Politecnico di Torino

BSc in Telecommunications Engineering

A.a. 2024/2025

Improving User Throughput in Multi-RAT Networks Using Reinforcement Learning

Supervisors:

Doriana Monaco
María Canales Compés

Author:

Álvaro Solana Lambán

Abstract

The exponential growth in connected mobile devices has intensified the demand for efficient Radio Access Technology (RAT) selection in heterogeneous networks where users must choose between options like WiFi and LTE. Traditional approaches based on heuristics or game theory, often rely on unrealistic assumptions such as perfect information and static environments. This thesis introduces a more practical and realistic, user-centric solution based on Deep Q-Networks (DQN), proposing a Reinforcement Learning framework capable of handling dynamic and complex environments with limited information.

A custom simulation environment in Python was developed to model realistic network scenarios, incorporating user mobility and signal degradation. The proposed DQN algorithm allows each user to make independent decisions based only on local link quality while minimizing excessive switching and ensuring fairness across the system. The framework was evaluated against heuristic and RL algorithms using metrics such as throughput, switching rate, fairness, and disconnection rates.

Results demonstrate that the DQN approach consistently outperforms or matches baseline methods, despite operating under constrained information conditions. The model not only maximizes user throughput but also minimizes unnecessary handovers and ensures balanced RAT utilization, all without explicit cooperation between users. These findings show the potential of DQN as a scalable and adaptive solution for real-world network management in high-density, multi-RAT environments.

Table of Contents

List of Figures

Chapter 1

Introduction

With the exponential growth of connected mobile devices, an efficient use of the wireless infrastructure has become increasingly important. In this context, Radio Access Technology (RAT) selection becomes a critical challenge in heterogeneous networks (HetNets), where users must choose between multiple available technologies such as WiFi and LTE to ensure optimal performance. Efficient RAT selection is crucial not only for enhancing user experience, and providing higher throughput and reliability, but also for ensuring balanced network utilization and fairness among users.

In real-world scenarios, effective RAT selection plays a key role in environments such as university campuses, airports, and convention centers, where users frequently navigate heterogeneous networks composed of LTE/5G and WiFi. Moreover, in wide-area network (WAN) scenarios, where only partial knowledge of the end-to-end path is available, selecting the appropriate radio access technology becomes even more critical.

Traditionally, this problem has been approached using heuristic methods or game-theoretic formulations, where users act as rational agents aiming to maximize their own utility. These strategies often rely on fixed rules or the assumption of complete information, which do not match realistic, dynamic network environments.

Recent research has explored Reinforcement Learning (RL) techniques to address the limitations of traditional methods. By interacting with the environment, RL agents can learn optimal RAT selection policies that adapt to network changes over time. However, classic RL approaches, although effective to solve the problem at its basic level, may also fail in highly variable and dynamic environments. To address the complexity of real scenarios, the Deep Q-Network (DQN) algorithm seems promising due to its ability to handle large state spaces and learn from its observations.

In this work, a solution for the RAT selection problem is presented, by using a realistic and practical approach where the users do not have information about the other users in the network. The solution is based on a RL-based algorithm that maximizes individual performance as well as global fairness. Specifically, a user-centric DQN-based algorithm is proposed, and its performance is compared with two baseline methods from the literature: a heuristic algorithm and the Hart-RL algorithm. All models are tested in a custom-designed simulation environment in Python, that mimics realistic RAT dynamics and constraints, such as signal degradation, user mobility, and disconnection.

The goal of this work is to show that the modified DQN consistently outperforms both baseline methods in different key metrics such as throughput, fairness, managing congestion, and user disconnection rate, demonstrating the effectiveness of this approach in managing the complex and dynamic RAT selection problem.

1.1 Thesis structure

This work begins with a review of the existing literature on the RAT Selection problem, exploring various approaches such as game-theory-based solutions, heuristic methods, and reinforcement learning frameworks. The strengths and limitations of these approaches are critically analyzed. Next, the essential concepts required to understand this study are introduced, covering fundamental ideas related to Radio Access Technologies and Reinforcement Learning. Then, the RAT selection problem is examined in depth, discussing the challenges and key variables that must be considered in developing an effective solution. Following this, a detailed discussion of the proposed solution is presented, focusing on the Reinforcement Learning framework and its components. The methodology is explained thoroughly to provide a clear understanding of the approach taken in this work. After that, the effectiveness of the proposed solution is evaluated by comparing it with other existing algorithms. Various performance metrics are analyzed to demonstrate the advantages of this approach, supporting its viability as an improved strategy for RAT selection. Finally, a conclusion of this work is provided, where limitations are discussed, and future lines of investigation are proposed to continue with this research.

1.2 Project Background and Collaboration

This thesis was developed within the Dipartimento di Automatica e Informatica (DAUIN) of Politecnico di Torino. It has been supported by the European

Union – Next Generation EU under the Italian National Recovery and Resilience Plan (NRRP), Mission 4, Component 2, Investment 1.3 (PE00000001 – program "RESTART"). The work was carried out as part of the Master's Programme in Communications Engineering at Politecnico di Torino, and is intended to serve as the foundation for a future academic publication.

Chapter 2

Related Work

In the past years, many studies have been conducted addressing the problem of Radio Access Technologies (RATs) selection, and proposing different approaches to solve it. One of the first approaches that was pursued was based on game-theoretic frameworks.

2.1 Game-theory approach

The RAT selection problem can be represented as a game, either a cooperative game or a non-cooperative game, based on the collaboration between the users.

In non-cooperative games, users are modeled as agents that aim to maximize only their own utility, which can be measured in terms of throughput, signal quality, bandwidth utilization, etc. These users are competing for the same resources in a shared environment, and although their strategy affects the rest of the users, their only concern is to maximize their own utility.

On the other hand, cooperative games consist of players who collaborate to achieve a more collective allocation of the resources. Quite frequently, these games try to reach the Nash Equilibrium, a well-studied concept, where no player can benefit by changing their policy, assuming the rest of the players' policies remain unchanged.

Reaching this equilibrium or other alternative solutions to these games can be defined as a mathematical problem, and even though it can provide a solution to the problem, it presents strong limitations that need to be considered:

- Scalability Issues: As the number of users and stations grows in the scenario, computing the equilibrium becomes too complex.

