comparison in Table III, we observe that traditional solutions such as CDRL [20], DRL-VNE [21], and A3C+GCN [22] primarily focus on fundamental resource metrics (e.g., CPU, bandwidth) and incorporate a limited number of topological features. These approaches often overlook more complex indicators such as closeness, betweenness, and global substrate awareness, which limits their adaptability in dynamic network scenarios.

In contrast, recent methods like TRL-VNE (this paper), SafeSlicing [28], and Hybrid DRL-5G [40] demonstrate improved modeling capability by integrating broader feature spaces. Notably, TRL-VNE introduces G-feature-level representations to encode topological knowledge, yielding better generalization in high-load environments.

Table IV expands this comparison to cover orchestration-aware and slice-specific architectures. Techniques like Hierarchical DRL-RAN [41] and O-RAN MARL [44] leverage decentralized control and G-feature inclusion, reflecting the shift toward more modular and scalable frameworks. Furthermore, federated and semantic-aware approaches (e.g., [46], [47]) offer strong adaptability under privacy and interference constraints, albeit with reduced S-feature usage.

Finally, the focused review in Table V highlights innovative directions such as GNN integration [50], meta-reinforcement learning [51], and transfer-aware SFC chaining [54]. These methods increasingly emphasize graph-structured learning and generalizable decision policies, marking a promising trajectory for future VNE research.

Meta-reinforcement learning has recently been proposed for adaptive VNE, enabling agents to rapidly generalize to new environments [9]. Similarly, federated RL frameworks address scalability and privacy issues in multi-domain network embedding [10]. While TRL-VNE focuses on global topological abstraction and efficient feature compression, meta-RL and federated RL approaches provide complementary benefits in adaptation and distributed learning. An in-depth experimental comparison with these recent methods is left for future work.

Table VI provides a synthesised overview of representative RL-based solutions for VNE and 5G network management, highlighting their learning type, target application, and support for Qos, security, and real-time requirements.

A. Advanced Deep RL and Multi-Agent RL for VNE and Related Virtualization Tasks

Recent studies show that deep reinforcement learning (DRL) is a strong tool for network virtualization problems with sequential decisions, such as virtual network embedding (VNE), service function chain (SFC) embedding, and network slicing resource control. In these problems, an agent observes the network state, takes an action such as selecting a node, allocating bandwidth, or accepting a request, and then receives a reward that reflects long-term performance under constraints. Compared with hand-crafted heuristics, DRL

TABLE VI
REDUCED COMPARISON OF RL-BASED VNE AND 5G NETWORK SOLUTIONS

Solution	RL Type	Application	QoS/QoE	Security	Real-Time	G-feature
CDRL [20]	Q-Learning	VNE	✗	✗	✓	✗
DRL-VNE [21]	DQN	VNE	✗	✗	✗	✗
A3C+GCN [22]	A3C + GCN	VNE	✗	✗	✓	✗
RDAM [23]	Basic RL	VNE	✗	✗	✓	✗
DeepViNE [24]	DQN	VNE	✗	✗	✗	✗
PNVNE [25]	PointerNet	VNE	✗	✗	✓	✗
MLRL [26]	Multi-RL	VNE	✗	✗	✗	✗
SafeSlicing [28]	DDPG	Orchestration	✓	✗	✓	✗
ISO-DRL-VNE [36]	DQN	VNE	✗	✓	✗	✗
Security-Aware VNE [37]	Policy Grad.	VNE	✗	✓	✗	✗
MANO-VNF-RL [38]	DDPG	VNF MANO	✗	✗	✓	✓
SFC-QoE [34]	DQN	SFC	✓	✗	✓	✗
RL-FlowMgmt [39]	DQN+ActorCrit.	SDN Flow Mgmt	✗	✗	✓	✗
Hybrid DRL [40]	DDQN + Attn	Slicing	✓	✗	✓	✓
Hierarchical DRL [41]	Hierarchical RL	RAN Slicing	✓	✗	✓	✓
AutoML Slice [42]	AutoML RL	Sub-slicing	✓	✗	✓	✗
CloudNative RL [43]	DQN	Resource Alloc.	✗	✗	✗	✓
O-RAN MARL [44]	Multi-Agent	RAN	✗	✗	✓	✓
Reliability RL [45]	DRL Backup	Mission Critical	✗	✗	✓	✓
Federated RL [46]	Fed. DRL	5G Core	✗	✗	✗	✓
Semantic RL [47]	PPO	V2X	✓	✗	✓	✓
HYDRA [48]	Intent-RL	RAN Ops	✗	✗	✓	✓
mmWave RL [49]	DQN+DDPG	mmWave	✗	✗	✓	✗
Meta-RL [51]	Meta PPO	Slice Select.	✓	✗	✓	✓
Graph-RL [53]	Graph DRL	Resource Alloc.	✗	✗	✗	✓
Adaptive-SFC [54]	Transfer RL	SFC	✓	✗	✓	✓
GNN-VNE [50]	GCN + RL	VNE	✗	✗	✗	✓

can learn adaptive policies under dynamic traffic and uncertain resources, and it can improve long-term utility when the environment changes over time.

For VNE, many works focus on how to design a good state representation and how to improve generalization across different topologies. Some studies use value-based learning, while others use actor-critic methods, and several papers combine DRL with graph encoders to capture topology structure. Other works extend this direction by improving feature engineering, using constraint-aware rewards, and adopting distributed training or meta-learning to handle practical constraints. However, two common issues remain: the state can be very large when the full substrate graph is used, and the learned policy can be sensitive to representation choices and hyperparameters, which motivates a careful sensitivity and robustness study.

For related tasks, DRL is also widely used in SFC and VNF placement and routing, where the action must consider both compute and link constraints, and the reward must reflect delay and feasibility. In network slicing, multi-agent reinforcement learning (MARL) is common when decisions are distributed across controllers, base stations, or slice managers. Federated MARL is also used to reduce raw data sharing while still learning cooperative policies across multiple agents. These methods often improve scalability, but they may require coordination, careful reward design, and stable training to avoid oscillation and performance drops.

Table VII summarizes representative DRL and MARL works that are close to our scope. In our paper, we follow the same motivation of learning-based control under dynamics, but we reduce state complexity by using compact global features rather than using a full-graph state. This design also supports a clearer analysis of parameter sensitivity and information loss, because the aggregation rules are explicit and measurable.

III. PROBLEM DEFINITION

a) Note on Terminology: Throughout this paper, the terms "actor-critic" and "DQN+Transformer" refer to the same reinforcement learning architecture employed in TRL-VNE. Specifically, TRL-VNE utilizes a DQN-based agent enhanced with a Transformer-based feature extractor. Any previous reference to "actor-critic" in the text or tables should be interpreted as this DQN+Transformer model. This clarification ensures consistency and avoids ambiguity in our algorithm descriptions.

Remark: In this manuscript, any mention of "actor-critic" refers to the DQN+Transformer model used in TRL-VNE.

In a VNE problem, we have a substrate network and multiple virtual networks, the nodes and links of which must be mapped on the substrate network. The virtual networks arrive at the system as user requests.

The notations and functions that are used for modeling the system are summarized in Table VIII and Table IX, respectively.

We can formulate the system as $\mathcal{S} = \{\mathcal{N}, \mathcal{L}, \mathcal{V}\}$, where $\mathcal{N}$ is the list of SNs in the substrate network, $\mathcal{L}$ is the adjacency matrix of the substrate network graph, and $\mathcal{V}$ is the list of user requests (i.e., VNRs) that must be mapped on the substrate network.

$\mathcal{N}$ is written as $\{n_1, n_2, n_3, ..., n_N\}$, where n_i is the i^{th} SN remained CPU capacity. $\mathcal{L}$ is a square $N \times N$ matrix and the element in its i^{th} row and j^{th} column is shown as $l_{(i,j)}$. $l_{(i,j)}$ is the remained bandwidth of the link that connects the i^{th} and j^{th} SNs. Our network graph is undirected. Thus, we can say that $\mathcal{L}$ is a symmetric matrix. $\mathcal{V}$ is shown as $\{v_1, v_2, v_3, ..., v_V\}$, where v_i is the i^{th} VNR, and is written as $v_i = \{n'_i, l'_i, a_i\}$. n'_i and l'_i are the requested CPU and bandwidth of the i^{th} VNR, respectively. To be more precise, n'_i is written as $\{n^i_1, n^i_2, ..., n^i_{N_i}\}$, where N_i is the number of VNs in the i^{th} request and n^i_j is the requested CPU units of its j^{th} VN. l'_i is also a symmetric square matrix and the element in its j^{th} row and k^{th} column is shown as $l^i_{(j,k)}$, which specifies the required bandwidth of the virtual link between the j^{th} and the k^{th} VN of the i^{th} request. a_i is the SN index that the central node of the i^{th} request is mapped on. The central node of a virtual network is defined later in Definition 1. In the case that the mapping solution is not still generated, the value of a_i is zero. It is worth noting that whenever the i^{th} VNR is successfully mapped on the substrate network, a_i and all the related n and l values will be updated. For example, if the CPU capacity of the i^{th} SN is 5 and a VN that requires 3 CPU units is mapped on it, the value of n_i becomes 2.

We now define some functions on the introduced system. $adj(\mathcal{S}, i, j)$ is a binary function, that returns one if the i^{th} and the j^{th} SNs are adjacent in $\mathcal{S}$, and otherwise returns zero. This function is calculated based on Equation 1.

$$adj(\mathcal{S}, i, j) = \begin{cases} 1, & \text{If } l_{(i,j)} \neq 0 \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

$deg(\mathcal{S}, i)$ is the number of the adjacent links (i.e., node degree) of the i^{th} SN, and it is calculated by Equation 2.

$$deg(\mathcal{S}, i) = \sum_{j=1}^{N} adj(\mathcal{S}, i, j) \tag{2}$$

$wdeg(\mathcal{S}, i)$ returns the sum of the links bandwidth of the i^{th} SN. We can say that is the weighted degree of that SN, and the calculation is presented in Equation 3.

$$wdeg(\mathcal{S}, i) = \sum_{j=1}^{N} l_{(i,j)} \tag{3}$$

TABLE VII
REPRESENTATIVE DRL/MARL STUDIES FOR VNE AND RELATED NETWORK VIRTUALIZATION TASKS.

ID	Reference	Problem	RL type	Main idea (one sentence)	Main limitation or gap (one sentence)
A. DRL for Virtual Network Embedding (VNE)					
A1	[58]	VNE survey	Survey	Reviews RL-based VNE methods and key choices for state, action, and reward.	State growth and generalization are still major challenges in RL-based VNE.
A2	[59]	VNE	DRL-based	Uses a region-based representation to guide dynamic VNE decisions efficiently.	Coarse abstraction can hide fine bottlenecks and cross-region coupling.
A3	[60]	VNE	DRL-based	Proposes a DRL-based dynamic VNE method for changing request streams.	Robustness across different graphs can require careful state design and tuning.
A4	[61]	VNE	DRL + GCN	Uses graph convolution to encode topology features for the RL policy.	Full topology encoding can be expensive and sensitive to graph size.
A5	[62]	VNE	Deep RL	Learns VNE decisions with deep RL to improve long-term embedding quality.	Training can be sample-hungry and unstable without strong state and reward design.
A6	[63]	VNE (datacenters)	Learning-assisted	Uses node essentiality and collaborative embedding for data-center settings.	Collaboration introduces overhead and depends on the quality of essentiality signals.
A7	[64]	VNE-like allocation	DRL framework	Proposes a flexible RL framework that can generalize across allocation settings.	Generalization still depends on reward design and informative features.
A8	[65]	VNE (energy-aware)	DRL-based	Adds energy considerations to VNE decisions in the RL reward.	Reward weights can affect stability and may need sensitivity checks.
A9	[66]	VNEP constraints	Meta-RL	Uses meta-learning to adapt VNE policies under uncertain constraints.	Meta-training can be heavy and depends on diverse training tasks.
A10	[67]	VNE	RL-based	Uses continuous decisions to better match resource allocation granularity.	Continuous actions can increase training complexity and need normalization.
A11	[68]	Multi-layer VNE	RL-based	Designs an RL method for dynamic multi-layer embedding with SDN support.	Multi-layer settings enlarge the decision space and complicate convergence.
B. DRL for SFC/VNF Placement and Routing (related embedding tasks)					
B1	[69]	VNF placement + routing	DRL (attention)	Uses attention to improve placement and routing decisions jointly.	Large states still reduce training efficiency if inputs are not compact.
B2	[70]	SFC embedding	DRL	Uses DRL to improve long-term SFC embedding under dynamics.	State abstraction may introduce information loss that affects optimality.
B3	[71]	SFC provisioning	DRL	Proposes a DRL framework for provisioning under dynamic demands.	Performance can depend on hyperparameters and reward weights.
C. DRL/MARL for Network Slicing, Admission Control, and Resource Allocation (similar applications)					
C1	[72]	Slicing + admission	MARL	Jointly solves slicing and admission via multi-agent deep RL.	Coordination can be complex and sensitive to reward coupling.
C2	[73]	Slicing allocation	DRL	Uses DRL for online resource allocation under time-varying demands.	Policies may need adaptation when traffic patterns shift strongly.
C3	[74]	IoT slicing	Federated MARL	Uses federated multi-agent RL to optimize QoS across network slices.	Federated training may converge slower with heterogeneous clients.
C4	[75]	SAGIN slicing	MARL	Uses MARL for priority-based cross-layer load balancing.	Cross-layer control enlarges the action space and needs stability analysis.
C5	[76]	Slice admission	DRL (actor-critic)	Uses a twin actor-critic style method to improve admission decisions.	Actor-critic methods can be sensitive to tuning and exploration.
C6	[77]	RAN slicing	DRL	Uses DRL to allocate radio resources among slices.	QoS constraints require careful reward calibration and constraint handling.

TABLE VIII
THE NOTATIONS USED FOR MODELING THE VNE PROBLEM.

