A Reinforcement Learning Approach On Self-Optimizing Heterogeneous Networks

Nikolaos Prodromos
Computer Engineering and Informatics Department
University of Patras
Patras, Greece
Email: up1072549@ac.upatras.gr

Damianos Diasakos
Computer Engineering and Informatics Department
University of Patras
Patras, Greece
Email: up1084632@ac.upatras.gr

Apostolos Gkamas
Department of Chemistry University of Ioannina Ioannina, Greece
Email: gkamas@uoi.gr

Vasileios Kokkinos
Computer Engineering and Informatics Department
University of Patras
Patras, Greece
Email: kokkinos@cti.gr

Christos Bouras
Computer Engineering and Informatics Department
University of Patras
Patras, Greece
Email: bouras@upatras.gr

Philippos Pouyioutas
Computer Science Department
University of Nicosia
Nicosia, Cyprus
Email: pouyioutas.p@unic.ac.cy

***Abstract*— This paper presents a reinforcement learning-based approach for optimizing the performance of 5G mobile networks. By leveraging Deep Q-Networks (DQN), our system autonomously tunes network parameters across macro, micro, and pico cells, adapting to the dynamic distribution in a heterogeneous network environment. The agent is tasked with optimizing several Key Performance Indicators (KPIs) such as throughput, latency, interference, and Quality of Service (QoS). Each cell in the network can perform actions such as adjusting power levels, changing handover thresholds, allocating bandwidth, and performing interference mitigation. Our approach demonstrates significant improvements in user experience, resource utilization, and network efficiency over traditional static optimization methods. The results show that the proposed reinforcement learning -based algorithm not only reduces latency and interference but also ensures better load balancing and throughput optimization across heterogeneous cells.**

Keywords—Reinforcement Learning, Self-Optimizing Networks, Deep Q-Network, Multi-Cell Networks, Throughput Optimization, Latency Minimization, QoS, Interference Mitigation

I. INTRODUCTION

With the widespread deployment of 5G networks and the continuous growth in mobile data usage, managing modern network architectures (macro, micro, and pico cells) has become increasingly complex. Operators face the challenge of dynamically managing heterogeneous networks to ensure consistent Quality of Service (QoS) across diverse environments. In urban settings, users connect to different cell types with varying coverage and capabilities, making coordinated operation essential. Dynamic factors such as user demand, mobility, and environmental changes contribute to fluctuations in throughput, latency, and interference, necessitating advanced solutions for efficient network management. Traditional static network management approaches, including fixed settings for power and handover thresholds, struggle to adapt in real time. Manual optimization lacks scalability, prompting interest in Self-Optimizing Networks (SONs) that use machine learning, especially Reinforcement Learning (RL), to autonomously optimize network parameters. Prior research explored rule-based power control in small cells to improve energy efficiency, though it proved limited in dense environments. Recently, Deep Reinforcement Learning (DRL) has been used to enhance power allocation and interference management, but mainly in homogeneous or macrocell-focused networks [1],[2],[3],[4],[5].

Several studies have explored the use of machine learning in network optimization. For instance, Huang et al. [2] demonstrated the application of Multi-Agent Reinforcement Learning (MARL) for resource allocation and interference coordination in small-cell networks. Their approach showed promising results but was restricted to homogeneous network environments, leaving heterogeneous setups underexplored. Similarly, Xiang et al. [1] utilized MARL for Device-to-Device (D2D) communication power control, achieving efficiency gains in single-layer network configurations. While effective, these studies lack a comprehensive focus on multi-cell, multi-layer architectures like those in 5G heterogeneous networks. Other recent works [6], [7], [8] also applied RL to 5G scenarios, demonstrating performance improvements in aspects such as handover optimization, energy efficiency, and network slicing; however, these studies often lack integration of multiple heterogeneous cell types or simultaneous KPI optimization.

In addition to RL, hybrid approaches combining supervised learning and optimization techniques have been proposed for network management. Gao et al. [9] applied Graph Convolutional Networks (GCNs) with Long Short-Term Memory (LSTM) models to optimize resource allocation in 5G networks, highlighting the potential of deep learning in addressing complex temporal and spatial dependencies. However, such models often rely on static datasets, limiting their adaptability to dynamic environments. Our approach builds on this foundation, introducing a Deep Q-Network (DQN) to optimize multiple KPIs dynamically in a heterogeneous network environment.