- Underperformed equilibrium : Reaching the Nash Equilibrium does not necessarily maximize the total system utility.
- Movement limitations: Many game-theory models assume an scenario with static users, which may often not be a realistic approach.
- Coordination overhead: While cooperation between the users can improve their utility, it also requires a lot of communication which increases the complexity and the overhead of the system.
- Convergence Issues: In non-cooperative games, equilibrium may never be found if users are stuck in a loop where they are constantly switching RATs to obtain a higher utility.

These, among other reasons, make game theory an insufficient approach, which, although it can be used as a foundation for other solutions, requires more advanced algorithms to reach a solution.

2.2 Heuristic approach

In [1] a heuristic approach based on a non-cooperative game is proposed. In this work, users try to maximize their own throughput, while implementing mechanisms to avoid excessive switching, a common problem in multi-RAT scenarios. In this approach, users act selfishly, choosing the RAT with the highest available throughput, and connecting to it conditioned by a hysteresis mechanism that reduces unnecessary switches. However, this approach requires perfect information of all the aspects of the network, which in practice it is not realistic, as it would add a lot of overhead and complexity, especially if the number of users or stations increases.

2.3 Reinforcement learning approach

In the most recent years, with the development of machine learning, model-driven approaches were proposed as a solution for the RAT Selection problem. To be more specific, reinforcement learning presented itself as one of the best options to solve this problem, and to this day, it still is. In [2] they proposed using tabular Q-learning, with an adaptive switching probability. They also used feature learning for estimating link quality. This approach proved to accelerate convergence and improve resource utilization. However, the use of a Q-table makes it unsuitable for highly variable scenarios, where the number of possible states tends to be infinite.

In [3], a stateless Reinforcement Learning approach is proposed. Using the HartRL algorithm, they obtain a probability vector that the user will use to choose which RAT and station to connect to. These probabilities are based on past experiences to which the user was exposed. This solution is a notable simplification, which has proven effective in many scenarios. However, its performance is limited: If the user's available RATs change over time, it may not be able to adapt to these changes as it is a stateless approach.

To overcome these limitations, certain adjustments must be made to enable the algorithm to adapt to highly variable and dynamic scenarios that reflect real-world environments. Neural networks, when integrated in RL frameworks, allow the analysis of very dynamic and variable scenarios, by learning to choose the best RAT to maximize the user utility e.g. throughput, among other things. This is the approach that is followed in this work, use a RL framework that incorporates two neural networks within the algorithm to solve the RAT selection problem.

Chapter 3

Background

It is important to have a fair understanding of radio-access technologies, and Reinforcement Learning, to be able to properly comprehend this work. The basic notions that are required will be explained in this section.

3.1 Introduction to Radio Access Technologies

First of all, radio access technologies refer to the physical and link layer technologies that define how terminals connect to the network through a wireless link. The most used RATs, which are used in this work, are LTE (Long Term Evolution) which may be commonly referred to as 4G, and WiFi. Both of these technologies have undoubtedly become part of our lives. In addition, other RATs like 5G are already being incorporated but not to the extent of LTE and WiFi. Each of these RATs, although they ultimately provide a connection to the network, is designed for different purposes.

3.1.1 Comparison of LTE vs WiFi

LTE aims to offer the user a wider coverage, meaning, the same LTE station can provide connection to users in a range of hundreds of meters to kilometers. In addition, this technology offers robust mobility, allowing users to change from one cell (area covered by one LTE station) to another at a high speed while maintaining a stable connection. By using multiple LTE stations, you can design a network which could cover a huge area. Under a more formal definition, LTE is classified as a Wide Area Network (WAN) as it can provide coverage for entire cities or even larger areas like countries.

On the other hand, WiFi is designed for providing higher data rates in a smaller range of meters to a few tens of meters. Formally, it is considered a Local Area Network (LAN). It also requires less complexity and has a much lower deployment cost. However, it is much more limited in terms of mobility.

For both of these RATs, the node that provides connectivity to the network, the serving node, can be frequently called a station. However, the formal terminology is different for each RAT. In the case of LTE, the serving nodes are called cellular base stations (BS), or base stations, or LTE stations for simplicity. In the case of WiFi, the serving nodes that provide connectivity are called WiFi Access Points (APs), or APs for simplicity. In this work, when referring to a serving node, independent of the RAT that it belongs to, the term serving node or station will be used.

To avoid interference between RATS, they tend to operate in different frequency bands. For instance, LTE tends to operate in licensed bands, from 700 MHz to 2.6 GHz, while WiFi operates in unlicensed bands between 2.4 GHz and 5 GHz. This spectrum separation makes the inter-RAT interference very unlikely to occur, allowing the existence of heterogeneous networks with multiple RATs coexisting in the same area. In the case of serving nodes of the same RAT, simple but necessary techniques must be applied to avoid interference between them. In this work, frequency reuse will be implemented: By placing the stations in different areas of the scenario, they are sufficiently separated for the interference between them to be negligible.

Looking into the channel quality, in both LTE and WiFi there are mechanisms to measure and then indicate the channel conditions. In LTE there is the Channel Quality Index (CQI), which goes from 0 to 15. Higher CQI implies better signal quality and therefore the possibility to transmit at higher rates. Lower CQI implies the need for a more robust, and therefore slower, transmission. In the case of WiFi, to decide whether to choose higher rates or transmit slower but in a more robust way, the Signal to Noise Interference Ratio (SNIR) is evaluated.

Based on the channel conditions, the appropriate modulation and coding scheme (MCS) is selected. This choice directly influences transmission speed and robustness. Lower modulation implies the signal is more robust to noise, but also the transmission will be slower. Higher modulation makes it more vulnerable to noise, but is also much faster. This is a trade-off that has to be constantly evaluated based on the channel conditions.

3.2 Introduction to Reinforcement Learning

The algorithm proposed in this work is based on Reinforcement Learning (RL), which makes it of vital importance to understand the core principles of this technique.

3.2.1 Terminology and definitions

This is a standard definition of Reinforcement Learning given in [4], which is widely recognized by the literature.

"Reinforcement Learning is a framework in which an agent interacts with an environment over discrete time steps, learning to take actions that maximize a long-term reward signal."