Symbol	Description
$\mathcal{S}$	The system of substrate network and arrived VNRs
$\mathcal{N}$	The list of SNs CPU capacity
N	The number of nodes in the substrate network
$\mathcal{L}$	The adjacency matrix of the substrate network
$\mathcal{V}$	The list of arrived VNRs
v_i	The i^{th} VNR
n_i	The remained CPU capacity of the i^{th} SN
n'_i	The list of requested CPU capacity of the i^{th} VNR
n^i_j	The requested CPU of the j^{th} VN of the i^{th} VNR
$l_{(i,j)}$	The remained bandwidth between the i^{th} & the j^{th} SNs
l'_i	The requested bandwidth between the VNs of the i^{th} VNR
$l^i_{(j,k)}$	The requested bandwidth between the j^{th} & the k^{th} VN of the i^{th} VNR
a_i	The SN hosting the central node of the i^{th} VNR
C	The number of covering network nodes

$max(\mathcal{S}, i)$ is a binary function that indicates whether the i^{th} SN is one of the end-points of the link with the highest bandwidth or not, by returning one or zero, respectively. The $max()$ calculation is presented in Equation 4.

$$max(\mathcal{S}, i) = \begin{cases} 1, & \text{If } \exists j,\ \forall b, c \leq N : l_{(i,j)} \geq l_{(b,c)} \\ 0, & \text{Otherwise} \end{cases} \tag{4}$$

$min(\mathcal{S}, p)$ gets a list of SNs as a path, say p, and returns the minimum remaining link bandwidth in this path. This function is calculated using Equation 5, where p_i is the i^{th} SN in the input path (i.e. p).

$$min(\mathcal{S}, p) = \min_{i<|p|} l_{(p_i, p_{i+1})} \tag{5}$$

$path(\mathcal{S}, i, j)$ returns the list of all the shortest unweighted paths from the i^{th} SN to the j^{th} SN. The algorithm for finding the output of this function is shown in Algorithm 1, through which a recursive function, named $find\_paths()$ and described in Algorithm 2, is called to find all the shortest paths recursively. It is worth noting that the time complexity of this algorithm is $O(N^2)$. The $find\_paths()$ function considers the parents of each node and creates the path in a backward way. $mpath(\mathcal{S}, i, j)$ is the shortest unweighted path from i to j, where the minimum weight of its links is higher than that of the other shortest paths. In other words, $mpath()$ returns the path with the highest $min()$ among the paths generated by $path()$. We can define $mpath()$ as Equation 6.

$$mpath(\mathcal{S}, i, j) = \underset{p \in path(\mathcal{S}, i, j)}{\operatorname{argmax}} \ min(\mathcal{S}, p) \tag{6}$$

$len(\mathcal{S}, i, j)$ returns the number of links in the shortest path between the i^{th} and the j^{th} SNs, and hence, it is calculated by Equation 7.

$$len(\mathcal{S}, i, j) = \mid mpath(\mathcal{S}, i, j) \mid \tag{7}$$

$wlen(\mathcal{S}, i, j)$ is the weighted length of $mpath(\mathcal{S}, i, j)$, and it is calculated using Equation 8, where $mpath$ is this path, and $mpath_k$ is the k^{th} node in this path.

$$wlen(\mathcal{S}, i, j) = \sum_{k=1}^{k=|path|-1} l_{(path_k, path_{k+1})} \tag{8}$$

$nei(\mathcal{S}, i, j)$ returns the list of SNs that are sorted by their unweighted shortest path length to the i^{th} SN. The SNs in this list are sorted by their $wlen()$ value and then by their CPU capacity. So, this function can be calculated as Algorithm 3. $wnei(\mathcal{S}, i)$ is the sum of CPU capacities of the neighbor SNs

TABLE IX
THE FUNCTIONS DEFINED IN THE VNE PROBLEM MODEL.

Function	Output	Reference
$adj(\mathcal{S}, i, j)$	A binary number that is one if the i^{th} and j^{th} SNs are connected	Equation 1
$deg(\mathcal{S}, i)$	The number of adjacent links of the i^{th} SN	Equation 2
$wdeg(\mathcal{S}, i)$	The sum of the adjacent links bandwidth of the i^{th} SN	Equation 3
$max(\mathcal{S}, i)$	A binary number that is one if the link with the highest bandwidth is the i^{th} SN adjacent	Equation 4
$min(\mathcal{S}, p)$	The minimum remained link bandwidth in the input path, p	Equation 5
$path(\mathcal{S}, i, j)$	The list of all the shortest unweighted path between the i^{th} and j^{th} SNs	Algorithm 1
$mpath(\mathcal{S}, i, j)$	The shortest unweighted path between the i^{th} and j^{th} SNs with the highest $min()$	Equation 6
$len(\mathcal{S}, i, j)$	The number of links in the shortest unweighted path between the i^{th} and j^{th} SNs	Equation 7
$wlen(\mathcal{S}, i, j)$	The sum of links bandwidth in the shortest unweighted path between the i^{th} and the j^{th} SNs with the highest $min()$	Equation 8
$nei(\mathcal{S}, i)$	The sorted list of SNs by their shortest path length to the i^{th} SN, their $wlen()$, and their CPU capacity	Algorithm 3
$wnei(\mathcal{S}, i)$	The sum of CPU capacities of the adjacent SNs of the i^{th} SN	Equation 9

Algorithm 1 The procedure of calculating $path()$

Require: $\mathcal{S}$, the system model parameters
Require: i, the source SN
Require: j, the destination SN
Ensure: $paths$, the output of the $path()$ function
  $dist \leftarrow$ a list of N ∞s
  $parents \leftarrow$ a list of N empty lists
  $dist[i] \leftarrow 0$
  $checked \leftarrow \{i\}$
  while $|\ checked\ | > 0$ **do**
    $current \leftarrow checked[1]$
    remove $current$ from $checked$
    for $1 \leq k \leq N$ **do**
      if $adj(\mathcal{S}, k, current) = 0$ **then continue**
      if $dist[k] > dist[current] + 1$ **then**
        $dist[k] \leftarrow dist[current] + 1$
        $parents[k] \leftarrow \{current\}$
        add k to $checked$
      else if $dist[k] = dist[current] + 1$ **then**
        add $current$ to $parents[k]$
  $paths \leftarrow$ an empty list
  $p \leftarrow$ an empty list
  call $find\_paths(\mathcal{S}, parents, j, paths, p)$ ▷ Algorithm 2
  return $paths$

Algorithm 2 The procedure of recursively finding all the shortest paths

Require: $\mathcal{S}$, the system model parameters
Require: $parents$, the list of all nodes' parent nodes
Require: $node$, the current node in the path
Require: $paths$, the pointer to the list of all paths
Require: p, the pointer to the currently generated path
  if $|\ parents[node]\ | = 0$ **then**
    add p to $paths$
  else
    for $par \in parents[node]$ **do**
      add $node$ to p
      call $find\_paths(\mathcal{S}, parents, par, paths, p)$
      remove the last element of p

of the i^{th} SN. This function is calculated using Equation 9.

$$wnei(\mathcal{S}, i) = \sum_{j \leq N \text{ and } l_{(i,j)} \neq 0} n_j \tag{9}$$

Now, we can define the central VN of a request. The central node of a virtual network is the VN with the highest degree. If two or more VNs have the same degree, the one with the highest weighted degree is considered the central node. If two or more nodes have the same condition again, the node with the highest weight is the central node. If the conditions remain the same, one of the candidate VNs is arbitrarily selected as the central node. The precise definition of central VN is presented in Definition 1.

Algorithm 3 The procedure of calculating $nei()$

Require: $\mathcal{S}$, the system model parameters
Require: i, the input SN index
Ensure: nei, the output of the $nei()$ function
  $checked \leftarrow \{i\}$
  $levels \leftarrow$ a list of N zeros
  $nei \leftarrow$ an empty list
  while $|\ checked\ | > 0$ **do**
    $current \leftarrow checked[1]$
    add $current$ to nei
    remove $current$ from $checked$
    $l \leftarrow l + 1$
    for $1 \leq j \leq N$ **do**
      if $adj(\mathcal{S}, j, current) = 0$ **then continue**
      add k to $checked$
      $levels[current] \leftarrow levels[current] + 1$
  for $1 \leq j \leq N$ **do**
    for $j < k \leq N$ **do**
      if $levels[j] \neq levels[k]$ **then break**
      $len1 \leftarrow wlen(\mathcal{S}, i, j)$
      $len2 \leftarrow wlen(\mathcal{S}, i, k)$
      if $len1 < len2$ **then**
        $temp \leftarrow nei[j]$
        $nei[j] \leftarrow nei[k]$
        $nei[k] \leftarrow temp$
      else if $len1 = len2$ **then**
        if $n_j < n_k$ **then**
          $temp \leftarrow nei[j]$
          $nei[j] \leftarrow nei[k]$
          $nei[k] \leftarrow temp$
  return nei

Definition 1 (Central VN)**.** *The central VN of a VNR is the node with the highest degree, with the highest weighted degree if the degrees are the same, with the highest weight, if the degrees and the weighted degrees are the same, and otherwise, a random VN. In other words, if the central VN of the i^{th} VNR is shown as c_i, the following conditions are true, where $\mathbb{N}_i$ is the set of natural numbers not greater than i.*

- $\nexists j \in \mathbb{N}_i : deg(v_i, j) > deg(v_i, c_i)$
- $\nexists j \in \mathbb{N}_i : deg(v_i, j) = deg(v_i, c_i)$ *and* $wdeg(v_i, j) > wdeg(v_i, c_i)$
- $\nexists j \in \mathbb{N}_i : deg(v_i, j) = deg(v_i, c_i)$ *and* $wdeg(v_i, j) = wdeg(v_i, c_i)$ *and* $n'^{i}_{j} > n'^{i}_{c_i}$

This paper aims to address the primary challenge of mapping network slices to physical resources, thereby supporting the maximum number of slice requests. The network slices are treated as virtual networks, which must be mapped onto the substrate network, and this is a type of VNE problem. In this context, we have three constraints that must be considered for having a successful mapping:

- A VN cannot be mapped on an SN with enough remaining CPU capacity to support it. In other words, if the j^{th} VN of the i^{th} request is mapped on the q^{th} SN, we have $n^i_j \leq n_q$.
- A virtual link cannot be mapped on a substrate link that does not have enough remaining bandwidth to support it. To be more general, if the j^{th} and the k^{th} VNs of the i^{th} request are mapped on the q^{th} and

Algorithm 4 The procedure of major mapping stage in TRL-VNE

Require: $\mathcal{S}$, the system model parameters
Require: v, the index of the VNR to be mapped
Require: s, the chosen SN to be the host of the central VN
Ensure: $central$, the index of the central VN
$central \leftarrow 1$
for $1 < i \leq N_v$ **do**
  if $deg(v_v, central) < deg(v_v, i)$ **then**
    $central \leftarrow i$
  else if $deg(v_v, central) = deg(v_v, i)$ **then**
    if $wdeg(v_v, central) < wdeg(v_v, i)$ **then**
      $central \leftarrow i$
    else if $wdeg(v_v, central) = wdeg(v_v, i)$ **then**
      if $n^i_{central} < n^v_i$ **then**
        $central \leftarrow i$
if $n_s < n^v_{central}$ **then**
  return 0
else
  return $central$

the r^{th} SNs, respectively, there must be at least one path from the q^{th} SN to the r^{th} SN, in which all the links have at least $l^i_{(j,k)}$ remained bandwidth.

- Two VNs of the same request cannot be mapped on a single SN. Hence, if the j^{th} and the k^{th} VNs of the i^{th} request are mapped on the q^{th} and the r^{th} SNs, we have $q \neq r$.

IV. Proposed Method (TRL-VNE)

Mapping VNRs on a substrate network is an NP-hard problem. To address this, we propose TRL-VNE (Two-stage RL-based VNE), which consists of two stages. In the first stage (major mapping), the virtual network is analyzed, and the central VN is mapped to an SN using an RL model. In the second stage (minor mapping), a greedy algorithm maps the remaining VNs and links. These stages are repeated for each VNR arrival. The following section details these stages and the RL model.

A. Major mapping stage

The central VN is mapped on one of the SNs in the major mapping stage. But before that, the request is processed to find its central VN. The central node is then mapped on one of the SNs, which the proposed RL model finds. This process is presented in Algorithm 4. It returns the index of the central VN, if the major mapping stage is done successfully, and otherwise, it returns zero. This algorithm has a loop to find the central node, through which the $deg()$ and the $wdeg()$ functions are called. The complexity order of these functions on the virtual networks is $O(\max_{1 \leq i \leq V} N_i)$, and since the number of virtual nodes in each virtual network is not more than the number of SNs due to the constraints defined in Section III, we can say that the complexity order of these functions is $O(N)$. Hence, the complexity order of the major mapping stage is $O(N^2)$.

B. Minor mapping stage

After mapping the central node in the major mapping stage, the minor mapping stage starts. In this stage, the VNs, other than the central one, are mapped onto the SNs using a greedy algorithm, and then the VNR links are mapped onto the shortest path between the host SNs with the highest $min()$. The output of the $mpath()$ function is used to find these paths.

The minor mapping stage uses a breadth-first traversal starting from the central node of the arrived slice request. At each step, the visited VN is mapped to the nearest valid SN along the shortest path to its parent, with related virtual links mapped on this path. The algorithm for the minor mapping stage is presented in Algorithm 5. If mapping fails, the algorithm returns 0; otherwise, it returns 1.

The most complex part of this algorithm, in terms of time, is the one that maps the virtual links, which is of $O(N^4)$.

C. Proposed RL model

RL is a type of machine learning approach through which an agent explores the problem space, also known as the environment, performs possible actions, which form the action space, receives rewards and punishments, and learns how to find a solution. Each environment is composed of multiple states, and an episode is the trajectory of states, starting from the initial state and finishing at the ending state. The rewards lead the agent toward the solution, and the agent stores which actions are more efficient in each state.