Our approach addresses a multi-cell heterogeneous network where macro, micro, and pico cells interact dynamically. Unlike earlier work focused on single parameters, we simultaneously optimize multiple Key Performance Indicators (KPIs): throughput, QoS, latency, and interference. We leverage a DQN agent in a simulated 10x10 km environment with 1000 users connected across various cells. The DQN agent continuously adjusts network parameters, learning from real-time feedback to enhance network performance. The remainder of this paper is organized as follows: Section II details our reward function balancing key KPIs; Section III covers the DQN algorithm; Section IV describes the testbed environment setup, and Section V presents our performance evaluation. Finally, Section VI outlines future research directions, including user mobility and advanced RL algorithms.

II. Reward Function Analysis

The metrics used for the reward function and their mathematical and physical meaning are the following. The power level P_i of cell i represents the strength of the signal transmitted by the cell. For each cell, its P_i ranges between a P_{min} and a P_{max} which are the minimum and maximum allowable power levels for it. The power level is a critical metric that influences both the coverage area of the cell, and the interference generated in neighboring cells. Adjusting the transmission power allows the system to balance between extending the cell's coverage and limiting interference to other cells. For instance, increasing power may enhance the signal quality for users connected to that cell, but it can also increase interference for adjacent cells.

The bandwidth B_i allocated to each cell i reflects the amount of spectrum assigned to the cell for data transmission. Bandwidth allocation plays a crucial role in determining the cell's capacity to handle user traffic. By optimizing bandwidth, the system ensures that each cell efficiently serves its connected users while minimizing congestion. This allocation is particularly important in heterogeneous networks, where different types of cells (macro, micro, pico) may have different bandwidth capabilities. The bandwidth allocation can be adjusted for things like increasing the throughput for a high-demand cell or reducing the spectrum in underutilized cells to free up resources for others. Interference I is a critical metric in cellular networks as it measures the negative impact of signals from nearby cells on the users connected to cell. Interference reduces the signal quality, affecting the overall user experience and decreasing the network's throughput. The level of interference experienced by a cell depends on several factors, including the transmission power of neighboring cells and their geographical proximity. The agent can manage interference by adjusting power levels, beamforming, and other parameters. Equation 1 below shows the model for interference.

$$\sum_{j \neq i} \frac{Pj}{d^2_{ij}} \tag{2}$$

Where P_j is the transmission power of interfering cell j, and d_{ij} is the distance between cells i and j.

The Signal-to-Noise Ratio (SNR) S_i is a measure of the signal quality for users connected to cell i. It is the ratio of the desired signal strength to the background noise and interference. A higher SNR indicates better signal quality, leading to more reliable data transmission and higher throughput. The SNR depends on both the transmission power of the cell and the level of interference from neighboring cells. The SNR for one user can be calculated using the distance between the cell and the user connected as well as the signal strength of that cell. Throughput T_i is a key metric that represents the amount of data successfully transmitted to users connected to cell i per unit time. Throughput is a direct measure of the network's efficiency and user satisfaction, as higher throughput translates to faster data rates for users. The throughput of a cell depends on several factors, including its bandwidth allocation, power level, and SNR. The reward function R(s,a) seen in equation 2 is designed to balance KPIs like the ones explained above. The goal is to guide the agent (in this case, the DQN model) to take actions that optimize the overall network performance across multiple cells and users. The reward function is expressed as:

$$R(s,a) = \alpha T_{avg} - \beta I_{avg} + \gamma QoS_{avg} - \delta L_{avg} \tag{1}$$

Where T_{avg} is the average throughput of users across all cells. Throughput measures the amount of data successfully delivered to users per unit of time. Higher throughput is desirable. I_{avg} is the average interference level across all cells. Interference degrades the signal quality and negatively impacts user experience. Lower interference is desirable. QoS_{avg} is the average QoS score. QoS reflects the users' satisfaction with the network service, combining factors like data rate, latency, and reliability. Higher QoS is better. L_{avg} is the average latency experienced by users. Latency measures the time it takes for data to travel between two points in the network. Lower latency is crucial, particularly for real-time applications like video calls or gaming. The parameters α, β, γ, and δ are weights assigned to each KPI to prioritize certain metrics over others. For example, a high value of α would emphasize the importance of throughput. A high δ would prioritize reducing latency. A high β would penalize the agent more for increasing interference. These weights can be tuned depending on the specific objectives of the network (e.g., prioritize low latency for real-time applications or high throughput for bandwidth-intensive tasks). The weights α, β, γ, δ are fixed constants representing the relative importance of each KPI in the reward function. These constants are determined based on the network's priorities and the desired optimization goals. For instance, higher α prioritizes throughput, while higher δ emphasizes low latency. In this study, their values were selected to balance the trade-offs between the KPIs, ensuring the algorithm effectively optimizes throughput, latency, interference, and QoS in a heterogeneous network environment. The goal is to maximize the reward, which means maximizing throughput and QoS, minimizing latency and interference.