Below, it is presented a definition of the terms mentioned in this definition, among others, which are crucial parts of a RL framework. In addition, for each definition, there is an example in the RAT selection context for a better understanding:

- Agent: The entity that takes actions in the environment. Example: The user which is moving in the network and connecting to different stations to obtain a higher throughput.
- Action(a) : A choice the agent makes to do something which will influence the environment. A represents the action space that include all the actions a. Example: One user action could be choosing to connect to a WiFi AP.
- Environment: The system where the agent operates. Example: The network where there are multiple serving nodes of different RATs, that provide a certain throughput to each user.
- State(s): Representation of the environment at a specific moment. Example: 5 users connected to WiFi and 15 connected to LTE.
- Reward(r): Evaluation of the action taken by the agent. Example: A user obtains very high reward for choosing a station that is very close.
- Reward function R(s,a): Gives the agent a reward based on the action it took in the current state. Example: Higher reward is given to the user that chooses a station that provides higher throughput, negative reward is given to the user who chooses a station that it cannot connect to.
- Policy(π): Strategy that the agent uses to decide which action to choose. Example: Choose always the station that is closer.

- Episode: A sequence of steps in the environment. In each step, there is a state, the agent takes an action, obtains a reward, that action leads to a new state. Example of a step: User was connected to station 1, and he chose to connect to station 2.
- Q-function $Q(s,a)$: Returns the expected long-term reward of taking a certain action a in the state s.
- Exploitation: The agent takes the action that it has learned yields the highest reward.
- Exploration: Taking a random action to discover new possible ways of obtaining a high reward.

3.2.2 A basic RL framework (Q-learning)

To show how these elements are linked together, and understand how Reinforcement learning works, here is the explanation of a basic Q-learning framework, which is also related to the solution proposed in this work:

First, the Agent observes the state of the environment. Then, it decides if it is going to perform exploitation or exploration. If it chooses exploitation, it will choose the action that is expected to return the highest reward in that state. It will use the Q-function, computing Q(s,a) for each possible action and choosing the one that outputs a higher value. If instead it chooses exploration, it will choose a random action. Then, the environment will give the agent a reward for that action. Based on that reward, the agent will update the Q-function.The Bellman equation (3.1) is the most popular equation for this Q-function update:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{3.1}$$

Where:

- $Q(s,a)$ is the current Q-value for taking action a in state s.
- α is the learning rate: Controls how much is new information valued over the previous experiences.
- r is the reward received for performing action a.
- γ is the discount factor, which balances the importance of long term vs short term rewards.
- s': next state resulting from action a.

- $\max_{a'} Q(s', a')$ is the maximum estimated Q-value for the next state s' over all possible actions a'.

This equation allows the agent to improve the Q-values over time, leading to a policy (π) that maximizes long-term reward. This Q-Learning algorithm can be visualized below.

Algorithm 1 Reinforcement Learning process with Q-learning

```
1: Initialize the Q-function $Q(s, a)$ arbitrarily
2: Observe the initial state $s$
3: while Learning do
4:     Choose exploration or exploitation
5:     if exploitation then
6:         $a = \arg\max_a Q(s, a)$
7:     else
8:         $a = random(A)$
9:     end if
10:    Execute $a$
11:    Obtain $r = R(s, a)$
12:    Update the Q-function(3.1)
13:    set $s \leftarrow s'$ (move to the new state)
14: end while
```

In step 4 of Algorithm 1: choosing exploration or exploitation, this is often done with an $\epsilon - greedy$ strategy. The parameter ϵ is between 0 and 1, and it is used as a probability: With probability ϵ the agent will choose to do exploration, with probability $1-\epsilon$ the agent will choose exploitation. ϵ can be a fixed value during the simulation e.g. 0.4, or can start at a high value e.g. 1 and decrease during the simulation, so that at the end the agents tend to exploit the strategies it has learned to yield higher rewards.

In classical Q-learning, the Q function $Q(s, a)$ is commonly defined as a Q-table that contains all possible state-action (s, a) pairs:

$s\backslash a$	a_1	a_2	a_3
s_1	2.5	3.1	1.8
s_2	4.0	2.2	3.5
s_3	1.1	2.9	0.5
s_4	3.3	4.1	2.7
s_5	5.0	3.8	4.2

This simple approach is very efficient in games with limited states and limited actions, but not in complex scenarios like the RAT selection problem. Although the number of actions might be limited (stations that you can choose), the number of possible states tends to infinity due to user mobility, varying channel quality, load distribution, etc. This makes the approach of using a Q-table impractical and unfeasible to implement.

3.2.3 Deep Q-Network (DQN)

Due to the limitations presented by classical Q-Learning, to solve complex problems with infinite state spaces, another way to approximate the Q-values, i.e., another way to define a Q-function $Q(s, a)$, needs to be found. One of the best options is using neural networks. The incorporation of this strategy to Q-Learning gives birth to the RL framework that will be used in this work to solve the RAT selection problem, a Deep Q-Network (DQN). This approach, although it is based on the Q-Learning framework explained before, has a few added components worth explaining:

- Primary Network: This neural network is used to approximate the Q-value function $Q(s, a)$.
- Experience buffer: Buffer used to stabilize the training of the DQN, it stores agents' experiences, i.e., (s, a, s', r) . During the learning, random batches of experiences are sampled from this buffer. This tool helps the network generalize and provide more accurate predictions.
- Target Network: A second neural network, that is used to compute the target Q-values during training. The parameters of this network are periodically updated with the weights of the primary network to ensure stability in the training.
- Loss function: It is used to measure the difference between the predicted Q-values and the target Q-values: $L = \mathbb{E}\left[(r + \gamma \max_{a'} Q(s', a') - Q(s, a))^2\right]$.

These components are required for the training and evaluation of this algorithm. The implementation of the DQN will be further explained in chapter 5, where the proposed solution by this work, which is based on a DQN, is explained.

Chapter 4

The RAT Selection Problem

Nowadays, the number of devices that demand a connection to the network is constantly increasing. For serving these devices and providing connectivity, there are multiple radio access technologies (RATs), like LTE and WiFi. In some scenarios, the increasing demand has led to dense heterogeneous networks, with multiple serving nodes from different technologies. However, the excessive densification of LTE stations and APs is ineffective if users do not select the right RAT and station to connect to. Individually, users choose stations arbitrarily, leading to lower utility. From the RATs' perspective, this results in a clear waste of resources, as users utilize them inefficiently. The solution is not to provide more options, but to distribute users across these heterogeneous networks more efficiently.