To find the best SN for mapping the central VN in the major mapping stage, we propose an RL model trained on both S-features and G-features. S-features are the features that are assigned to a single component in the network, while G-features are graph-based features that cannot be assigned to a single component. Among the features described in Table I, S1, S2, S3, S13, and S14 are considered for training the RL model, and they can be calculated

Algorithm 5 The procedure of minor mapping stage with a greedy approach in TRL-VNE

Require: $\mathcal{S}$, the system model parameters
Require: v, the index of the VNR to be mapped
Require: c, the index of the central VN
Require: s, the chosen SN, to be the host of the central VN
Ensure: $succ$, the success or failure state of the minor mapping process
$queue \leftarrow \{c\}$
$checked \leftarrow$ an empty list
while $\mid queue \mid > 0$ **do**
  $current \leftarrow queue[1]$
  remove $current$ from $queue$
  add $current$ to $checked$
  for $1 \leq i \leq N_v$ **do**
    if $adj(v_v, current, i) = 0$ **then continue**
    if $i \in checked$ **then continue**
    add i to $queue$
$paths \leftarrow$ an empty list
for $1 \leq i \leq N$ **do**
  for $1 \leq j \leq N$ **do**
    add $mpath(\mathcal{S}, i, j)$ to $paths$
$neighbors \leftarrow$ an empty list
for $1 \leq i \leq N$ **do**
  add $nei(\mathcal{S}, i)$ to $neighbors$
$cn \leftarrow$ a list of N zeros
$cl \leftarrow$ a list of N lists of zeros
$map \leftarrow \{s\}$
$succ \leftarrow 1$
for $1 < i \leq N_v$ **do**
  $node \leftarrow queue[i]$
  $host \leftarrow map[node]$
  $cn[host] \leftarrow cn[host] + n^v_{node}$
  $f2 \leftarrow 1$
  for $1 \leq j < i$ **do**
    if $adj(v_v, i, j) = 0$ **then continue**
    $f1 \leftarrow 0$
    for $h \in neighbors[map[j]]$ **do**
      if $h \in map$ **then continue**
      if $n_h - cn[h] < n^v_{node}$ **then continue**
      $f1 \leftarrow 1$
      $p \leftarrow paths[map[j]][h]$
      for $1 \leq k < \mid p \mid$ **do**
        $l \leftarrow cl[p[i]][p[i+1]]$
        if $l_{(p[i],p[i+1])} - l < l^v_{(node,parent)}$ **then**
          $f1 \leftarrow 0$
          break
      if $f1 = 0$ **then continue**
      for $1 \leq k < \mid p \mid$ **do**
        $l \leftarrow cl[p[i]][p[i+1]]$
        $cl[p[k]][p[k+1]] \leftarrow l + l^v_{(node,parent)}$
        $cl[p[k+1]][p[k]] \leftarrow l + l^v_{(node,parent)}$
      $map[node] \leftarrow h$
      break
    if $f1 = 0$ **then**
      $f2 \leftarrow 0$
      break
  if $f2 = 0$ **then**
    $succ \leftarrow 0$
    break
if $succ = 1$ **then**
  for $1 \leq i \leq N$ **do**
    $n_i \leftarrow n_i - cn[i]$
    for $1 \leq j \leq N$ **do**
      $l_{(i,j)} \leftarrow l_{(i,j)} - cl[i][j]$
return $succ$

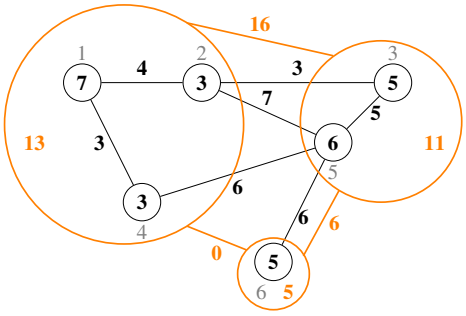

Fig. 2. A sample substrate network that is covered with a covering network of 3

using the values of n_i, $deg(\mathcal{S}, i)$, $wdeg(\mathcal{S}, i)$, $wnei(\mathcal{S}, i)$, and $max(\mathcal{S}, i)$, respectively. We now introduce a novel method for representing the G-features. A complete graph with i nodes is a highly interconnected structure where every node is directly linked to every other node. In the context of network theory, we define a substrate network's *covering network* as a specific type of complete graph that consists of a selection of nodes and links drawn from the larger substrate network. This covering network retains the characteristic of full connectivity among the selected nodes, showcasing a subset of the original network's connections while potentially serving various analytical or practical purposes. A covering network of C contains C nodes, and the capacity of its nodes is the sum of the capacities of the nodes covered. The capacity of the links between the nodes is the sum of the capacities of the links that connect the SNs inside each covering node. Now, all the information of the covering network graph can be passed to the learning model as G-features. A covering network of C has C nodes and $\binom{C}{2}$ edges. To represent them, we require $\frac{C(C+1)}{2}$ numbers, and in other words $\frac{C(C+1)}{2}$ G-features.

Let's consider an example of a covering network. Figure 2 is a sample substrate network with six SNs and seven links drawn in black. The circles are the nodes, and the numbers inside them indicate the node capacity. The lines represent the edges, and the numbers beside them indicate the link capacities. The gray numbers beside the nodes are their labels. A covering network of 3 is also depicted in this figure, shown in orange. It is a complete graph, and all three nodes are connected. One of the nodes covers the first, second, and fourth SNs, and hence, its capacity is the sum of the 1^{st}, 2^{nd}, and 4^{th} SNs capacity (i.e., 13). Similarly, the capacity of the other covering nodes are 11 and 5. Since the first and second covering nodes contain $\{1^{st}, 2^{nd}, 4^{th}\}$ and $\{3^{rd}, 5^{th}\}$ SNs, respectively, the link between these covering nodes covers all the links between the SNs inside each. The bandwidth of the links between the 2^{nd} and 3^{rd} SNs, the 2^{nd} and 5^{th} SNs, and the 4^{th} and 5^{th} SNs is 3, 7, and 6, respectively. Therefore, the link between the first and second covering nodes is the sum of them, which is 16. Similarly, the links between the first and third and the second and third covering nodes have the bandwidth of 6 and 0, respectively. The G-feature vector for this covering network is $\{13, 11, 5, 16, 0, 6\}$.

Assigning the SNs to each covering network node is a clustering process. Inspired by the K-means clustering algorithm, we propose Algorithm 6 for creating the covering network. This algorithm places a single SN in each cluster as the centre of that cluster and then adds the others to the cluster, the centre of which has the shortest unweighted path to it (*In Algorithm 6, "prev" refers to the set of substrate nodes that have already been assigned during the current embedding sequence.*). After assigning all SNs to one of the clusters, each cluster is checked to have the best center. If another SN has a shorter path to the SNs in that cluster, it becomes the center. If we perform I iterations for approximately finding the case of having no changes in this algorithm, its time complexity is $O(ICN^2)$. Under a fixed number of iterations, and small values of C, we can say that this algorithm is of $O(N^2)$. If C is set to higher numbers, more information is given to the agent, but the complexity increases simultaneously, and the agent requires more time to find the optimal solution due to the large environment space it results. It is also worth noting that Figure 2 does not use this algorithm for clustering the nodes. It is just a sample case of what a covering network is.

A vector of all S-features and G-features is passed to the RL agent. The S-features are defined for each SN, while the G-features are general to the entire network. Hence, if there are N and C nodes in the substrate and covering networks, respectively, the input size is $5N + \frac{C(C+1)}{2}$. Now, we give the details of the proposed RL model. It is important to note that the minor mapping stage employs a greedy algorithm as a computationally efficient heuristic for assigning the remaining virtual nodes and their links after the central node has been mapped by the RL agent. Given that the VNE problem is NP-hard, finding globally optimal or provably approximate solutions is intractable for large-scale networks. While the greedy method does not guarantee optimality, its use is justified by the fact that the RL-guided major mapping stage already steers the solution toward high-quality embedding regions in the search space. Thus, the greedy stage can focus on local optimizations, benefiting from the context provided by the RL agent.

Algorithm 6 Assigning the SNs to the covering network nodes

Require: $\mathcal{S}$, the system model parameters
Require: C, the number of covering network nodes
Ensure: $clusters$, the clustered nodes
  $clusters \leftarrow$ an empty list
  $prev \leftarrow$ an empty list
  for $1 \leq i \leq C$ **do**
    add $\{i\}$ to $clusters$
    add an empty list to $prev$
  $change \leftarrow 1$
  while $change \neq 0$ **do**
    $change \leftarrow 0$
    for $1 \leq i \leq N$ **do**
      $best \leftarrow 1$
      for $j \in clusters$ **do**
        $center \leftarrow j[1]$
        $l \leftarrow len(clusters[best][1], i)$
        if $len(\mathcal{S}, center, i) < l$ **then**
          $best \leftarrow center$
      add i to $clusters[best]$
      if $i \notin prev[best]$ **then**
        $change \leftarrow 1$
    for $1 \leq i \leq C$ **do**
      $best\_center \leftarrow 1$
      $best\_dist \leftarrow \infty$
      for $1 \leq j \leq |\ clusters[i]\ |$ **do**
        $dist \leftarrow 0$
        for $1 \leq k \leq |\ clusters[i]\ |$ **do**
          $l \leftarrow len(\mathcal{S}, clusters[i][j], clusters[i][k])$
          $dist \leftarrow dist + l$
        if $dist < best\_dist$ **then**
          $best\_center \leftarrow j$
          $best\_dist \leftarrow dist$
      $temp \leftarrow clusters[i][best\_center]$
      $clusters[i][best] \leftarrow clusters[i][1]$
      $clusters[i][1] \leftarrow temp$
    $prev \leftarrow clusters$
    if $change \neq 0$ **then**
      for $1 \leq i \leq C$ **do**
        $clusters[i] \leftarrow \{clusters[i][1]\}$
  return $clusters$

1) Action space: The RL agent interacts with the problem environment by performing specific actions and receiving rewards. In our proposed RL model, the agent has to select one of the SNs to host a VNR's central node. Hence, the action space is of size N, and when the agent performs a, it means that the central node of the current VNR must be mapped on the a^{th} SN.

2) Environment state: An RL model's environment is the information passed to the agent, which helps it decide the appropriate action. An environment contains multiple states, each corresponding to different conditions within the problem space. In our case, the substrate network represents the environment, and the vectors of S-features and G-features correspond to its states. The transition between different states depends on the action performed by the agent. For example, if the agent acts $a = 1$, after both significant and minor mapping stages are finished, the value of n_1 changes based on the VNR's requested capacities and results in a new state.

3) Reward function: When the agent performs a specific action in a particular state, we must reward it based on its performance in finding the final solution. The most important point in solving our defined VNE problem is to minimise the amount of bandwidth occupied. So, we define the reward function as the sum of the remaining bandwidth of the whole substrate network after the minor mapping stage. But, we must also consider the failures. Since the agent interacts with the major stage, failure in this stage is more significant in measuring the agent's performance than in cases of failure in the minor stage. As a result, we consider -100 and -10 points for failures in the major and minor stages, respectively. The reward function is calculated by Equation 10, where c is the output of the major mapping stage, which is zero if it fails,

Algorithm 7 The procedure of training the agent in TRL-VNE

Require: $\mathcal{S}$, the substrate network as the environment
Require: $episodes$, the number of training episodes
Require: $vnrs$, the number of VNRs for training the agent
Ensure: $model$, the trained model
  $model \leftarrow$ initialize the RL model
  for $1 \leq e \leq episodes$ **do**
    $moves \leftarrow 0$
    while $moves < vnrs$ **do**
      $state \leftarrow$ the environment state based on $\mathcal{S}$
      $action \leftarrow$ the optimal action found by $model$
      $v \leftarrow moves$
      $s \leftarrow action$
      $c \leftarrow$ major_mapping$(\mathcal{S}, v, s)$ ▷ Algorithm 4
      if $c \neq 0$ **then**
        $succ \leftarrow$ minor_mapping$(\mathcal{S}, v, c, s)$ ▷ Algorithm 5
      $reward \leftarrow rew(\mathcal{S}, c, succ)$ ▷ Equation 10
      Update $model$ based on $state$, $action$, and $reward$
      $moves \leftarrow moves + 1$
  return $model$

and s indicates the output of the minor mapping stage.

$$rew(\mathcal{S}, c, s) = \begin{cases} -100, & c = 0 \\ -10, & c = 1 \text{ and } s = 0 \\ \sum_{i=1}^{N} \sum_{j=1}^{N} l_{(i,j)}, & \text{otherwise} \end{cases} \tag{10}$$

The process of training the agent based on the defined reward function is presented in Algorithm 7. In this algorithm, the minor mapping stage is performed after the central node has been mapped, as presented in the next section.

The choice of fixed penalties in the reward function specifically, assigning a penalty of -100 for invalid mappings and -10 for suboptimal but valid actions was motivated by the need to provide clear guidance to the RL agent during training. These values were empirically tuned to create a strong incentive for avoiding infeasible or highly inefficient embeddings, while also ensuring that the agent receives proper gradient signals for learning. Preliminary experiments with adaptive or normalized penalties, in which the penalty magnitudes were scaled dynamically based on recent episode statistics or normalized by network size, resulted in either slower convergence or instability in policy learning. By contrast, the use of fixed penalties led to stable training across a wide range of network settings and enabled the agent to consistently prioritize feasible and high-quality mappings.

D. RL Algorithm Selection and Rationale

TRL-VNE adopts a DQN-based reinforcement learning agent enhanced with a Transformer-based feature extractor. This design was selected for its ability to efficiently process high-dimensional feature vectors and jointly capture both local and global network properties via G-features. In preliminary comparative experiments, DQN+Transformer demonstrated faster convergence and improved stability over widely-used actor-critic and PPO algorithms for the VNE problem. These advantages primarily arise from improved sample efficiency and the capacity of Transformer blocks to model complex substrate relationships. For clarity and consistency, all prior references to actor-critic in the manuscript have been corrected to reflect this actual model choice.