III. Description of Algorithm

The DQN algorithm is based on Q-learning, a RL technique. The DQN aims to learn a policy π(a□s) that maximizes the cumulative reward over time. The key idea shown in equation 3

is to estimate the Q-value of each state-action pair (s,a), which tells us how good it is to take action a in state s. The Q-value is updated iteratively using the Bellman equation, which is expressed as:

$$Q(s,a) \leftarrow Q(s,a) + \eta(r + \gamma max_{a'} Q(s',a') - Q(s,a)) \quad (3)$$

Where Q(s,a) is the Q-value for state s and action a. r is the reward received after taking action a in state ss, which is calculated using the reward function mentioned above. η is the learning rate, a factor between 0 and 1 that determines how much the Q-value should be updated. γ is the discount factor, which controls the importance of future rewards. A higher γ values future rewards more, while a lower γ focuses more on immediate rewards. s′ is the next state after taking action a in states. Equation 4 is the maximum Q-value over all possible actions in the next states.

$$max_{a'} Q(s',a') \quad (4)$$

The difference between the current Q-value and the new estimate (based on the reward and the maximum possible future Q-value) is called the Temporal Difference (TD) Error, which is calculated using equation (5):

$$TD\ Error = r + \gamma max_{a'} Q(s',a') - Q(s,a) \quad (5)$$

This TD Error represents how much the Q-value needs to be adjusted. The Q-value is then updated by adding a fraction (determined by the learning rate η) of the TD Error to the current Q-value.

Finally, the Q-value is updated using the TD: Q(s,a)←Q(s,a)+ η×TD Error.

Over time, this process refines the Q-values, leading to a more accurate estimation of the optimal policy.

The DQN uses a neural network to approximate the Q-value function, especially when the state space is large (as it is in your network environment). During training, the DQN learns which actions lead to higher rewards in different states, eventually forming a policy π(a□s) that chooses the action with the highest Q-value for any given state [10],[11],[12].

To ensure balanced exploration and exploitation, the DQN agent uses the ε-greedy exploration strategy provided by Stable Baselines3. In this approach, the agent selects a random action with probability ε, which decays over time, and the action with the highest estimated Q-value with probability 1−ε. This mechanism enables the agent to explore a wide range of actions early in training while converging to more optimal behavior as learning progresses.

The policy π(a□s) shown in equation 6 is derived by selecting the action that has the highest Q-value in any given state:

$$\pi(a \mid s) = argmax_a Q(s,a) \quad (6)$$

In other words, the agent will choose the action that maximizes the expected cumulative reward based on its current knowledge of the environment.

Algorithm 1: StableLearningCallback – Adaptive Reward-Guided Learning

Input: check_freq, learning_rate_decay_factor ← 0.9

Initialization: best_mean_reward ← −∞

Procedure:

1: **For** each training step do
2: **If** n_calls mod check_freq = 0 **then**
3: mean_reward ← average reward of last 100 episodes
4: **If** mean_reward > best_mean_reward **then**
5: best_mean_reward ← mean_reward
6: **Save** current model as "best_stable_model"
7: **Else if** mean_reward < best_mean_reward − 0.1 **then**
8: new_lr ← current_learning_rate × learning_rate_decay_factor
9: Update model learning rate to new_lr
10: **End if**
11: **End if**
12: **End for**

13: Instantiate environment ← MultiCellSelfOptNet()
14: Instantiate model ← DQN(MlpPolicy, environment, learning_rate=0.0002, verbose=1)
15: Instantiate callback ← StableLearningCallback(check_freq=1000)

The StableLearningCallback class is a custom callback designed for use in RL algorithms. It inherits from BaseCallback (a class in libraries like Stable-Baselines that allows for customization of the training process). When the StableLearningCallback is initialized, it takes two key arguments: check_freq, which specifies how often the callback should perform checks and updates, and verbose, which controls the level of detail in the printed output. Inside the initialization function, two important variables are set: best_mean_reward, which is initialized to negative infinity to track the best performance during training, and learning_rate_decay_factor, which is set to 0.9 and will later be used to reduce the learning rate if the model's performance declines. The purpose of this initialization step is to set up the conditions under which the callback will operate during the training process, ensuring that performance monitoring and learning rate adjustments can be handled dynamically.