The RAT Selection problem involves choosing the optimal station within a given RAT for a user to connect to at any moment. However, is important to consider that there are many variables that can add significant complexity to this problem and that need to be optimized for finding an optimal solution.

4.1 Available information

First of all, it is important to establish how much information the user has to make decisions. Each user in the network knows at least which stations it can connect to, and the link quality of that connection by computing the Signal to Noise Interference Ratio(SNIR). Based on it, the user can estimate the rate at which it would be able to transmit in the case that it had all the bandwidth of the system available. However, if more users are connected to the same stations, and the bandwidth is divided between them, the final throughput obtained by the user will not correspond to the previously estimated rate.

To have a better estimation, we would need to know the users that are using that station, or to be even more precise, know which users are about to connect to that station. This would be a perfect information game where users have an overview of all the network, and would make solving the problem much easier. But in a real scenario, using all this information would add a lot of overhead in the network, increase its complexity, and slow it down. It would also limit the scalability of the system and make it unfeasible to implement for a large number of users and stations of different RATs. In this work, to make the system more realistic and practical to implement, it is considered that the user does not know about the number of users in the network, which stations they are connected to, or their future actions. It can only see the link quality of the potential connection to the different stations.

4.2 Users mobility

Another problem that needs to be addressed is the mobility of the users. This affects two aspects in the scenario: First, when the user moves, some stations may offer a better signal quality than others that did before, some new available stations may appear, and some others will no longer be available. Moreover, the rest of the users also move, so some stations will become more requested than before, while others become less utilized. From the user's point of view, only a bit of variability was introduced in the state of the scenario, as it can only see itself, but if we add up all the variability introduced by the rest of the users, a lot has changed from one moment to the next. This constant and unperceived variability makes it harder for the user to learn how to decide and choose the right station to connect to.

4.3 Excessive switching

Another problem which is amplified by the variability introduced by the mobility of the users, is excessive switching. It is very likely that when one user moves, another station may offer a better link quality, and without knowing anything else about the other users, he might estimate that that new station would be the best RAT. However, the user might change to that RAT and obtain a lower throughput as it is loaded with more users. Then, the users in that congested station will realize that there is another close station offering a similar link quality but probably is not as congested, so all the users switch to that one, and then the new station becomes congested and the one they were before becomes free. This switching cycle can keep happening as we are not aware of the actions of the other users.

It will not only reduce the users' throughput but also will increase the overhead in the network required by the constant handover process of changing from one station to another, or from one RAT to another. This excessive switching problem clearly needs to be addressed.

4.4 System fairness

Another aspect that needs to be addressed is the fairness of the system. The goal is to maximize the throughput for all users, which means trying to divide the available stations as fairly as possible between the users, avoiding situations where many users are placed in a congested RAT while one or two lucky users are alone in another one. The system needs to be designed so that all the users can get a fair amount of resources. However, it needs to be done without explicit cooperation between users, given the issues that it implies. This will increase the difficulty of this challenge.

To sum up, choosing the best RAT for each user at any moment is a multi-variable optimization problem that involves not only maximizing throughput, but also minimizing users' switches and maximizing system fairness. This optimization problem takes place in a scenario where users do not have any information about the rest of the users, they are not cooperating among themselves, and they are constantly moving, so both the channel quality and the stations' load are constantly changing.

Chapter 5

Solution

The proposed solution is a user-centric approach, defined as a game in which users try to maximize their own throughput. This game is modeled as non-cooperative to avoid the added overhead, complexity, and other issues mentioned before, that come with the collaboration between users. Moreover, to make the solution realistic and practical to implement, it will be an imperfect information game, in which the user lacks information about the rest of the users, their actions, the stations they are connected to, or their reward. The user only has information about the link quality between him and other stations, as would be the case in a real scenario.

The algorithm proposed to solve this game is based on a Deep Q-Network (DQN). This is a Reinforcement Learning framework where neural networks are used to approximate the Q-Values that correspond to the actions that the user can take in a given state. In a few words, the user will select the action that offers the maximum Q value in that state. Based on that action, it will get a reward that will be used to update the parameters of the neural networks, so that future estimations of the Q-values can lead to selecting actions that maximize the reward for the user. This framework will be further explained in this chapter.

5.1 Environment design

For implementing this algorithm, the environment must first be designed, along with the different components that the RL framework requires from it: The physical scenario, the state definition, the reward function, and the physical channel which is required for our use case.

5.1.1 Physical scenario

The environment consists of a 150 x 150 meters area where users and serving nodes, i.e., LTE base stations (BS) and WiFi Access Points (APs), can be found. There are 4 LTE base stations that are randomly placed close to each corner of the square. There are also 9 APs that are placed in the following way: The area is divided into equal size sections, and each AP is randomly placed inside each section. During each episode, the stations are static, i.e., their position does not change. Moreover, there are 20 users who have been randomly placed across the area in a position where they initially are able to connect to one AP and one LTE station. You can see an example of the distribution of these elements in Figure 5.1 .