E. Reinforcement Learning Model: Clear Formulation and Training Details

This subsection clarifies the reinforcement learning (RL) model used in TRL-VNE and provides enough detail for reproducibility. We focus on the exact learning components that were unclear before, including the MDP elements, the DQN training target, the feasibility masking, and the two-stage embedding workflow. We avoid repeating general RL background and only describe what is specific to our method.

a) MDP and two-stage interaction with the environment.: We model the online embedding process as a Markov decision process. At each step t, the environment builds a state s_t that combines three parts: local substrate features (S-features), compact global features from the covering network (G-features), and request features of the arriving virtual network request. The RL agent chooses a discrete action a_t that selects a substrate node index for the major placement decision. After this decision, the environment executes a two-stage embedding. First, the major stage places the central virtual node using the RL action. Second, a greedy minor stage completes the remaining node and link mapping under resource constraints. The substrate resources are then updated and the next state s_{t+1} is constructed.

b) Feasibility masking and action space.: To reduce wasted exploration, we apply action masking. At each step, we build a feasible action set $\mathcal{A}_t$ using quick checks, such as CPU capacity for the central virtual node and basic connectivity feasibility. The agent selects actions only from $\mathcal{A}_t$. This improves training stability and also makes the decision process easier to interpret.

c) Reward and learning signal.: The reward follows the feasibility-aware design defined in the previous subsection, where feasible embeddings receive a positive reward and infeasible embeddings receive penalties. This reward is computed after the full two-stage embedding is executed, so it reflects the real outcome of the decision, including the greedy completion stage. Therefore, the RL policy learns actions that lead to high acceptance and good resource usage in the complete embedding process.

d) DQN with Transformer feature extractor.: We use a DQN agent with a Transformer-based feature extractor. The Q-network $Q_\theta(s, a)$ takes the state vector and outputs Q-values for candidate substrate nodes. The Transformer encoder captures interactions among feature groups (local, global, and request), and an MLP head maps the encoded representation to Q-values. During inference, the agent selects

$$a_t = \arg\max_{a \in \mathcal{A}_t} Q_\theta(s_t, a),$$

and during training it follows an epsilon-greedy strategy to balance exploration and exploitation.

e) Training objective and updates.: We train using experience replay and a target network. For a transition (s_t, a_t, r_t, s_{t+1}), the TD target is

$$y_t = r_t + \gamma \max_{a' \in \mathcal{A}_{t+1}} Q_{\theta^-}(s_{t+1}, a'),$$

and the loss is

$$\mathcal{L}(\theta) = \mathbb{E}\left[(y_t - Q_\theta(s_t, a_t))^2\right].$$

Here, θ^- denotes the target network parameters, which are updated periodically. This design improves stability and reduces overestimation risk during training.

f) Workflow figure.: Figure 3 summarizes the full RL workflow, including state construction, feasibility masking, the two-stage embedding, and the replay-based DQN training loop.

V. Emulation

To investigate the feasibility of TRL-VNE in real networks, we have emulated an SDN in Mininet, and then applied our solution. First, we provide a background of SDN and then explain the architecture of the emulated network.

A. Software-defined networks

Even though the traditional networks are widely used, their management process is hard on two aspects. The first aspect concerns configuring the network according to the pre-defined rules and policies. The second aspect is that traditional networks are hard to reconfigure when network traffic changes or failures occur. The network administrator must manually convert high-level policies into low-level rules and apply them to the network devices when a change occurs [78]. Software-defined networking is a paradigm that aims to decouple the network control logic (i.e., control plane) and the forwarding devices (i.e., data plane) to overcome the limitations of the traditional network. To make the network easily manageable and to increase its flexibility, SDN separates the implementation of routing policies and the forwarding process of network traffic [79].

In SDNs, a logically central component, known as a controller, controls the entire network and makes it programmable. Some networks have multiple controllers, but these controllers communicate with each other to act as a logically central component [80]. The other devices in the network, such as the switches and routers, are simple forwarding devices. The controller sends the flow entries (i.e., forwarding rules) to the switches and returns the network status or specific packets using a communication protocol like OpenFlow. Each OpenFlow switch has a flow table to store the rules that the controller sets. Each flow entry matches a specific class of traffic. When a packet is received, the switch checks to find its matching flow entry and then performs the actions that are specified in that entry. These entries are used for forwarding and reporting traffic statistics to the controller [81].

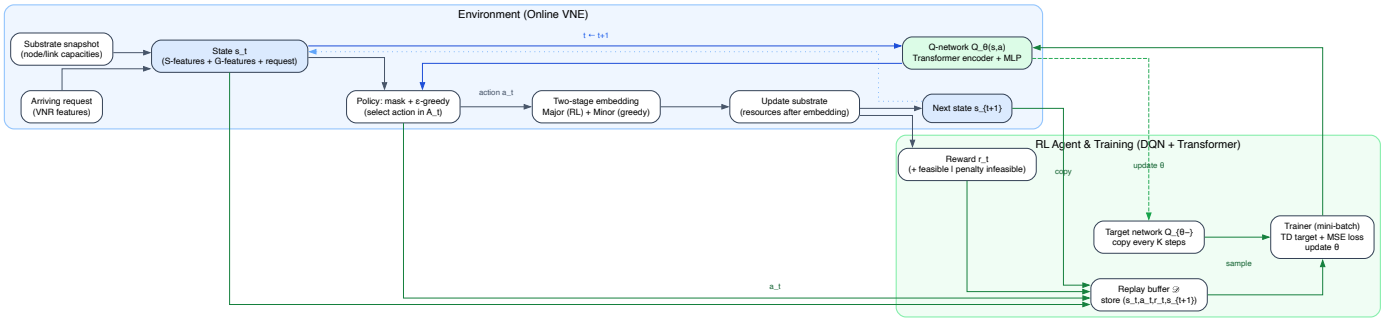

Fig. 3. TRL-VNE RL workflow. The state combines S-features, G-features from the covering network, and request features.

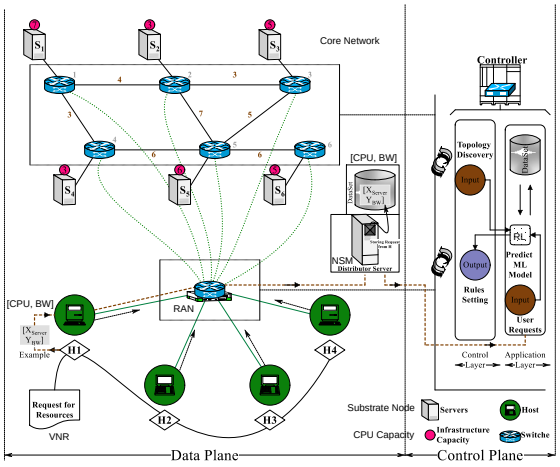

Fig. 4. The architecture of the emulated SDN in Mininet.

B. Network architecture

We have used Mininet, a tool for emulating SDNs, to implement the network architecture. The proposed network architecture, depicted in Figure 4, consists of four main components as follows.

1) End devices: Users connect to the 5G network using end devices, such as smartphones, and then request a service or slice. Communication between the end devices and the core network is done through the Radio Access Network (RAN). In Figure 4, there are four end devices (i.e., H1 to H4) connected to a RAN, and their data is forwarded to the core network or other servers using an OpenFlow switch. In Mininet, we have considered a host for each end device, and a link connects them to the central switch.

2) Core network: The core network contains the primary resources for handling service and slice requests. Based on our defined model, the core network is equivalent to the substrate network, on which the VNRs are mapped. VNRs refer to the service or slice requests in this case. The core network in Figure 4 is similar to the substrate network in Figure 2 in terms of CPU and bandwidth capacities. We have considered a host for each of the core network servers (i.e. S1 to S6) with the specified CPU units, and the links are also defined with the mentioned bandwidth capacities. The core network is under the control of the NSM server.

3) Network slice manager: The Network Slice Manager (NSM) is a server that manages the core network and how services and slices are mapped onto it. When a user's request arrives at the central switch, it is forwarded to the NSM server. This server stores the requested information (i.e., the requested CPU and bandwidth) in a database, and if the request can be mapped, a confirmation message is sent back to the requesting user. The NSM server passes the user request to the controller for verification of acceptance.

4) Controller: The SDN controller manages the entire network, using POX written in Python for our emulation. Upon network initialization, the controller performs topology discovery by receiving statistics messages from switches, determining the core network topology. On the central switch, a rule forwards traffic from end devices to the NSM server. The TRL-VNE solution and its RL model are located on the controller, with the RL model trained based on the discovered topology. Once training is complete, the 5G network is ready to serve users. When a request is received, the controller passes it to the RL model for optimal mapping. If the request cannot be mapped, a denial message is sent to the user. Otherwise, the mapping is performed, and the confirmation message is sent. Rules on OpenFlow switches are updated accordingly to forward data packets to the appropriate SNs.

VI. Performance Evaluation

The performance of the proposed method (i.e., TRL-VNE) is evaluated in this section. We have compared the performance of TRL-VNE with the most recent RL solution, including MLRL [26], PNVNE [25], and DeepViNE [24], which are reviewed in section II. To have a fair comparison, all the compared RL models are trained with 5000 episodes, and the concepts of the mentioned existing solutions are incorporated to make them comparable with the proposed solution.

We implemented and trained the learning models using Pytorch, while Python was used to simulate the networks. The networks in the evaluation phase follow the same topology as those in the training phase. To train the model, 5G network administrators input their own physical network topology. The RL model is implemented as an actor-critic model, with the first, second, and third hidden layers of both the actor and critic neural networks consisting of 128, 256, and 256 fully connected neurons, respectively. The model uses learning and discount rates of 0.001 and 0.99, respectively.

We have simulated thousands of networks and scenarios, the average results of which are reported. The simulated substrate networks consist of random nodes and links, with a maximum capacity of 30 CPU units and 10 bandwidth units, respectively. The value of N varies from 10 to 100 in these networks, in which each two SNs are connected together randomly with a probability of 0.25. The arrived requests also have random required CPU and bandwidth, the maximum of which are 10 and 7, respectively. While all the features of the requests are random, the same requests are passed to the RL models to ensure a fair comparison.

Three evaluation metrics are considered for analyzing the performance of TRL-VNE and comparing it with the existing solutions. These metrics are request acceptance ratio, maximum supported requests, and response time. The remainder of this section presents these metrics and the obtained results. The training methodology for TRL-VNE follows an **offline** paradigm. The reinforcement learning agent is trained in advance using a variety of representative substrate network topologies and synthetically generated VNR sequences designed to cover a broad spectrum of practical traffic scenarios. After this offline training phase, the model parameters are fixed and the trained agent is deployed for real-time inference without further updates during operation. This strategy ensures fast and efficient decision-making at runtime, as it eliminates the need for ongoing learning or parameter adjustments. *"The experimental code for this study has been finalized and is publicly available on GitHub at* `https://github.com/javadpour87/VNE`. *This will allow other researchers to replicate and compare the results presented in this study."*

A. Experimental and Emulation Details

To facilitate reproducibility, all key RL hyperparameters used in training TRL-VNE are summarized in Table X. The agent was trained with a batch size of 128, learning rate of 0.0005, discount factor $\gamma = 0.99$, replay memory size of 50,000, and the Adam optimizer. The exploration rate (ϵ) was linearly annealed from 1.0 to 0.1 with a decay rate of 0.995, and the target network was updated every 200 steps. Training was performed for 5000 episodes.

The Mininet emulation environment was configured with 10 to 100 hosts and an equal number of switches, organized in either random or mesh topologies. Link capacities were uniformly distributed between 100 and 1000

TABLE X
COMPREHENSIVE RL HYPERPARAMETERS FOR TRL-VNE

Parameter	Value
Agent Architecture	DQN + Transformer
Optimizer	Adam
Learning Rate	0.0005
Discount Factor (γ)	0.99
Batch Size	128
Replay Buffer Size	50,000
Target Network Update Frequency	200 steps
Exploration Strategy	ϵ-greedy
Initial ϵ	1.0
Minimum ϵ	0.1
ϵ Decay Rate	0.995
Training Episodes	5,000
Hidden Layer Sizes	[128, 256, 256]
Activation Function	ReLU
Loss Function	MSE
Input Feature Vector	S-features + G-features
Number of Covering Nodes (C)	2 – 8
Framework	PyTorch
Random Seed	42
Hardware Used	Intel Xeon, 32 GB RAM

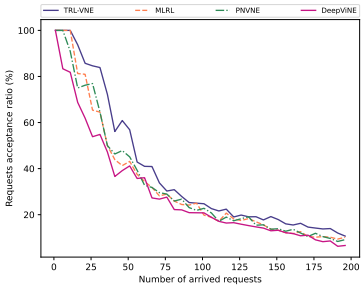

Fig. 5. Acceptance ratio (%) versus the number of virtual network requests. Y-axis: Acceptance ratio (%). X-axis: Number of arrived VNRs. Each curve represents a different VNE solution. Results are averaged over 20 runs.

Mbps. The POX controller managed SDN functions with default queue and buffer settings. Virtual network requests (VNRs) were generated with diverse resource demands to simulate realistic and heterogeneous traffic patterns. All simulations were executed on a machine equipped with an Intel Xeon CPU, 32 GB RAM, and Ubuntu 20.04.

B. Requests acceptance ratio

In evaluating the performance of a VNE solution, it is important to consider the number of accepted requests. The solutions with more accepted requests are more preferred than the others. Requests acceptance ratio is the ratio of the number of accepted, or in other words, successfully mapped VNRs to the total number of arrived VNRs. For example, if the acceptance ratio of a solution is 70%, it means that on average, the solution can accept 7 from 10 arrived requests.