The _on_step function is the core logic of the callback, and it is triggered at each step during training. The function first checks whether the current step count (stored in n_calls) is a

multiple of check_freq. This ensures that the callback intervenes periodically instead of at every single step, which helps in avoiding over-frequent adjustments. If the condition is met, the function computes the mean reward over the last 100 episodes by accessing the rewards from the training environment. This moving average of rewards serves as a measure of the agent's performance over time. If verbose mode is enabled, it prints the current step number and the computed mean reward to help monitor the training process. The DQN algorithm, built using Stable Baselines3, is designed to learn an optimal policy by interacting with the environment, receiving feedback (rewards), and adjusting network parameters to maximize the overall performance. The environment is initialized using the MultiCellSelfOptNet() function, which is the initializer of a class made by us, to simulate the environment and its possible actions. After the algorithm and the environment initialization the model is trained using the built-in learn() function and then its predict() built-in function is used to predict the optimal action for each state so that the reward function is maximized.

IV. Testbed Environment

The environment simulates a multi-cell wireless network consisting of 2 macrocells, 4 microcells, and 4 picocells. Macrocells provide large coverage areas but generate higher interference. Microcells offer moderate coverage, while picocells are designed for dense areas, providing high capacity but limited coverage. The cells were placed using a pseudo-random approach within a 10x10 km grid, ensuring realistic spatial distribution. This approach aims to provide balanced coverage while accounting for varying user densities and minimizing coverage gaps. The pseudo-random placement also reflects practical challenges in real-world network deployments, where geographic constraints and user demand influence cell positioning.

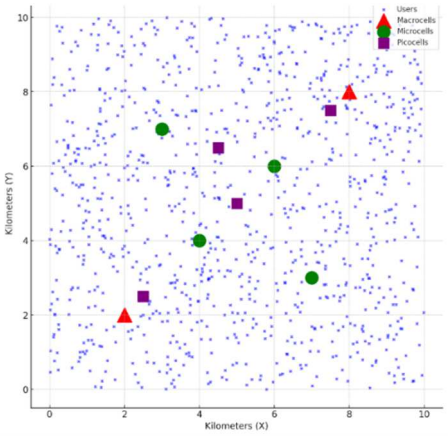

Fig. 1. Map of the area

The simulation models a 10x10 km grid with 1000 users randomly distributed across the area. Each user connects to the nearest cell (macro, micro, or pico) based on geographical proximity. Figure 1 illustrates the layout of the deployment, including the positions of all base stations and users. The observation space, or state space, is represented as a matrix where rows correspond to cells and columns represent key metrics: power level, total bandwidth, average interference, average SNR of connected users, and throughput. These metrics, described in Section II, define the network state for decision-making.

TABLE I. Initial Base Station Parameters

Cell Type	Power (Watts)	Handover Margin (db)	Bandwidth Capacity (MHz)
Macrocell 1	31	5	100
Macrocell 2	31	5	100
Microcell 1	5	3	40
Microcell 2	5	3	40
Microcell 3	5	3	40
Microcell 4	5	3	40
Picocell 1	0.25	1	15
Picocell 2	0.25	1	15
Picocell 3	0.25	1	15
Picocell 4	0.25	1	15

TABLE II. Additional System Parameters

Parameter	Value	Description
Grid Size	10 x 10 km	Represents the geographical area for the simulation.
Number of Users	1000	Users are randomly distributed across the grid.
Modulation Schemes	QPSK, 16-QAM, 64-QAM, 256-QAM	Standard 5G modulation techniques used to balance data rate and signal quality.
User Scheduling Techniques	Round-robin, Maximum Throughput, QoS-aware	Scheduling techniques implemented to balance fairness, speed, and QoS
Observation Metrics	Power, Bandwidth, Interference, SNR, Throughput	Metrics used to define the network state and guide the agent's actions.
Action Space	110	Total discrete actions available for optimization across all cells.
Cell Placement Method	Pseudo-random	Ensures realistic cell distribution, reflecting geographic and user demand constraints.