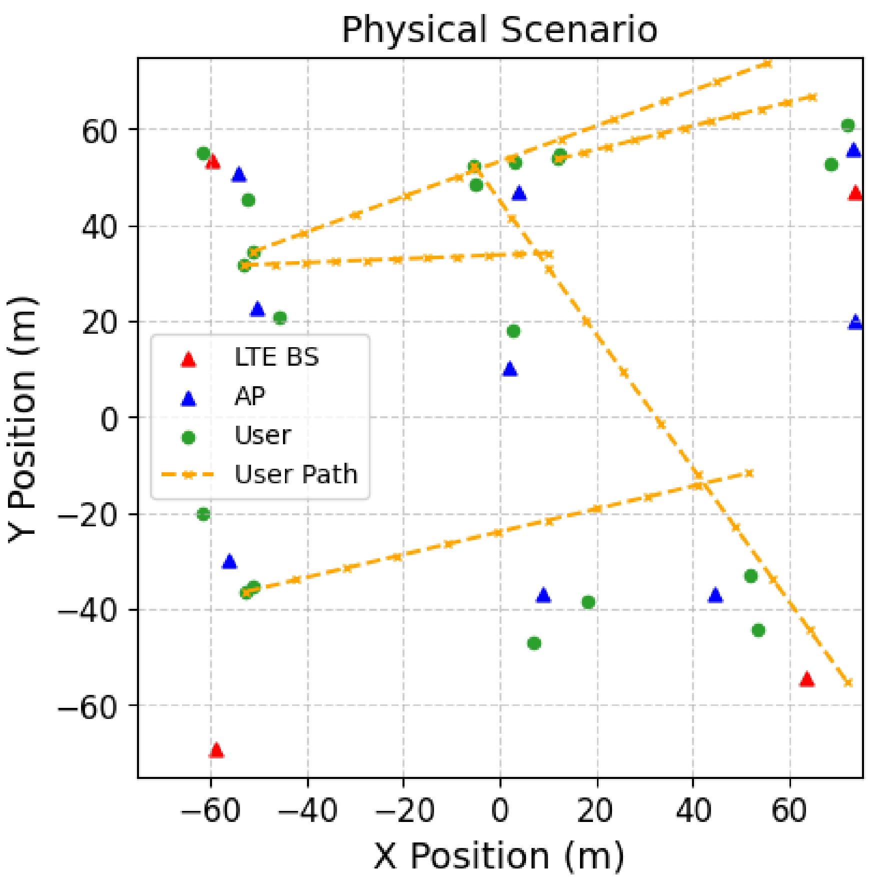

Figure 5.1: Physical scenario.

At the beginning of the episode, users will be assigned to the closest serving node, either an LTE station or an AP. During the episode, users move using the Random

Waypoint Mobility Model(RWP): Once the user is placed in the environment, it will choose a destination, which has to be at least 50 meters away from its initial position, and it will move towards it in a straight line, in 10 steps. In each step, the user will move the same distance, and choose the station to be connected to. When users reach their destination, the episode restarts.This means that the stations and access points will be again randomly placed, and the same for the users.

5.1.2 Physical channel

To estimate the link quality between a user and a serving node, the model of a physical channel is proposed, where the channel attenuation effects are based on the ones used in [2]: For the fading effect, Rayleigh Fading is proposed. Then, to model the path loss, the COST 231 Walfisch-Ikegami model is proposed, where the loss is computed with the following equation:

$$\text{Loss} = -35.4 + 20\log_{10}(d) + 20\log_{10}(f_c) \tag{5.1}$$

Where:

- d is the distance between user and station (in meters).
- f_c is the carrier frequency (in Hz).

For the LTE stations, each user has available 20 MHz of bandwidth, while for WiFi the user can use up to 80 MHz. Each RAT operates on a different carrier frequency: 2.6 GHz for LTE and 5.8 GHz for WiFi. By operating on different frequency bands, an interference-free channel between RATs can be assumed . Additionally, by placing the different serving nodes separate from each other, it can be assumed that there is not any interference between serving nodes that belong to the same RAT. Finally, there is Additive White Gaussian Noise (AWGN) added in the channel, with a power spectral density of -120 dBm/Hz. From the user's side, its transmitting power is 1 W. Finally, it is worth mentioning that the APs' range has been limited to 12 meters. All the channel parameters can be visualized in table 5.1 .

Parameter	**Value / Description**
Pathloss model	COST 231 Walfisch-Ikegami
Fading model	Rayleigh fading
Carrier frequency (LTE)	2.6 GHz
Carrier frequency (WiFi)	5.8 GHz
Bandwidth (LTE)	20 MHz
Bandwidth (WiFi)	80 MHz
Transmit power (User)	1 W (30 dBm)
Noise model	Additive White Gaussian Noise (AWGN)
Noise power spectral density	-120 dBm/Hz
Inter-RAT interference	None (different frequency bands)
Intra-RAT interference	None (frequency reused)

Table 5.1: Physical channel model parameters

Based on this channel model, the user is able to estimate the signal to noise ratio(SNR) between the serving node and the user itself. Then, for LTE those SNR values can be mapped to CQI values, and then based on table 5.2, the Modulation and Coding Scheme(MCS) that corresponds to that CQI value can be selected.

CQI	**MCS**	**Spectral Efficiency (bits/Hz)**	**SNR (dB)**
0	Out of range	-	-
1	QPSK, 78/1024	0.1523	-9.478
2	QPSK, 120/1024	0.2344	-6.658
3	QPSK, 193/1024	0.377	-4.098
4	QPSK, 308/1024	0.6016	-1.798
5	QPSK, 449/1024	0.877	0.399
6	QPSK, 602/1024	1.1758	2.424
7	16QAM, 378/1024	1.4766	4.489
8	16QAM, 490/1024	1.9141	6.367
9	16QAM, 616/1024	2.4063	8.456
10	16QAM, 466/1024	2.7305	10.266
11	16QAM, 567/1024	3.3223	12.218
12	16QAM, 666/1024	3.9023	14.122
13	16QAM, 772/1024	4.5234	15.849
14	16QAM, 873/1024	5.1152	17.786
15	16QAM, 948/1024	5.5547	19.809

Table 5.2: 3GPP CQI to MCS and spectral efficiency mapping for LTE

The MCS selected provides a certain spectral efficiency. To estimate the available data rate in an LTE system, the following equation can be used:

$$\text{Rate} = \text{Bandwidth} \times \text{Spectral Efficiency} \tag{5.2}$$

In the case of WiFi, a similar procedure is followed, but in this case using a different mapping that can be seen below in Table 5.3.

MCS Index	Modulation	Spectral Efficiency (bps/Hz)	SNR (dB)
0	BPSK	0.5	-3.83
1	QPSK	1.0	0
2	QPSK	1.5	2.62
3	16-QAM	2.0	4.77
4	16-QAM	3.0	8.45
5	64-QAM	4.0	11.67
6	64-QAM	4.5	13.35
7	64-QAM	5.0	14.91
8	256-QAM	6.0	17.99
9	256-QAM	-	-

Table 5.3: MCS's and required SNR values in IEEE 802.11ac

Following this procedure, each user is able to estimate the available rate for each of the serving nodes of the environment. It is important to remember that this rate is just an estimation, and it may not correspond to the actual throughput that the user gets. This will depend on the number of users that are connected to the same station, which the user is not aware of. This estimation, although sometimes it might be inaccurate, it has proven to be very useful.