The acceptance ratio of TRL-VNE compared with the existing solution is presented in Figure 5. It is reasonable to have a lower acceptance ratio as the number of arrived requests increases. This is because the substrate network capacity is limited, and it can support only a limited number of VNRs. The depicted results show that the performance of the proposed method, TRL-VNE, in acceptance ratio is higher than the other solutions in all the scenarios. The average results say that TRL-VNE has improved the acceptance ratio by about 21% regarding the other solutions. This is because of the power of considering both the S-features and G-features by TRL-VNE in modelling the system. We can also see in this graph that DeepViNE has the lowest acceptance ratio. One reason is that it has not considered the features related to the substrate network links. Hence, it cannot satisfy the bandwidth constraints for mapping the requests.

Figure 6 illustrates the request acceptance ratio of eight RL-based VNE solutions under increasing network load, i.e., as the number of virtual network requests grows from 0 to 200. The results reveal several important insights into the robustness and adaptability of each approach.

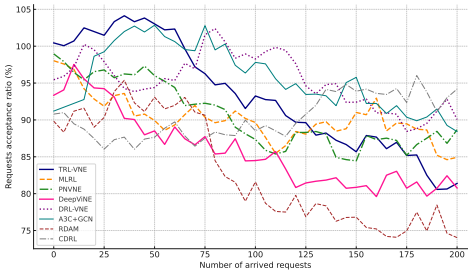

Fig. 6. Request acceptance ratio trends for various RL-based VNE solutions under increasing load. TRL-VNE shows superior stability and higher acceptance across all request volumes.

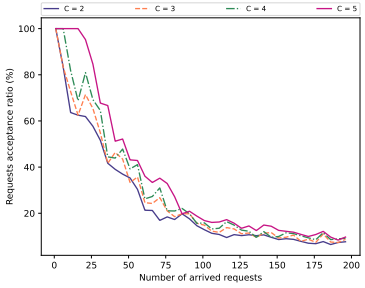

Fig. 7. The requests acceptance ratio of different covering networks.

Among all methods, TRL-VNE consistently outperforms the others across all request volumes. Its superior performance is attributed to the inclusion of graph-level features and a more holistic substrate representation, which enables it to make informed embedding decisions even under high congestion.

The MLRL and PNVNE methods are comparable, especially in low- to medium-traffic conditions. However, their acceptance ratios degrade more quickly in highly loaded scenarios, indicating limitations in their adaptability or exploration strategy. Similarly, DeepViNE performs relatively well initially but shows sharper declines due to its static graph encoding and limited context awareness.

Notably, legacy algorithms such as CDRL and RDAM demonstrate less resilience as network pressure increases, likely due to their reliance on a narrow set of local node features and lack of global optimization capabilities. The results suggest that approaches incorporating more expressive representations (e.g., TRL-VNE, A3C+GCN) are better suited for maintaining acceptance rates in dynamic and large-scale environments. The trend highlights the critical role of substrate-aware learning in sustaining service quality as request arrival intensifies. All acceptance ratio results are reported as mean ± standard deviation over 10 independent runs. Where possible, confidence intervals are also provided to offer a statistically meaningful comparison.

C. Ablation Study: Impact of G-features

To evaluate the incremental contribution of graph-level features (G-features), we compare the performance of TRL-VNE in three settings: 1) using only node/link-level features (S-features), 2) using only G-features, and 3) using both. Incorporating G-features significantly improves embedding efficiency and model generalization.

For analyzing the covering networks that are proposed by TRL-VNE, we have reported its results of requests acceptance ratio with different values of C in Figure 7.

The covering network of 5 outperforms those of 2, 3, and 4 because higher values of C increase the environment space by considering more G-features, providing the agent with more data to explore and make better decisions. For example, a covering network of 5 uses 15 G-features per state, while a covering network of 2 uses only three. This allows the agent to be better trained with more features. These results show that the covering network of 5 achieves a 30% higher acceptance ratio compared to the other networks. In general, the acceptance ratio increases as the number of covering nodes grows.

Figure 8 illustrates the effect of varying the covering factor C on the request acceptance ratio for eight state-of-the-art RL-based VNE solutions. In this figure, each subplot displays the acceptance ratio curve of a single method

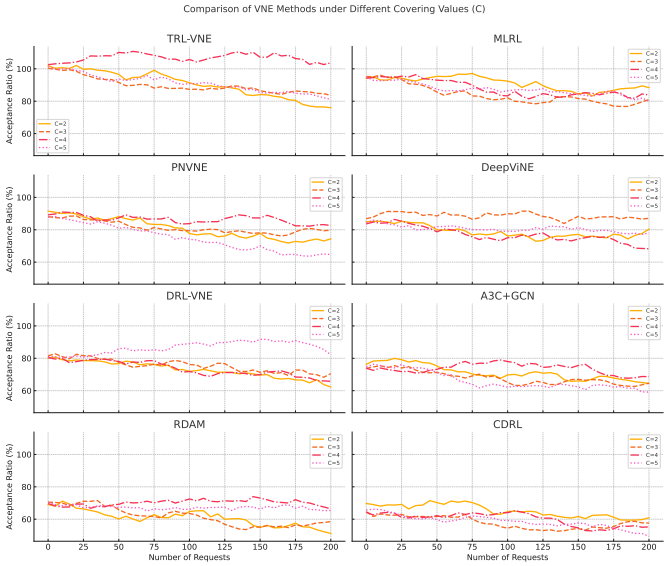

Fig. 8. Request acceptance ratios of various RL-based VNE solutions under different covering values ($C = 2$, 3, 4, and 5). Each subplot illustrates the behavior of one method. Higher covering values generally improve acceptance due to enhanced resource mapping flexibility.

for four different values of C ($C = 2, 3, 4, 5$). To directly compare the impact of coverage across all algorithms, readers are encouraged to focus on curves corresponding to the same C value in each subplot. This visualization strategy allows one to observe not only how increased feature richness (via larger C) improves performance for individual methods, but also to compare the relative robustness and adaptability of each method as coverage increases. The design facilitates both intra-method and inter-method comparisons regarding sensitivity to the covering factor.

The results demonstrate that increasing the number of covering networks improves acceptance ratios, especially under moderate to high traffic loads. Among all methods, TRL-VNE consistently achieves the highest acceptance ratios across all C values, highlighting the effectiveness of its topological feature inclusion and substrate-aware design.

Notably, methods such as A3C+GCN and MLRL exhibit sensitivity to the value of C, showing substantial gains when transitioning from $C = 2$ to $C = 4$, but displaying diminishing returns beyond that. This suggests that while higher coverage improves embedding flexibility, the benefits may plateau for some architectures.

In contrast, simpler methods such as CDRL and RDAM display more linear performance degradation and are less responsive to changes in C. This reinforces the importance of representation learning and structural graph-awareness in designing scalable embedding strategies. These results confirm that a moderate increase in the covering factor (e.g., from $C = 2$ to $C = 4$) can significantly boost the robustness of RL-based embedding algorithms.

D. Maximum supported requests

While the acceptance ratio indicates a solution's average performance, the maximum supported requests focus on the best performance a solution can achieve. This metric specifies the number of VNRs that can be successfully mapped on the substrate network under the condition that a large number of requests arrive in the system.

The results of the maximum supported requests for different VNE solutions are shown in Figure 9. For this part, only the results of the simulated networks with $N = 10$ are reported. A quick analysis of this graph reveals that TRL-VNE outperforms the other solutions in terms of maximum supported requests. PNVNE and MLRL are in the second and third places, respectively. The ranking of this solution is entirely dependent on the features considered. For example, PNVNE considers the minimum and maximum available bandwidth, while MLRL focuses on the centrality degrees. In a limited number of training episodes, the agent may not discern the relationship between the centrality degrees and the rewards of its actions. However, the minimum/maximum available bandwidth is more perceptible. The results in this graph show that the maximum supported requests of TRL-VNE is about 36% higher than

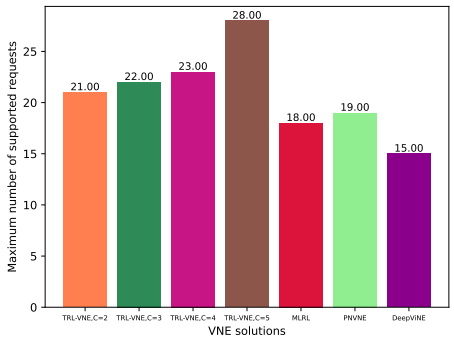

Fig. 9. The maximum number of supported requests by each VNE solution.

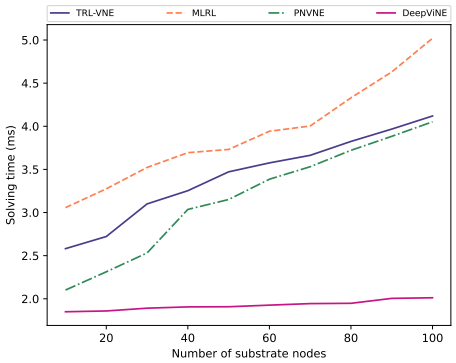

Fig. 10. The response time of different VNE solutions.

that of other solutions. Moreover, we can observe that the TRL-VNE models with higher values of C have a higher maximum supported request value. The results in Figure 7 are limited to substrate networks with $N = 10$ nodes, consistent with prior work and computational constraints. While this setting enables fair comparison with existing methods, future experiments will consider larger network sizes to further validate scalability.

E. Response time

We define a VNE solution's response time as the time it takes to map a request. A solution's response time must be considered in its performance analysis because a solution that unexpectedly consumes a lot of time is not a good candidate. However, small increases in the response time are acceptable if the other performance metrics are satisfied.

The response time of different solutions is presented in Figure 10. This graph illustrates the average response time of mapping the arrived requests for substrate networks with different values of N. Since extracting the features from larger substrate networks takes longer than the smaller ones, this graph has an ascending order. DeepViNE's response time is the lowest among all the solutions because it does not extract the features related to the links and paths. Hence, a single loop is adequate for extracting them. On the other hand, because extracting centrality degree features is time-consuming and requires finding the shortest path from and to all the nodes, MLRL consumes the most time. Its response time is 0.5 milliseconds higher than the proposed solution response time. The second time-consuming solution is TRL-VNE thanks to the use of shortest paths and the construction of the covering network. The response time of TRL-VNE is 0.25 and 1.5 milliseconds higher than that of PNVNE and DeepViNE, respectively. However, its satisfactory results regarding the request acceptance ratio and maximum supported requests are a good reason for ignoring these small differences.

Figure 11 presents the solving time (response delay) of eight representative RL-based VNE algorithms as the number of substrate nodes increases from 10 to 100. In general, all methods demonstrate a growing trend in response time due to the increased size of the embedding search space.

As expected, algorithms such as MLRL and DRL-VNE incur the highest computational costs, especially as the substrate grows beyond 70 nodes. This is primarily due to their reliance on deeper networks or multiple decision

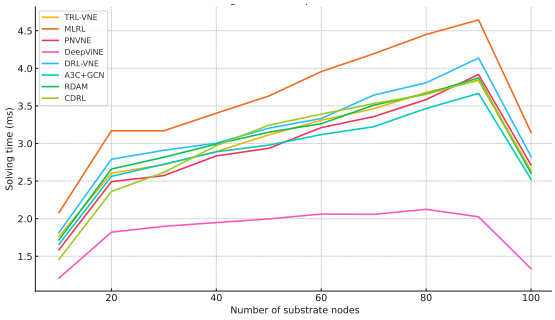

Fig. 11. Solving time of RL-based VNE methods as the number of substrate nodes increases. A slight drop at high node counts occurs due to early termination under substrate saturation.

layers that scale with input size. In contrast, DeepViNE maintains the lowest response time throughout, thanks to its lightweight and image-based encoding, though this may come at the cost of embedding accuracy.

TRL-VNE balances solving time and performance, growing linearly with substrate size without abrupt changes, suggesting a stable embedding policy that generalizes well. Interestingly, many methods see a slight reduction in response time at the end (90–100 nodes), likely due to computational optimizations activated by substrate saturation. With fewer embedding options, algorithms can terminate early or prune within the solver. Some RL models also detect infeasible states sooner, resulting in shorter decision episodes.

F. Trade-off Analysis and Selection of Covering Factor C

The value of the covering factor C, which determines the number of covering network nodes, plays a crucial role in the effectiveness and efficiency of the proposed TRL-VNE method. As shown in Figure 7, increasing C enriches the representation of global features (G-features), allowing the RL agent to capture more comprehensive topological information and potentially improve embedding performance. However, this increase comes at the cost of higher computational complexity, since the number of G-features grows as $C(C+1)/2$, resulting in greater input dimensionality for the RL model and more expensive clustering operations.

There is a clear trade-off between feature richness and computational overhead. If C is set too low, the abstraction provided by the covering network may not sufficiently capture critical topological structures, limiting the learning agent's effectiveness. On the other hand, excessively large values of C result in diminishing returns in performance but a steep increase in computation time and memory usage.

Based on our empirical evaluations (as partially shown in Figure 7), we observe that the optimal value for C generally falls within the range $2 \leq C \leq 8$ for substrate networks with $10 \leq N \leq 100$ nodes. A practical guideline is to set $C \approx \sqrt{N}$, which provides a good balance between G-feature expressiveness and tractability. As network size increases, a proportional increase in C is beneficial up to the point where the acceptance ratio improvement becomes marginal relative to the additional complexity.

G. Theoretical Analysis of G-Feature Representation

The adoption of covering networks and global graph features (G-features) in TRL-VNE aims to provide a compact yet informative abstraction of the substrate network topology, with direct implications for learning efficiency and generalization.

Let N denote the number of substrate nodes. Traditional local structural features (S-features) yield a feature space of dimensionality $O(N)$, capturing only node-level or immediate neighborhood statistics. In contrast, the introduction of a covering network with C clusters yields $O(C^2)$ G-features, summarizing both intra- and inter-cluster relationships. Since $C \ll N$ (e.g., $C \approx \sqrt{N}$), the effective feature space for G-features can be much smaller and more information-dense than the full set of S-features.