The action space includes 110 discrete actions, representing 11 possible adjustments for each of the 10 cells. Actions include adjusting power levels, changing handover threshold margins, allocating bandwidth, adjusting beamforming, activating carrier aggregation, reconfiguring cell sectors, adjusting modulation and coding schemes, enabling or disabling a cell, adjusting user scheduling, managing interference mitigation, and enabling network slicing. The agent learns to take the most appropriate action for each cell based on the current network state. Table I outlines the initial base station parameters, such as power levels, handover margins, and bandwidth capacities, which were selected to simulate a realistic heterogeneous network. Table II

provides additional details about other parameters used in the simulation, their values, and their relevance to the testbed.

The possible modulation schemes are QPSK, 16-QAM, 64-QAM and 256-QAM. The user scheduling techniques that were available for use were round-robin scheduling, maximum throughput scheduling and QoS-aware scheduling [13],[14].

V. Performance Evaluation

In this section, the algorithm is evaluated on 2000 steps. The self-optimizing network RL algorithm was trained for 100000 episodes (where one episode consists of 200 actions). After that, the algorithm was judged on 2000 actions. In Figures 2,3 and 4 the latency over time, the interference over time and the QoS over time are observed in those 2000 actions (known as steps) made by the trained algorithm and how these metrics are decreasing or increasing over time since the reward function of the algorithm works mostly to lower latency and interference and strengthen the QoS. The metrics of latency, QoS and interference are scaled in the (0,1) range. Figure 2 illustrates the evolution of average user latency (measured in milliseconds) during the 2000-step process. The primary goal of the algorithm was to optimize the network for lower latency, as minimizing delay is crucial for ensuring smooth user experience, particularly for real-time applications like video conferencing or gaming. At the beginning of the process, the latency fluctuates between 0.59ms and 0.67ms, with some noticeable spikes. These spikes can be attributed to moments when the network reconfigures critical parameters such as power levels, handovers, or user scheduling strategies. As this progresses, the algorithm learns to adjust network parameters more effectively, reducing the frequency and severity of these latency spikes. By the end, the latency stabilizes, demonstrating that the algorithm has successfully learned to minimize delay.

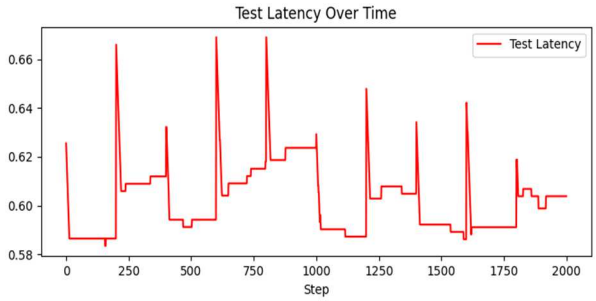

Fig. 2. Latency over 2000 steps

Figure 3 tracks the average interference levels across all cells. Interference is measured on a normalized scale, where lower values indicate less interference. In the early stages, the interference fluctuates between 0.68 and 0.64, reflecting the challenges the algorithm faces in managing network congestion and inter-cell interference. As the algorithm adjusts power levels, beamforming, and scheduling strategies, interference begins to increase, from 0.64 peaking at 0.66. This is likely due to moments when the algorithm is experimenting with higher power levels and bandwidth allocation to improve throughput and coverage. However, by the later stages, the interference decreases, indicating that the algorithm has found a better balance between cell power levels and user distribution.

Figure 4 shows how the average user experience evolves throughout the process, measured as a normalized QoS score ranging from 0 to 1. Initially, the QoS increases rapidly as the algorithm optimizes basic parameters like power and bandwidth allocation. The graph reflects a steady rise in QoS, peaking at 0.74, before encountering periodic drops as the algorithm makes trade-offs between QoS, interference, and latency optimization. Since the focus of this scenario was on reducing latency and interference, the algorithm prioritizes those goals, which can occasionally lead to a reduction in QoS as seen in the later stages. However, despite these dips, the overall QoS score stabilizes, indicating that the algorithm has managed to balance network performance effectively.