5.1.3 State definition

The state is defined as the stack of the available information that each of the users has. This information is divided in 3 features:

- Available Rate: A list of the estimated rates that each station offers to the user based on the link quality. This rate will be normalized following the min-max normalization:

$$\hat{r}_i = \frac{r_i - r_{\min}}{r_{\max} - r_{\min}} \tag{5.3}$$

  where:

  - r_i is the estimated rate from station i.
  - $r_{\min} = \min_j r_j$ is the minimum rate a user can achieve.

- $r_{\max} = \max_j r_j$ is the maximum rate a user can achieve.
- $\hat{r}_i$ is the normalized rate for station i.

If r_i is 0, meaning the user is not able to connect to station i, in the state of the environment the rate will be indicated as $r_i = -0.1$, so that the algorithm learns to assign that negative value with a station to which it cannot connect, and therefore avoids choosing it.

- Station id : This is a one-hot vector where it is marked with 1 the station i to which the user is currently connected. It is important to track in which station the users are at every step, in order to address the problem of excessive switching.
- Rat type: In this vector, the LTE stations are marked as 0 and the APs as 1.

The length of each of these features is the number of stations, and it is ordered by having the LTE stations first, and then the APs. The element i of each feature corresponds to the same serving node. Below you can see an example of the state of one user:

$$\text{State} = \underbrace{[0.7,\ -0.1,\ -0.1,\ 1,\ -0.1, ...,\ 0.8,\ -0.1,\ 1.0,\ -0.1,\ -0.1]}_{\text{Available Rate}}$$

$$\|\ \underbrace{[0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 1,\ 0,\ 0,\ 0,\ 0]}_{\text{Station ID}}$$

$$\|\ \underbrace{[0,\ 0,\ 0,\ 0,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1]}_{\text{RAT Type}}$$

Finally, to create the environment state, all the states of all the users are stacked.

5.1.4 Reward function

As we have seen in the background section, the reward function is a key part of every RL framework. It is how the actions of the agent are evaluated. In this case, the actions correspond to the stations chosen by the user. The reward for each user is based on the throughput that it obtains. This throughput depends mainly on two things: The available estimated rate computed by the user, and the number of users that are currently connected to the station it has chosen. If the user is alone in the station, the throughput equals the rate, but if there are more users sharing that station, the actual throughput that the user will receive needs to be computed. This computation depends on which RAT the users are connected to, LTE or WiFi. The throughput models that have been used for each technology are widely recognized by the literature, and used in [1],[2],[3]. For the case of LTE, a *Proportional-Fair Model* is proposed, where the throughput obtained by user j choosing the BS i is given by:

$$\bar{T}_j^i = \frac{R_j^i}{n_i} \tag{5.4}$$

where:

- $\bar{T}_j^i$ is the average throughput obtained by user j on BS i.
- R_j^i is the estimated rate of user j on BS i.
- n_i is the number of users connected to BS i.

For WiFi, the model is a *Throughput-Fair Model*, in which all the users connected to the same AP have the same throughput. The throughput obtained by a user j connected to AP i is expressed as:

$$\bar{T}_j^i = \left(\sum_{j=1}^{n_i} \frac{1}{R_j^i} \right)^{-1} \tag{5.5}$$

where:

- $\bar{T}_j^i$ is the average throughput obtained by user j on AP i.
- R_j^i is the estimated rate of user j on AP i.
- n_i is the total number of users connected to AP i.

Once the throughput is obtained, the reward can be estimated by normalizing the throughput as it was previously done with the rate. If the station that the user chose cannot provide any connection, i.e., the estimated rate was 0, the reward will be equal to -0.1.

Even though one of the goals of the user is to maximize the throughput, this is not the only measure of a good RAT selection. One of the other key factors in this problem is minimizing the number of switches. Specially, the ones between different RATs which have the higher cost to the network. To deal with this, the reward function is adapted: When the user changes from one RAT to another, and obtains less than a 10% increase in the throughput, this change is considered unnecessary and therefore penalized: The user loses 10% of the expected reward. Later in the evaluation section, we will see how this clearly reduces the number of unnecessary switches.

5.2 Neural networks

Once the environment is ready, neural networks are needed for the agent to interact with it. The algorithm proposed is a DQN, so it requires 2 neural networks: the primary network and the target network. The primary network will process the state of each user and choose the best RAT. On the other hand, the target network, has a very important role in adding stability in the training and achieving convergence. This network is a copy of the primary network and contains the target Q-values. Both these neural networks have been designed with the parameters shown in Table 5.4:

Component	**Description**
Input dimension	$n_{\text{stations}} \times 3$ (User state)
Number of hidden layers	1
Hidden layer size	512 neurons
Activation function	ReLU (after linear layer)
Normalization	Layer Normalization after linear layer
Output dimension	n_{stations} (Q-values for each station)
Optimizer	Weighted Adam
Learning Rate	$2.5e-4$
Loss function	Mean Squared Error(MSE) Loss

Table 5.4: Architecture and parameters of primary and target networks

5.3 Reinforcement learning framework

In this section, the framework for training these neural networks is discussed in more detail. First of all, the training required 100,000 steps. In each step, first it is decided whether to do exploration or exploitation. To make this decision, an ε-greedy policy is used. ε starts at 1 and linearly decays to 0.05 in the first half of the simulation. This parameter represents the probability of performing exploration. When this happens, a random action is chosen for each user, meaning that each user is assigned to a random station. In case of exploitation, the environment state is given as input to the primary network, which will output an estimation of the Q values. Then each agent will choose the action that has the highest Q value assigned. With the actions, the algorithm takes a step in the environment, i.e., connects the users to the stations that they have chosen based on the actions. Then each user obtains the corresponding reward, and the state is updated, which includes moving the users to the next position of their path. This transition composed by the previous state, future state, actions, and rewards (s, s', a, r) will be stored in an experience buffer that can store up to 10,000 different experiences. When the buffer is full, new experiences will replace old ones.