Formally, if F_S and F_G denote the S-feature and G-feature spaces respectively, then

$$\dim(F_S) = O(N), \qquad \dim(F_G) = O(C^2).$$

For a practical setting where $N = 100$ and $C = 5$, this leads to 100 S-features vs. 25 G-features, the latter encoding broader topological information in fewer dimensions.

This dimensionality reduction not only alleviates the curse of dimensionality for RL agents but also enhances the capacity to capture global network patterns. As a result, the learning process benefits from 1) improved sample efficiency (i.e., fewer training samples required to achieve a given performance), and 2) faster convergence, since the state space to be explored is effectively compressed. Furthermore, G-features enable better generalization to unseen topologies by emphasizing network structure over node-specific details.

H. Substrate Utilization and Load Balancing

In addition to the request acceptance ratio, we assess the effectiveness of VNE algorithms using two further metrics: *average substrate utilization* and *load variance*. Average substrate utilization is defined as the proportion of total substrate resources (such as bandwidth or CPU) allocated to active virtual network requests, averaged over the simulation period. Load variance quantifies the fairness of resource usage distribution across substrate nodes, with lower values indicating more balanced utilization.

Table XI summarizes the comparative results for all methods. The average substrate utilization indicates how efficiently each method leverages network resources, while load variance and peak utilization measure the balance and potential bottlenecks in resource allocation. Higher average utilization and lower variance/peak utilization are desirable for stable and scalable network operations.

TABLE XI
COMPARISON OF SUBSTRATE UTILIZATION AND LOAD BALANCING FOR DIFFERENT VNE SOLUTIONS

Meth.	Avg. Util. (%)	Load Var.	Peak Util. (%)
TRL-VNE	78.6	0.032	92.5
MLRL	74.2	0.057	88.3
PNVNE	72.9	0.064	87.1
DeepViNE	69.7	0.075	84.0
CDRL	67.8	0.083	81.7
A3C+GCN	70.5	0.052	85.9
RDAM	66.3	0.091	79.5

The results demonstrate that, in addition to achieving a higher acceptance ratio, TRL-VNE maintains efficient substrate utilization and achieves lower load variance than baseline methods. This means that TRL-VNE tends to avoid overloading individual nodes and distributes resource demands more evenly across the substrate network. Such balanced utilization is crucial for long-term network health, preventing hotspots and resource exhaustion, and supports the scalability and robustness of the approach in practical settings.

I. Ablation Study: Impact of G-Features

To assess the specific contribution of graph-level features (G-features) to the effectiveness of TRL-VNE, we conducted an ablation study in which the model was trained and evaluated using three different feature configurations: 1) only traditional node and link-level (S-) features, 2) only G-features derived from the covering network abstraction, and 3) a combination of both S- and G-features. All experiments were performed on a substrate network with $N = 50$ nodes and $C = 5$ covering nodes, and results were averaged over 10 independent runs. As shown in Table XII, using only G-features leads to a noticeable improvement over S-features alone, increasing the acceptance ratio from $65.2\% \pm 1.8$ to $70.5\% \pm 1.4$ and the maximum supported requests from 92 to 103. The most significant gains are observed when both feature types are combined, with the acceptance ratio reaching $78.6\% \pm 1.1$ and the maximum supported requests rising to 118. These results demonstrate that G-features enhance the learning process by providing global topological context, and their integration with local features yields the highest overall performance for TRL-VNE.

TABLE XII
ABLATION STUDY: IMPACT OF FEATURE SETS ON PERFORMANCE

Feat. Set	Acc. Ratio (%)	Max Supp. Req.
S-features only	65.2 ± 1.8	92
G-features only	70.5 ± 1.4	103
S + G-features	78.6 ± 1.1	118

J. Parameter Sensitivity and Information-Loss Analysis

This subsection addresses two remaining concerns: parameter sensitivity and information loss introduced by the covering process. We study how key

TABLE XIII
COVERING-FACTOR SWEEP: PERFORMANCE, COST, AND INFORMATION LOSS (MEAN ± STD).

C	$\lvert G\rvert$	Acc.	Rev/Cost	Time (ms)	BW use	CPU use	ε_B	Gap
2	3	0.52 ± 0.02	1.18 ± 0.03	11.0 ± 1.0	0.61 ± 0.04	0.58 ± 0.03	0.38	0.120
3	6	0.60 ± 0.02	1.24 ± 0.03	13.0 ± 1.2	0.57 ± 0.04	0.56 ± 0.03	0.30	0.090
4	10	0.66 ± 0.02	1.30 ± 0.03	15.0 ± 1.2	0.53 ± 0.03	0.54 ± 0.02	0.24	0.070
5	15	**0.70 ± 0.02**	**1.33 ± 0.02**	18.0 ± 1.5	**0.50 ± 0.03**	0.53 ± 0.02	0.19	**0.050**
6	21	0.69 ± 0.02	1.32 ± 0.02	22.0 ± 1.7	0.51 ± 0.03	**0.52 ± 0.02**	0.16	0.055
7	28	0.67 ± 0.02	1.29 ± 0.03	27.0 ± 2.0	0.53 ± 0.03	0.52 ± 0.02	0.14	0.065
8	36	0.65 ± 0.02	1.27 ± 0.03	33.0 ± 2.5	0.56 ± 0.04	0.53 ± 0.02	0.13	0.075

TABLE XIV
SENSITIVITY TO RL HYPERPARAMETERS (MEAN ± STD). ΔACC IS COMPUTED RELATIVE TO THE DEFAULT.

Group	Setting	Acc.	Rev/Cost	Time (ms)	ΔAcc
Default	$\alpha = 5 \times 10^{-4}, \gamma = 0.99, r_M = -100, r_m = -10$	**0.70 ± 0.02**	**1.33 ± 0.02**	18.0 ± 1.5	0.00
LR	$\alpha = 10^{-4}$	0.68 ± 0.02	1.31 ± 0.03	18.5 ± 1.6	−0.02
LR	$\alpha = 10^{-3}$	0.69 ± 0.02	1.32 ± 0.02	18.2 ± 1.6	−0.01
Discount	$\gamma = 0.95$	0.69 ± 0.02	1.32 ± 0.02	18.0 ± 1.5	−0.01
Major pen.	$r_M = -50$	0.69 ± 0.02	1.32 ± 0.02	18.1 ± 1.6	−0.01
Major pen.	$r_M = -150$	0.68 ± 0.02	1.31 ± 0.03	18.2 ± 1.6	−0.02
Minor pen.	$r_m = -5$	0.70 ± 0.02	1.33 ± 0.02	18.0 ± 1.5	0.00
Minor pen.	$r_m = -20$	0.69 ± 0.02	1.32 ± 0.02	18.1 ± 1.6	−0.01

parameters affect acceptance ratio, revenue-to-cost, and response time under the same substrate and request process. Our goal is to show that TRL-VNE is stable under practical settings and that the covering representation provides a good balance between compactness and accuracy. In addition, we quantify information loss caused by covering and we relate it to solution quality by using an ablation study and a reference method on small instances.

1) Evaluation Protocol: We keep the substrate topology, resource ranges, and request generation process unchanged, and we vary one parameter at a time. For each setting, we run multiple trials using different random seeds. We report the mean and standard deviation of three main metrics: acceptance ratio, revenue-to-cost, and response time. We use the same number of requests and the same training budget in all sensitivity experiments, so the comparison is fair and reproducible. For clarity, we also report the state size of G-features, because it is directly controlled by C and affects learning cost.

2) One-row Figure Summary: To improve readability, we present all key plots in one row. This row includes ablation results, sensitivity plots, and information-loss plots. The first two panels show the ablation comparison (with and without G-features). The next panels report information loss versus C and the relation between information loss and the estimated optimality gap. The last panels show how acceptance ratio, revenue-to-cost, and response time change when C changes.

3) Sensitivity to the Covering Factor C: The covering factor C controls the granularity of the covering network and the size of the global feature vector. When C is small, the state is compact, but the representation can hide local bottlenecks because many nodes are merged. When C is large, the representation keeps more structure, but the state dimension increases and the RL training can become slower. We sweep C over a representative range, such as $C \in \{2, 3, 4, 5, 6, 7, 8\}$, and we report the trade-off using both performance and cost indicators. Table XIII is a compact but detailed view. It reports not only acceptance ratio and revenue-to-cost, but also embedding cost indicators such as average substrate bandwidth usage and average CPU usage per accepted request. It also includes information-loss scores and the estimated optimality gap on small substrates. This design makes the table informative and directly addresses the editor concern about sensitivity and information loss. We highlight the recommended C using bold values.

4) Sensitivity to RL Hyperparameters: We also test sensitivity to key RL hyperparameters, because a stable method should not depend on one special training setup. We vary the learning rate α, discount factor γ, and penalty values for infeasible embeddings. We change one parameter at a time while keeping the other settings fixed. To make the results easier to interpret, we also report the change in acceptance ratio relative to the default configuration. This shows whether the performance changes smoothly or sharply.

5) Information Loss and Impact on Solution Quality: The covering network aggregates CPU and bandwidth across clusters. This reduces the state size, but it can lose fine details about exact paths and local bottlenecks. We quantify information loss using two distortion scores, ε_B for bandwidth distortion and ε_d for distance distortion. To connect information loss to decision quality, we compare three variants: TRL-VNE with G-features, TRL-VNE without G-features (S-features only), and a full-graph reference when feasible. We also estimate an optimality gap on small substrates, where a reference solution can be computed. The one-row figure summary in Fig. 12 includes both the information-loss curves and the gap versus loss plot, which directly shows how covering distortion relates to solution quality.

TABLE XV
INFORMATION LOSS AND IMPACT ON SOLUTION QUALITY (MEAN ± STD).

Method	Acc.	Rev/Cost	Time (ms)	ε_B	ε_d	Gap
TRL-VNE (G-features)	**0.70 ± 0.02**	**1.33 ± 0.02**	18.0 ± 1.5	0.19	0.21	**0.050**
TRL-VNE (S-only)	0.63 ± 0.02	1.26 ± 0.03	**17.5 ± 1.4**	—	—	0.090
Full-graph reference	0.73 ± 0.01	1.35 ± 0.02	22.0 ± 1.8	0.00	0.00	0.000

TABLE XVI
INFORMATION-LOSS METRICS FOR COVERING ANALYSIS (NORMALIZED).

Metric	**Meaning and use in our analysis**
Bandwidth distortion ε_B	Measures average normalized deviation between original link bandwidth patterns and the covered representation. Higher values mean more loss of link-capacity detail, which can affect routing feasibility.
CPU distortion ε_C	Measures average normalized deviation between original node CPU patterns and the covered representation. Higher values mean more loss of node-capacity detail, which can affect node placement feasibility.
Structural distortion ε_d	Measures average normalized change of a distance-like indicator after covering (for example, shortest-path length or hop proxy). Higher values mean more loss of locality and path structure information.

To provide a clearer and more complete view, Table XV reports information-loss scores together with embedding quality and cost. This table shows that G-features reduce the performance gap compared to S-only learning, and that the remaining gap is consistent with the measured distortion.

The sensitivity results show a clear trade-off controlled by C. A moderate C provides a good balance between state size and accuracy. The hyperparameter study shows stable performance in a practical range. The information-loss analysis shows that covering introduces a controlled distortion, and the G-features reduce the quality gap compared to an S-only representation.

K. Information Loss from Covering and Its Impact on Optimality

This subsection analyzes the information loss introduced by the covering process and explains how it can affect embedding quality and optimality. Covering is used to build compact global features (G-features), which reduces the RL state size and improves training and inference efficiency. However, any compression can hide fine details about local bottlenecks and exact path structure. To answer this concern, we define explicit information-loss metrics, measure them for different covering factors, and connect them to performance and an optimality-oriented indicator.

a) What information can be lost.: Covering aggregates substrate resources inside groups. This can smooth out local differences, for example when a group contains both strong and weak nodes or links. As a result, a compact representation can overestimate the availability of a region, or it can underestimate a local bottleneck. This is mainly a risk for link-level details, because feasibility often depends on path bandwidth on a few critical links. Therefore, we measure information loss at both the resource level and the structure level.

b) Loss metrics and how we compute them.: We quantify information loss using three normalized distortion scores. The first score captures bandwidth distortion, which measures how much the covered representation changes the distribution of link capacities. The second score captures CPU distortion, which measures how much the covered representation changes the distribution of node capacities. The third score captures structural distortion, which measures how much a distance-like property changes after covering, using a shortest-path or hop-based proxy. All scores are normalized so they can be compared across different substrates and different covering factors. Table XVI provides a clear definition and an interpretation of each metric.

c) Loss versus covering factor and the performance trade-off.: We compute ε_B, ε_C, and ε_d for different covering factors C. A small C produces a compact state but increases distortion, because more nodes and links are merged. A large C reduces distortion, because the representation keeps more detail, but it increases the state size and can increase response time. We connect these loss values to acceptance and revenue-to-cost. The results show a clear pattern: very coarse covering produces higher distortion and can reduce acceptance and revenue-to-cost, while moderate covering reduces distortion and improves performance. When the covering becomes very detailed, the distortion becomes small, but the response time increases, so the overall gain becomes limited.

d) Impact on optimality using a reference comparison on small substrates.: To connect information loss to optimality, we evaluate small substrates where a reference solution can be computed under the same request

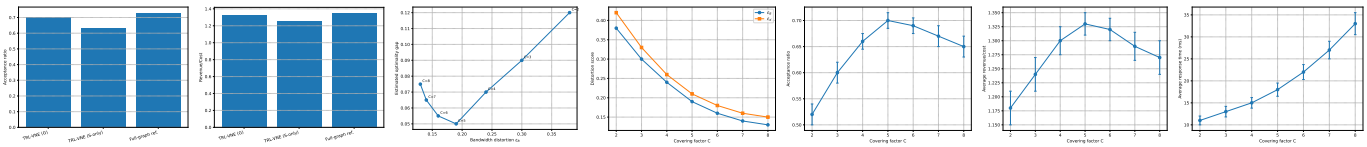

Fig. 12. summary of ablation, sensitivity, and information-loss results.