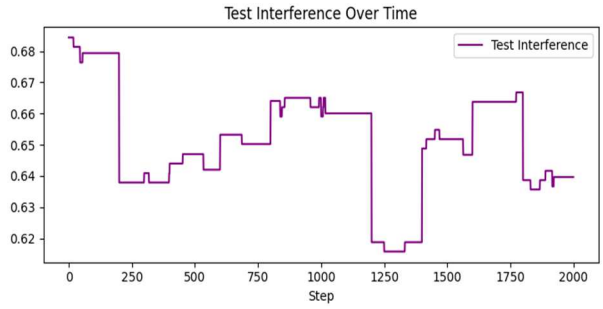

Fig. 3. Interference over 2000 steps

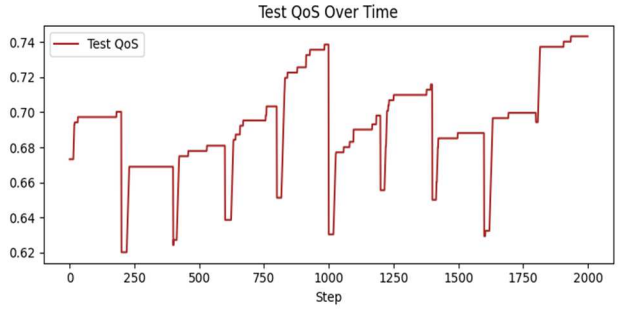

Fig. 4. QoS over 2000 steps

Fig. 5. Reward over 2000 steps

The reward over time seen in Figure 5, is a function of the above metrics, representing the overall performance of the network as the algorithm progresses. Initially, the reward fluctuates, reflecting the challenges the algorithm faces as it learns to balance the various network metrics. As we continue, the reward gradually improves while still exhibiting fluctuations. This indicates that the algorithm has found a configuration that better balances the conflicting objectives of

lower latency, reduced interference, and higher QoS. By the end, the reward shows bigger stability, indicating that the algorithm has effectively learned an efficient policy for managing the network. This progressive increase in reward with reduced variance over time also indicates that the DQN training was stable and converged toward an effective policy. As can be deduced, with minimal training, this algorithm was able to constantly maximize the reward function which depends on the QoS (which it maximized), the latency (which it minimized) and the interference (which it minimized). Further adjustments could allow it to avoid the "spikes" and to converge to optimal and adaptive solutions easier.

VI. Conclusion And Future Work

This work presented a reinforcement learning-based approach using Deep Q-Networks (DQN) to optimize the performance of self-organizing 5G heterogeneous networks composed of macro, micro, and pico cells. By simulating 1,000 users across a 10×10 km grid, the proposed model dynamically adjusted network parameters such as transmission power, handover margins, and bandwidth allocation. The results demonstrate significant improvements in latency and interference reduction while preserving high throughput and Quality of Service (QoS), validating the effectiveness of the proposed reward function in balancing conflicting optimization goals. A key strength of this approach lies in its ability to simultaneously manage multiple KPIs in a multi-layer, multi-cell network environment—an area often underexplored in prior work. However, some limitations remain. The current model assumes static user positions and does not incorporate real-world constraints such as user mobility, physical obstacles, or dynamic interference patterns. Additionally, the impact of hyperparameter sensitivity, reward function weighting, and training dynamics on DQN stability and performance has not been thoroughly analyzed. To address these limitations, future work will incorporate dynamic user mobility to simulate realistic handover scenarios and shifting interference patterns. We also aim to conduct sensitivity analysis on the reward function weights (α, β, γ, δ) to better understand how different optimization priorities affect network performance. Moreover, future work will include benchmarking against traditional baselines such as random assignment, round-robin, and static heuristic approaches to quantitatively demonstrate the performance improvements of the proposed DQN method. Expanding the reward function to include additional metrics such as energy efficiency, packet loss, and jitter will further enhance alignment with real-world KPIs. Additionally, scaling the framework to support thousands of users and hundreds of cells will be explored through decentralized learning techniques, such as Multi-Agent Reinforcement Learning (MARL). We also plan to investigate the training stability and convergence behavior of the DQN model, analyzing how exploration strategies, learning rate schedules, and replay buffer dynamics influence performance. Finally, we aim to extend the simulation by modeling 3GPP-compliant gNB behavior and evaluating the system using real-world datasets to bridge the gap between simulation and deployment.

Acknowledgment

The research project was supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the "2nd Call for H.F.R.I. Research Projects to support Faculty Members & Researchers" (Project Number: 02440).