If the training is still in the first steps of the simulation, the first 10,000 steps, where the buffer is not full yet, it will keep storing experiences until it is, then the learning can start. When the learning phase starts, in every step the algorithm takes a batch of 128 samples from the buffer, and uses them to update the weights of the network with the following procedure: First, the target Q value is obtained using the target network and following the expression below:

$$Q_{\text{target}} = r + \gamma \cdot \max_{a'} Q(s', a') \cdot (1 - done) \tag{5.6}$$

where:

- Q_{target} is the target Q value.
- r is the reward received from the environment.
- γ is the discount factor, which controls the importance of future rewards.
- s' is the next state.
- a' are the possible actions in the next state.
- $\max_{a'} Q(s', a')$ is the maximum Q value for the next state, across all possible actions.
- *done* is an indicator of whether the episode has ended or not(1 if it has, 0 otherwise).

Then, the algorithm computes the current Q-value using the primary network, and compares it with the Q target value to compute the loss.

$$Q_{\text{old}} = Q(s, a) \tag{5.7}$$

$$\text{Loss} = \frac{1}{N} \sum_{i=1}^{N} (Q_{\text{target},i} - Q_{\text{old},i})^2 \tag{5.8}$$

where:

- $Q(s, a)$ is the estimated Q-value for state s and action a.
- Loss is the Mean Squared Error (MSE) loss function.
- N is the number of experiences in the batch.
- $Q_{\text{target},i}$ is the Target Q-value for experience i.

- $Q_{\text{old},i}$ is the Current Q-value for experience i.

Once the loss is computed, it performs backpropagation to update the weights of the network.

$$\theta \leftarrow \theta - \alpha \nabla_\theta L \tag{5.9}$$

where:

- θ are the parameters of the primary network.
- α is the learning rate.
- $\nabla_\theta L$ is the gradient of the loss with respect to the parameters.

Another important step in the learning phase is to update the parameters of the target network, which is done following equation (5.10).

$$\theta_{\text{target}} \leftarrow \tau\theta + (1-\tau)\theta_{\text{target}} \tag{5.10}$$

where:

- θ_{target} are the parameters of the target network.
- θ are the parameters of the primary network.
- τ is the update rate.

An overview of the complete algorithm is shown on the next page(Algorithm 2).

Although the approach of this solution is non-cooperative, and the user only takes decisions based on its own information, in the training the loss is computed based on all the actions from all the users. Therefore, although in the implementation the user is alone when taking the decision, it has previously been trained with a global loss function where it has indirectly learned a policy that maximizes not only its own reward but also the reward of the rest of the users. In the next chapter, we will see how this global computation of the loss has served the performance of this algorithm, not only in maximizing the throughput for the user, but also to optimize the fairness of the system.

Algorithm 2 DQN Algorithm for RAT Selection

```
1:  Initialize environment, neural networks and experience buffer
2:  Set initial exploration rate $\epsilon \leftarrow 1$
3:  for each global step $t = 1$ to $sim\_steps$ do
4:      Update exploration rate (linear decay)
5:      if random number $< \epsilon$ then
6:          $a = random(A)$
7:      else
8:          $a = \arg\max_{a} Q(s, a)$
9:      end if
10:     Execute $a$
11:     Observe $s'$, $r$, and termination flag $done$
12:     Store $(s, s', r, done, a)$ in the buffer
13:     if buffer is full then
14:         $(s, s', r, done, a)$ = buffer.sample(batch)
15:         Get target values: $Q_{\text{target}}(s, a) = r + \gamma \max_{a'} Q(s', a')$
16:         Get current values $Q_{\text{current}}(s, a) = Q(s, a)$
17:         Compute loss $L = \frac{1}{N} \sum_{i=1}^{N} (Q_{\text{target},i} - Q_{\text{current},i})^2$
18:         Update primary network parameters $\theta \leftarrow \theta - \alpha \nabla_\theta L$
19:         Update target network parameters $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$
20:     end if
21: end for
```

Chapter 6

Evaluation

To evaluate the proposed solution, several metrics have been measured to validate the algorithm performance. These metrics are the following: The average throughput of the users over time, the fairness of the system over time, the number of switches, the congestion of the stations, the distribution between RATs, and the percentage of users that are disconnected. This evaluation will take place in Python, using an environment with the same characteristics as the one where the DQN was trained.

The performance of the proposed solution will also be compared to 2 other algorithms proposed in the literature to solve the RAT selection problem. The first algorithm was proposed in [1]. It is a non-cooperative game where they used the heuristic approach to solve the RAT Selection Problem. This algorithm has the same goal: the user tries to maximize its own throughput, and also tries to reduce the number of switches taking into account how much throughput it gains by switching to another RAT. However, this approach is based on a perfect information game, where the users already know the throughput they will obtain if they join another RAT or connect to a different station based on the number of users that are already connected. Also, for controlling the number of switches, they implement a mechanism that depends on the actions of other users, meaning that every user knows the actions of the other users and also to which station they are connected. All this information gives a significant advantage to this algorithm for choosing the best possible RAT. The goal will be to match or outperform it with the DQN, which is a more realistic approach, and therefore the information that the users have is much more limited. An overview of this heuristic algorithm can be seen below in Algorithm 3.

Algorithm 3 Heuristic RAT Selection Algorithm

```
Require: Client $i$'s parameters: $\eta, T, p, h$, Set of RATs.
Ensure: Decision to switch and selected RAT
 1: for each RAT $k'$ do
 2:     if $\frac{\omega_{i,k}[t+1]}{\omega_{i,k}[t]} > \eta, \forall t = t - T + 1, \ldots, t$ then
 3:         if class($k'$) = class($k$) then
 4:             if rand < $p^{m_i+1}$ then
 5:                 Switch to $k'$
 6:                 if concurrent move then
 7:                     Increment $m_i$
 8:                 else
 9:                     Reset $m_i$ to 0
10:                 end if
11:             end if
12:         else
13:             if $\omega_{i,k} > h[\text{class}(k')]$ then
14:                 if rand < $p^{m_i+1}$ then
15:                     Switch to $k'$, $h[\text{class}(k)] \leftarrow \omega_{i,k}$
16:                     if concurrent move then
17:                         Increment $m_i$
18:                     else
19:                         Reset $m_i$ to 0
20:                     end if
21:                 end if
22:             end if
23:         end if
24:     end if
25: end for
```

The DQN will also be compared to another algorithm proposed in [3], where they propose to aid a HartRL-based algorithm with feedback from the network. This feedback would be equivalent to the reward given by our environment, which can help the user improve future choices. This algorithm requires less information than the DQN, as it is a stateless algorithm that chooses the RAT based on a probability vector obtained from previous experiences. Although it has been proven to effectively solve the RAT selection problem, it is a very lightweight solution, and the environment used in this work is very dynamic, contains a lot of randomness, and not all stations are available at all times. This algorithm may not be as strong as the DQN in addressing all these realistic issues. An overview of this algorithm is shown below in Algorithm 4.