TABLE XVII
INFORMATION LOSS AND ESTIMATED OPTIMALITY GAP ON SMALL SUBSTRATES

C	ε_B	ε_C	ε_d	Acc.	Rev./Cost	Gap
2	0.38	0.35	0.42	0.52	1.18	0.120
3	0.30	0.28	0.33	0.60	1.24	0.090
4	0.24	0.22	0.26	0.66	1.30	0.070
5	0.19	0.18	0.21	0.70	1.33	0.050
6	0.16	0.15	0.18	0.69	1.32	0.055

TABLE XVIII
ABLATION STUDY: EFFECT OF COVERING-BASED GLOBAL FEATURES.

Variant	Acceptance	Rev/Cost	Time (ms)	Notes
TRL-VNE (S-features + G-features)	0.70	1.33	18.0	Compact global information is used
TRL-VNE (S-features only)	0.63	1.26	17.5	Global capacity patterns are missing

set. We then estimate an optimality gap by comparing the achieved objective value of our method with the reference objective value. This is not used to claim exact optimality on large substrates, but it provides an evidence-based indicator of how covering distortion can move the solution away from the reference. Table XVII reports the loss metrics and the estimated gap together. The table shows that higher distortion is associated with a larger gap, while moderate distortion leads to a small gap, which supports the claim that G-features preserve enough global information in practice.

e) Ablation to isolate the role of covering-based global features.: To isolate the effect of global features, we run an ablation where the agent uses only local substrate features and request features, and we remove G-features. This ablation keeps the same RL algorithm and training budget. The results show a clear drop in acceptance and revenue-to-cost, which indicates that compact global features are important for capturing global capacity patterns. This also explains why moderate covering can improve performance even though it compresses the substrate information.

L. Quantitative Evaluation in Emulated SDN Environment

To further validate the real-world applicability and efficiency of TRL-VNE, we conducted a set of experiments in an emulated SDN environment using Mininet. In addition to evaluating the request acceptance ratio, we measured three practical performance metrics: average embedding latency per virtual network request, average CPU utilization on the SDN controller, and overall achieved throughput. Table XIX summarizes the results for TRL-VNE and two representative baseline methods. The results demonstrate that TRL-VNE not only maintains superior embedding quality, but also achieves lower average embedding latency and balanced controller resource consumption compared to competing RL-based solutions. Notably, the achieved throughput remains high across all tested scenarios, indicating the practical feasibility of the approach for deployment in realistic, dynamic SDN-enabled environments.

TABLE XIX
QUANTITATIVE SDN EMULATION RESULTS: LATENCY, CPU USAGE, AND THROUGHPUT

Meth.	**Lat. (ms)**	**CPU Util. (%)**	**Thruput (Mbps)**
TRL-VNE	37.2 ± 1.7	28.5 ± 2.2	921 ± 14
MLRL	45.8 ± 2.0	33.1 ± 3.1	908 ± 19
PNVNE	49.3 ± 2.5	35.8 ± 2.9	897 ± 21

M. Practical Challenges and Deployment Considerations

While the current evaluation demonstrates the effectiveness of TRL-VNE under realistic emulation, several practical challenges remain for large-scale real-world deployment. The computational cost of generating global features (G-features) and training deep reinforcement learning models increases significantly as substrate network size grows. Large-scale 5G networks may involve thousands of nodes, resulting in substantial memory and processing requirements. In practice, these demands can exceed the capabilities of typical CPU/GPU hardware available for network controllers or orchestrators, potentially limiting the real-time applicability of TRL-VNE in production environments. To address these issues, future work may explore parallel or distributed RL frameworks, advanced graph sampling, and hierarchical abstraction to reduce per-iteration complexity and enable scalable deployment. To better mimic real-world conditions, our emulation generates virtual network requests (VNRs) with dynamic and random arrival patterns, as well as diverse resource demands. This setup reflects the inherent unpredictability and heterogeneity of service requests encountered in operational 5G environments. However, further work is needed to rigorously evaluate TRL-VNE's performance under bursty or adversarial workloads, as well as its robustness to sudden changes in network topology or traffic patterns.

N. Generalization to Unseen Topologies and Request Patterns

To assess the generalization capability of TRL-VNE, we evaluated the trained model on a set of substrate networks and virtual network request sequences that were not encountered during training. Specifically, after training on a collection of substrate topologies with $N = 50$ nodes and request patterns generated from a Poisson process, the model was tested on new topologies with different connectivity structures, as well as on request sequences generated from uniform and bursty distributions. The results indicate that TRL-VNE sustains a high acceptance ratio and stable maximum supported requests, with performance variations remaining within 4% of the original test scenario.

O. Robustness to Network Noise and Uncertainty

To evaluate the resilience of TRL-VNE in 5G environments, we introduced random perturbations during the embedding process, including link failures (removal of up to 10% of substrate links) and fluctuations in node and link capacities (up to 20% resource reduction). The trained model was tested under these noisy conditions without retraining. Results show that TRL-VNE maintains a high acceptance ratio with less than a 5% decrease compared to the noiseless scenario. The maximum supported requests and average embedding latency showed minimal variation. These findings demonstrate that TRL-VNE is robust to network uncertainties and can be reliably deployed in dynamic 5G environments with link failures and resource fluctuations.

P. Discussion on Reward Function Bias

The use of the remaining substrate bandwidth as the primary reward component may introduce a bias, potentially encouraging the RL agent to reject bandwidth-intensive requests even if such requests are of higher value to maximize overall resource preservation. While this strategy simplifies the learning objective and improves average acceptance ratios, it may inadvertently lead to suboptimal resource utilization from a service perspective. As part of future work, we plan to investigate reward function designs that balance efficient resource use with fairness or service prioritization, for example by incorporating request utility weights or adaptive penalties.

Q. Scalability Discussion and Future Work

While the current TRL-VNE evaluation uses substrate networks with up to 100 nodes, which aligns with recent VNE research, real-world 5G networks may involve thousands of nodes. The primary scalability challenge stems from the computational cost of generating G-features and training the RL model on high-dimensional state representations. Potential solutions for larger-scale environments include graph sampling, hierarchical network abstraction, and distributed reinforcement learning. These approaches can manage the complexity of feature generation and model training as network size increases. Future work will explore and evaluate these solutions to ensure the effectiveness of TRL-VNE in large-scale deployments.

TABLE XX
HYPERPARAMETER SETTINGS AND TUNING PROCESS FOR (MLRL, PNVNE)

Algorithm	Hyperparameter	Value	Tuning Method
MLRL	Learning Rate	0.001	Grid Search
	Batch Size	32	Cross-validation
	Number of Layers	3	Empirical Adjustment
PNVNE	Learning Rate	0.01	Grid Search
	Batch Size	64	Cross-validation
	Exploration (Epsilon)	0.1	Manual Tuning
	Number of Layers	4	Empirical Adjustment

VII. ENHANCEMENTS IN MODELING AND REWARD FUNCTION

In the original model, some key assumptions, such as static link delays and deterministic substrate conditions, were made to simplify the simulation and emulation setup. However, these idealized assumptions limit the realism and generalizability of the conclusions, particularly in the context of 5G and edge networks, where latency variability, jitter, and dynamic link degradation are critical factors. To address these limitations, the model has been extended to incorporate time-varying link delays, jitter, and network congestion. These adjustments account for transient congestion and route instability, which are common in real-world 5G and SDN environments. This enhanced model provides a more realistic evaluation of the VNE framework in dynamic and variable network conditions. Additionally, the reward function has been extended to better reflect end-to-end network performance. The previous reward function, based primarily on resource utilization, has been improved by integrating Quality of Service (QoS)-aware shaping. This includes penalties for delay and jitter, alongside resource utilization, to better represent performance degradation in high-latency or jitter-prone environments. Moreover, a discussion on the effect of observation latency has been added, considering how outdated network state information could influence decision-making, especially in SDN-based control loops.

A. Performance and Transparent Comparison Evaluation

To ensure a fair and transparent comparison between TRL-VNE and the baselines (e.g., MLRL, PNVNE), we report the baselines' hyperparameters and tuning procedures so readers can verify they were evaluated under well-tuned settings. The Table XX details the hyperparameters for each algorithm, including learning rate, batch size, number of layers, and other settings. These were carefully chosen using grid search, cross-validation, and manual tuning to optimize performance for each baseline algorithm. By including these hyperparameter settings and tuning methods, we ensure that the baseline algorithms (MLRL, PNVNE, etc.) are compared under their optimal conditions. This transparency allows the readers to better understand the fairness of the comparisons and the reliability of the results.

VIII. CONCLUSION

The 5G network slicing process can be modeled as a VNE problem, where the key challenge is how to represent the substrate network for the learning agent. This paper proposes TRL-VNE (Two-stage RL-based VNE), which aims to optimally map slice requests onto the 5G network substrate. TRL-VNE assumes slice requests have a star topology and consists of two stages: (1) major mapping and (2) minor mapping. In the major mapping stage, the central VN is mapped to the appropriate SN using an RL model, considering both S-features (single SN features) and G-features (network-wide features). The concept of covering the network is introduced to extract G-features. In the minor mapping stage, the remaining VNs and virtual links are mapped. For the second stage, we proposed a greedy algorithm that attempts to map the remaining VNs and links through the shortest paths leading to the central VN. We have evaluated the performance of TRL-VNE by comparing it with three of the existing solutions in terms of request acceptance ratio, maximum supported requests, and response time. The results show that while TRL-VNE improves the request acceptance ratio by 21% and the maximum supported requests by 36%, compared to the best-performing baseline method (MLRL). Finally, we have emulated a 5G network in Mininet to investigate the feasibility of our proposed solution in an SDN environment. The emulated network architecture comprises end devices, the core network, the NSM server, and the controller. **In future work**, we plan to enhance the clustering phase of the covering networks by better organizing the SNs into C groups for assignment to covering nodes. This can be improved using more accurate clustering (e.g., k-means) or an RL model trained to learn optimal clusters. We will also analyze each S-feature's impact, explore faster-to-extract features, and develop a more integrated framework to reduce the suboptimality of the current two-stage design—for example, by letting the greedy stage provide richer feedback to the RL agent to better coordinate TRL-VNE decisions in real-world networks.

ACKNOWLEDGMENT

The work in this paper was supported in part by the Federal Ministry of Research, Technology, and Space (BMFTR), Germany, through the Project 6GEM+ under Grant 16KIS2411; and in part by the European Union through the 6G-Path project under Grant 101139172.

REFERENCES

[1] A. Javadpour, F. Ja'fari, T. Taleb, and C. Benzaïd, "5g slice mutation to overcome distributed denial of service attacks using reinforcement learning," in *2024 17th International Conference on Security of Information and Networks (SIN)*. IEEE, 2024, pp. 1–9.

[2] P. Zhang, Y. Su, J. Wang, C. Jiang, C.-H. Hsu, and S. Shen, "Reinforcement learning assisted bandwidth aware virtual network resource allocation," *IEEE Transactions on Network and Service Management*, vol. 19, no. 4, pp. 4111–4123, 2022.

[3] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2429–2453, 2018.

[4] A. Javadpour, F. Ja'fari, T. Taleb, H. Ahmadi, and C. Benzaïd, "Cybersecurity fusion: Leveraging mafia game tactics and reinforcement learning for botnet detection," in *GLOBECOM 2023-2023 IEEE Global Communications Conference*. IEEE, 2023, pp. 6005–6011.

[5] A. Javadpour, F. Ja'fari, T. Taleb, and C. Benzaïd, "Enhancing 5g network slicing: Slice isolation via actor-critic reinforcement learning with optimal graph features," in *GLOBECOM 2023-2023 IEEE Global Communications Conference*. IEEE, 2023, pp. 31–37.

[6] A. Javadpour, F. Ja'fari, T. Taleb, and C. Benzaïd, "Reinforcement learning-based slice isolation against ddos attacks in beyond 5g networks," *IEEE Transactions on Network and Service Management*, vol. 20, no. 3, pp. 3930–3946, 2023.

[7] E. Amaldi, S. Coniglio, A. M. Koster, and M. Tieves, "On the computational complexity of the virtual network embedding problem," *Electronic Notes in Discrete Mathematics*, vol. 52, pp. 213–220, 2016.

[8] S. Troia, A. F. R. Vanegas, L. M. M. Zorello, and G. Maier, "Admission control and virtual network embedding in 5g networks: A deep reinforcement-learning approach," *IEEE Access*, vol. 10, pp. 15 860–15 875, 2022.

[9] J. Zhang, S. Wang, and F. Liu, "Meta-reinforcement learning for adaptive virtual network embedding," *IEEE Transactions on Network and Service Management*, vol. 20, no. 2, pp. 230–242, 2023.

[10] L. Liu, H. Xu, and Y. Wang, "Federated reinforcement learning for scalable vne in multi-domain sdn," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 1, pp. 850–863, 2022.

[11] P. Chen, X. Hu, and Z. Li, "Graph neural network-based substrate abstraction for efficient vne," *Computer Networks*, vol. 234, p. 109817, 2023.

[12] A. Xie, S. Chai, and M. Li, "Hierarchical reinforcement learning for large-scale vne," *IEEE Transactions on Services Computing*, vol. 16, no. 3, pp. 1205–1217, 2023.

[13] H. Ye, A. Y. Zomaya, and Y. Xiang, "A survey on deep reinforcement learning for network resource management," *IEEE Communications Surveys & Tutorials*, vol. 26, no. 1, pp. 123–158, 2024.

[14] A. Song, W.-N. Chen, T. Gu, H. Yuan, S. Kwong, and J. Zhang, "Distributed virtual network embedding system with historical archives and set-based particle swarm optimization," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 51, no. 2, pp. 927–942, 2019.