References

[1] H. Xiang, Y. Yang, G. He, J. Huang and D. He, "Multi-Agent Deep Reinforcement Learning-Based Power Control and Resource Allocation for D2D Communications," in IEEE Wireless Communications Letters, vol. 11, no. 8, pp. 1659-1663, Aug. 2022, doi: 10.1109/LWC.2022.3170998.

[2] X. Huang, S. Leng, S. Maharjan and Y. Zhang, "Multi-Agent Deep Reinforcement Learning for Computation Offloading and Interference Coordination in Small Cell Networks," in IEEE Transactions on Vehicular Technology, vol. 70, no. 9, pp. 9282-9293, Sept. 2021, doi: 10.1109/TVT.2021.3096928.

[3] Xiao, Y., Nazarian, S., & Bogdan, P. (2019). Self-optimizing and self-programming computing systems: A combined compiler, complex networks, and machine learning approach. *IEEE transactions on very large scale integration (VLSI) systems*, *27*(6), 1416-1427.

[4] Xiong, Z., Zhang, Y., Niyato, D., Deng, R., Wang, P., & Wang, L. C. (2019). Deep reinforcement learning for mobile 5G and beyond: Fundamentals, applications, and challenges. *IEEE Vehicular Technology Magazine*, *14*(2), 44-52.

[5] Yajnanarayana, V., Rydén, H., & Hévizi, L. (2020, September). 5G handover using reinforcement learning. In *2020 IEEE 3rd 5G World Forum (5GWF)* (pp. 349-354). IEEE.

[6] F. Kavehmadavani, V. -D. Nguyen, T. X. Vu and S. Chatzinotas, "On Deep Reinforcement Learning for Traffic Steering Intelligent ORAN," *2023 IEEE Globecom Workshops (GC Wkshps)*, Kuala Lumpur, Malaysia, 2023, pp. 565-570, doi: 10.1109/GCWkshps58843.2023.10464606.

[7] A. Staffolani, V. -A. Darvariu, L. Foschini, M. Girolami, P. Bellavista and M. M. Foschini, "PRORL: Proactive Resource Orchestrator for Open RANs Using Deep Reinforcement Learning," in *IEEE Transactions on Network and Service Management*, vol. 21, no. 4, pp. 3933-3944, Aug. 2024, doi: 10.1109/TNSM.2024.3373606.

[8] V. Sciancalepore, X. Costa-Perez and A. Banchs, "RL-NSB: Reinforcement Learning-Based 5G Network Slice Broker," in *IEEE/ACM Transactions on Networking*, vol. 27, no. 4, pp. 1543-1557, Aug. 2019, doi: 10.1109/TNET.2019.2924471.

[9] X. Gao, J. Wang, and M. Zhou, "The Research of Resource Allocation Method Based on GCN-LSTM in 5G Network," *IEEE Communications Letters*, vol. 27, no. 3, pp. 926-930, March 2023, doi: 10.1109/LCOMM.2022.3224213.

[10] Yang, Y., Juntao, L., & Lingling, P. (2020). Multi-robot path planning based on a deep reinforcement learning DQN algorithm. *CAAI Transactions on Intelligence Technology*, *5*(3), 177-183.

[11] Fan, J., Wang, Z., Xie, Y., & Yang, Z. (2020, July). A theoretical analysis of deep Q-learning. In *Learning for dynamics and control* (pp. 486-489). PMLR.

[12] Oh, J., Hessel, M., Czarnecki, W. M., Xu, Z., van Hasselt, H. P., Singh, S., & Silver, D. (2020). Discovering reinforcement learning algorithms. *Advances in Neural Information Processing Systems*, *33*, 1060-1070.

[13] M. P. Mota, D. C. Araujo, F. H. Costa Neto, A. L. F. de Almeida and F. R. Cavalcanti, "Adaptive Modulation and Coding Based on Reinforcement Learning for 5G Networks," 2019 IEEE Globecom Workshops (GC Wkshps), Waikoloa, HI, USA, 2019, pp. 1-6, doi: 10.1109/GCWkshps45667.2019.9024384.

[14] Y. Cai, Z. Qin, F. Cui, G. Y. Li and J. A. McCann, "Modulation and Multiple Access for 5G Networks," in IEEE Communications Surveys & Tutorials, vol. 20, no. 1, pp. 629-646, Firstquarter 2018, doi: 10.1109/COMST.2017.2766698.