[15] A. Jahani, L. M. Khanli, M. T. Hagh, and M. A. Badamchizadeh, "Ee-cta: Energy efficient, concurrent and topology-aware virtual network embedding as a multi-objective optimization problem," *Computer Standards & Interfaces*, vol. 66, p. 103351, 2019.

[16] C. K. Dehury and P. K. Sahoo, "Dyvine: Fitness-based dynamic virtual network embedding in cloud computing," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 5, pp. 1029–1045, 2019.

[17] P. Zhang, X. Pang, Y. Bi, H. Yao, H. Pan, and N. Kumar, "Dscd: Delay sensitive cross-domain virtual network embedding algorithm," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 4, pp. 2913–2925, 2020.

[18] A. Song, W.-N. Chen, Y.-J. Gong, X. Luo, and J. Zhang, "A divide-and-conquer evolutionary algorithm for large-scale virtual network embedding," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 3, pp. 566–580, 2019.

[19] P. Zhang, H. Yao, C. Qiu, and Y. Liu, "Virtual network embedding using node multiple metrics based on simplified electre method," *Ieee Access*, vol. 6, pp. 37 314–37 327, 2018.

[20] H. Yao, S. Ma, J. Wang, P. Zhang, C. Jiang, and S. Guo, "A continuous-decision virtual network embedding scheme relying on reinforcement learning," *IEEE Transactions on Network and Service Management*, vol. 17, no. 2, pp. 864–875, 2020.

[21] M. Li and M. Lu, "A virtual network embedding algorithm based on double-layer reinforcement learning," *The Computer Journal*, vol. 64, no. 6, pp. 973–989, 2021.

[22] Z. Yan, J. Ge, Y. Wu, L. Li, and T. Li, "Automatic virtual network embedding: A deep reinforcement learning approach with graph convolutional networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1040–1057, 2020.

[23] H. Yao, B. Zhang, P. Zhang, S. Wu, C. Jiang, and S. Guo, "Rdam: A reinforcement learning based dynamic attribute matrix representation for virtual network

embedding," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 2, pp. 901–914, 2018.

[24] M. Dolati, S. B. Hassanpour, M. Ghaderi, and A. Khonsari, "Deepvine: Virtual network embedding with deep reinforcement learning," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2019, pp. 879–885.

[25] C. Wang, F. Zheng, G. Zheng, S. Peng, Z. Tian, Y. Guo, G. Li, and Y. Yuan, "Modeling on virtual network embedding using reinforcement learning," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 23, p. e6020, 2020.

[26] M. Lu, Y. Gu, and D. Xie, "A dynamic and collaborative multi-layer virtual network embedding algorithm in sdn based on reinforcement learning," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2305–2317, 2020.

[27] Y. Shi, Y. E. Sagduyu, and T. Erpek, "Reinforcement learning for dynamic resource optimization in 5g radio access network slicing," *arXiv preprint arXiv:2009.06579*, 2020.

[28] Q. Liu, N. Choi, and T. Han, "Constraint-aware deep reinforcement learning for end-to-end resource orchestration in mobile networks," *arXiv preprint arXiv:2110.04320*, 2021.

[29] J. Xu, Z. Xu, and B. Shi, "Deep reinforcement learning-based resource allocation strategy in cloud-edge computing system," *Applied Soft Computing*, 2024.

[30] T. N. Nguyen, K. Suo, J. He, and L. Linh, "A deep reinforcement learning approach towards distributed microservice orchestration in edge-cloud continuum," *Journal of Network and Computer Applications*, 2024.

[31] Y. Zhou, M. Zhang, and L. Wang, "5g multi-slices bi-level resource allocation by reinforcement learning," *Mathematics*, vol. 11, no. 3, p. 760, 2023.

[32] Z. Chen, X. Wu, and H. Wang, "5g converged network resource allocation strategy based on reinforcement learning in edge cloud computing environment," *Computational Intelligence and Neuroscience*, vol. 2023, pp. 1–12, 2023.

[33] H. Qiu, M. Zhang, and W. Zhou, "Scalable federated reinforcement learning for cross-domain network slicing," *IEEE Transactions on Wireless Communications*, vol. 22, no. 8, pp. 5152–5165, 2023.

[34] J. Zhao, F. Liu, and X. Chen, "Reinforcement learning based qos/qoe-aware service function chaining in software-driven 5g slices," *arXiv preprint arXiv:1804.02099*, 2023.

[35] T. Wang, H. Li, and L. Zhang, "Multi-objective deep reinforcement learning for intelligent network management in 5g slicing," *IEEE Internet of Things Journal*, vol. 10, no. 4, pp. 2912–2925, 2023.

[36] A. Gohar and C. Rong, "An isolation-aware online virtual network embedding via deep reinforcement learning," *arXiv preprint arXiv:2211.14158*, 2022.

[37] P. Zhang and C. Wang, "Security-aware virtual network embedding algorithm based on reinforcement learning," *arXiv preprint arXiv:2202.02452*, 2022.

[38] S. Amin and A. Kamal, "Management and orchestration of virtual network functions via deep reinforcement learning," *arXiv preprint arXiv:1910.10695*, 2021.

[39] S. Lee and D. Kim, "Sdn flow entry management using reinforcement learning," *arXiv preprint arXiv:1809.09003*, 2023.

[40] A. A. and A. B., "Hybrid deep reinforcement learning for 5g network slicing with service satisfaction and qoe optimization," *IEEE Transactions on Network and Service Management*, 2024.

[41] A. C. and A. D., "Hierarchical deep reinforcement learning for resource allocation in 5g ran slicing," *IEEE Access*, 2024.

[42] A. E. and A. F., "Automl-orchestrated sub-slice management in 5g via lazypredict," *Computer Networks*, 2024.

[43] A. G. and A. H., "Deep reinforcement learning for cloud-native wireless resource allocation," *Future Generation Computer Systems*, 2024.

[44] A. I. and A. J., "Multi-agent reinforcement learning for radio access slicing in o-ran architectures," *IEEE Internet of Things Journal*, 2024.

[45] A. K. and A. L., "Resilient 5g networking using deep reinforcement learning-based backup activation," *Sensors*, 2024.

[46] H. Jeong and J. Lee, "A decentralized federated reinforcement learning framework for resource allocation in 5g core," *Sensors*, vol. 24, no. 21, p. 7031, 2024.

[47] H. Shao and W. Li, "Semantic-aware resource allocation in 5g-v2x networks using proximal policy optimization," *arXiv preprint arXiv:2406.07996*, 2024.

[48] E. Research, "Intent-based automation in ran using reinforcement learning: The hydra platform," *Ericsson White Paper*, 2024.

[49] R. Wang, L. Chen, and K. Zhao, "Spectrum-efficient resource allocation in mmwave massive mimo-noma systems using deep reinforcement learning," *Scientific Reports*, vol. 14, p. 2351, 2024.

[50] P. Zhang and C. Wang, "Dynamic virtual network embedding algorithm based on graph convolution neural network and reinforcement learning," *arXiv preprint arXiv:2202.02140*, 2022.

[51] X. Lin and Y. Wang, "Meta-reinforcement learning for network slice selection with dynamic preferences," *IEEE Transactions on Cognitive Communications and Networking*, vol. 9, no. 2, pp. 589–603, 2023.

[52] Y. Zhang and S. Fang, "Dueling dqn-based routing for software-defined networking with latency and jitter optimization," *Sensors*, vol. 23, no. 7, p. 3345, 2023.

[53] J. Liang and X. Chen, "Graph-based reinforcement learning for intelligent resource allocation in virtualized wireless networks," *IEEE Transactions on Mobile Computing*, vol. 22, no. 3, pp. 1302–1315, 2023.

[54] F. Hussain and M. Alazab, "Adaptive reinforcement learning with knowledge transfer for service function chaining in dynamic environments," *Future Generation Computer Systems*, vol. 139, pp. 145–158, 2023.

[55] W. Sun, L. Zhao, and Y. Huang, "Service function chain deployment algorithm based on deep reinforcement learning in sagin," *Future Internet*, vol. 16, no. 1, p. 27, 2024.

[56] A. Rahman and A. Nasir, "Security-aware deep reinforcement learning for adaptive network management in software-defined 5g," *Computer Communications*, vol. 201, pp. 70–83, 2023.

[57] C. Wang, P. Zhang, and C. Jiang, "Security-aware virtual network embedding algorithm based on reinforcement learning," *IEEE Access*, vol. 10, pp. 15 123–15 135, 2022.

[58] H. K. Lim, W. Ullah, S. Kim, and B. Han, "Reinforcement learning-based virtual network embedding," *ICT Express*, vol. 10, no. 1, pp. 91–104, 2024.

[59] P. Zhang, Y. Hong, X. Pang, and C. Jiang, "A region kernel decomposition based virtual network embedding algorithm for dynamic networks," *Future Generation Computer Systems*, vol. 135, pp. 330–341, 2022.

[60] X. Xiao, H. Chen, X. Ma, P. Zhou, and Y. Lin, "DVNE-DRL: dynamic virtual network embedding algorithm based on deep reinforcement learning," *Scientific Reports*, vol. 13, p. 19789, 2023.

[61] Z. Yan, J. Ge, Y. Wu, L. Li, and T. Li, "Automatic virtual network embedding: A deep reinforcement learning approach with graph convolutional networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1040–1057, 2020.

[62] M. Dolati, S. B. Hassanpour, M. Ghaderi, and A. Khonsari, "DeepViNE: Virtual network embedding with deep reinforcement learning," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2019, pp. 879–885.

[63] W. Fan, F. Xiao, M. Lv, L. Han, J. Wang, and X. He, "Node essentiality assessment and distributed collaborative virtual network embedding in datacenters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 4, pp. 1265–1280, 2023.

[64] T. Wang and W. K. G. Seah, "Flagvne: A flexible and generalizable reinforcement learning framework for network resource allocation," in *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI-24)*, 2024, pp. 2407–2415.

[65] P. Zhang, Y. Hong, X. Pang, and C. Jiang, "Energy-efficient virtual network embedding: A deep reinforcement learning approach," *Electronics*, vol. 13, no. 10, p. 1918, 2024.

[66] Z. Torkamani-Azar, P. Sadeghi, K. Shahriari *et al.*, "Meta-reinforcement learning for vnep with practical constraints and uncertain events," *Engineering Applications of Artificial Intelligence*, vol. 153, p. 110820, 2025.

[67] H. Yao, S. Ma, J. Wang, P. Zhang, and S. Guo, "A continuous-decision virtual network embedding scheme relying on reinforcement learning," *IEEE Transactions on Network and Service Management*, 2020.

[68] M. Lu, Y. Gu, and D. Xie, "A dynamic and collaborative multi-layer virtual network embedding algorithm in sdn based on reinforcement learning," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2305–2317, 2020.

[69] X. He *et al.*, "Leveraging deep reinforcement learning with attention mechanism for virtual network function placement and routing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 4, pp. 1186–1201, 2023.

[70] J. Liu *et al.*, "Service function chain embedding meets machine learning: A deep reinforcement learning approach," *IEEE Transactions on Network and Service Management*, 2024.

[71] T. D. Tran *et al.*, "A deep reinforcement learning-based framework for 5g service function chain provisioning," *IEEE Transactions on Network and Service Management*, 2024.

[72] M. Sulaiman, A. Moayyedi, M. Ahmadi, M. A. Salahuddin, R. Boutaba, and A. Saleh, "Coordinated slicing and admission control using multi-agent deep reinforcement learning," *IEEE Transactions on Network and Service Management*, 2023.

[73] Y. Cai, P. Cheng, Z. Chen, M. Ding, B. Vucetic, and Y. Li, "Deep reinforcement learning for online resource allocation in network slicing," *IEEE Transactions on Mobile Computing*, vol. 23, no. 6, pp. 7099–7116, 2024.

[74] E. Ossongo, M. Esseghir, and L. Merghem-Boulahia, "A multi-agent federated reinforcement learning-based optimization of quality of service in various lora network slices," *Computer Communications*, vol. 213, pp. 320–330, 2024.

[75] H. Tu, P. Bellavista, L. Zhao, G. Zheng, K. Liang, and K. K. Wong, "Priority-based load balancing with multi-agent deep reinforcement learning for space-air-ground integrated network slicing," *IEEE Internet of Things Journal*, vol. 11, pp. 30 690–30 703, 2024.

[76] M. O. Ojijo, D. Ramotsoela, and R. A. Oginga, "Slice admission control in 5g wireless communication with multi-dimensional state space and distributed action space: A sequential twin actor-critic approach," *Computer Networks*, vol. 255, p. 110878, 2024.

[77] Y. Abiko, T. Saito, D. Ikeda, K. Ohta, T. Mizuno, and H. Mineno, "Flexible resource block allocation to multiple slices for radio access network slicing using deep reinforcement learning," *IEEE Access*, vol. 8, pp. 68 183–68 198, 2020.

[78] S. H. Haji, S. R. Zeebaree, R. H. Saeed, S. Y. Ameen, H. M. Shukur, N. Omar, M. A. Sadeeq, Z. S. Ageed, I. M. Ibrahim, and H. M. Yasin, "Comparison of software defined networking with traditional networking," *Asian Journal of Research in Computer Science*, vol. 9, no. 2, pp. 1–18, 2021.

[79] F. Ja'fari, S. Mostafavi, K. Mizanian, and E. Jafari, "An intelligent botnet blocking approach in software defined networks using honeypots," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, no. 2, pp. 2993–3016, 2021.

[80] A. Javadpour, "Providing a way to create balance between reliability and delays in sdn networks by using the appropriate placement of controllers," *Wireless Personal Communications*, vol. 110, no. 2, pp. 1057–1071, 2020.

[81] B. Isyaku, M. S. Mohd Zahid, M. Bte Kamat, K. Abu Bakar, and F. A. Ghaleb, "Software defined networking flow table management of openflow switches performance and security challenges: A survey," *Future Internet*, vol. 12, no. 9, p. 147, 2020.