Algorithm 4 HartRL With Network-Assisted Feedback

```
Require: Set of available RATs $S_A$
Ensure: Selected RAT and decision to switch
 1: Exploration: Each user $A$ sequentially explores all available choices $j \in S_A$
    to learn possible payoffs and feedback from potential RATs.
 2: Initialization: Generate random uniform probability $p_1(j)$ for all $j \in S_A$.
 3: for each global step $t = 1$ to sim_steps do
 4:     Action Selection: Select action $i_t = j$ according to the probability
    distribution $p_t(j)$.
 5:     Feedback Exchange: Obtain feedback $Y_t^j$ from the corresponding base
    station $j$.
 6:     Regret Update: For all $k \neq j \in S_A$:
 7:             • Update the user estimated regret $B_t(j, k)$.
 8:             • Update the network measured regret $Y_t(j, k)$.
 9:     Strategy Update: Update $p_{t+1}(k)$
10: end for
```

The DQN and these two algorithms will be evaluated by running a simulation with Python of 100 episodes, which means 1000 steps. These simulations will be executed 10 times, using different seeds in the random generators. Then, the average of the results will be computed, and the 95% confidence interval will be reported to provide a more statistically robust comparison. The actions that the users took, and the rewards that they received during this simulation, will be collected in order to extract from them the information needed to analyze the algorithms' performances.

6.1 Throughput evaluation

Below, in figure 6.1, we can visualize the average throughput aggregated every 5 episodes, for the three algorithms. The first noticeable point, is that the HartRL is not able to keep up with the dynamicity and randomness of this scenario, and therefore the throughput drops to 20 Mbps. Moreover, comparing the DQN with the Heuristic approach, their throughput is very similar. However, it is important to remember that the Heuristic algorithm has all the information from the environment, which represents an ideal but not realistic advantage. Nevertheless, the DQN is able to match or even outperform sometimes this algorithm even if the user has no knowledge about the rest of the network.

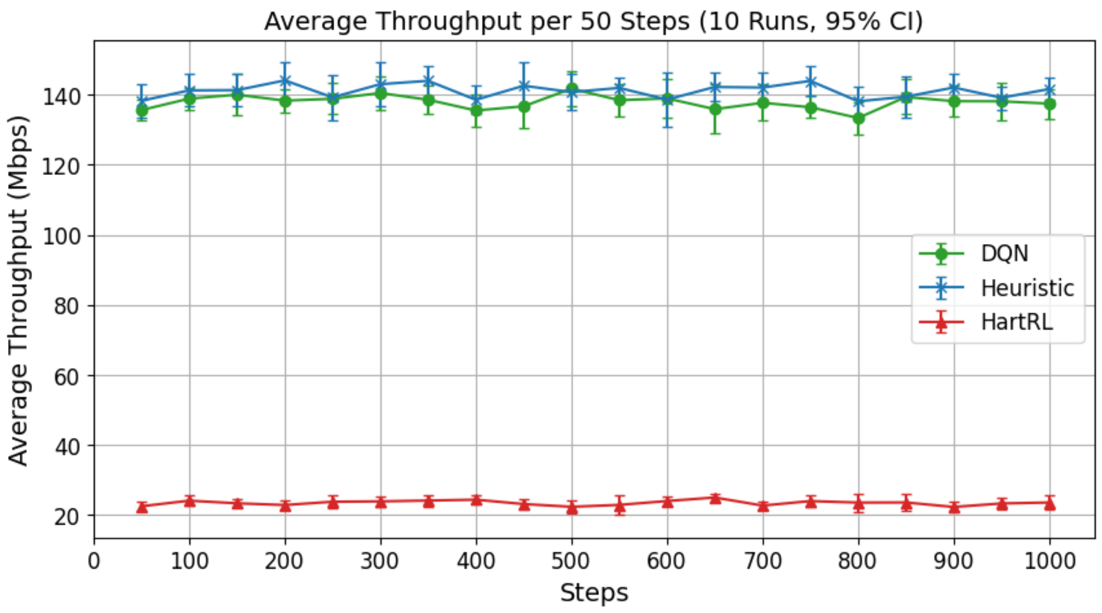

Figure 6.1: Throughput comparison between algorithms.

6.2 System fairness

Another important aspect of the RAT selection problem is offering all the users in the network a fair share of the resources, so that all users can experience a fairly equal quality of service.The fairness of the system will be measured using a widely recognized metric, the Jain Index, which is defined as follows:

$$J = \frac{\left(\sum_{i=1}^{N} x_i\right)^2}{N \sum_{i=1}^{N} x_i^2} \tag{6.1}$$

Where:

- J is the Jain's fairness index.
- x_i is the resource allocation for user i.
- N is the total number of users.

The Jain index ranges from 0 to 1, where 1 indicates perfect fairness. The fairness of the system will be based on the throughput obtained by the user. We saw that the average throughput was high, however, the goal is for it to be as distributed as possible among users. It is important to mention that due to the variability of the scenario, different user distributions, randomly placed stations, etc, it is impossible to reach perfect fairness, nor is it the goal of this work, as it would not be realistic. The goal is to obtain a fair system even if the users are not cooperating among themselves, and they do not have any knowledge about the other users. In the training phase, the loss function considers all the users so that the final policy of each user would maximize not only its own reward but indirectly also the overall system reward.

In Figure 6.2, we can see the average fairness computed in each episode, which is then aggregated using the average over 5 episodes (i.e., 50 steps).

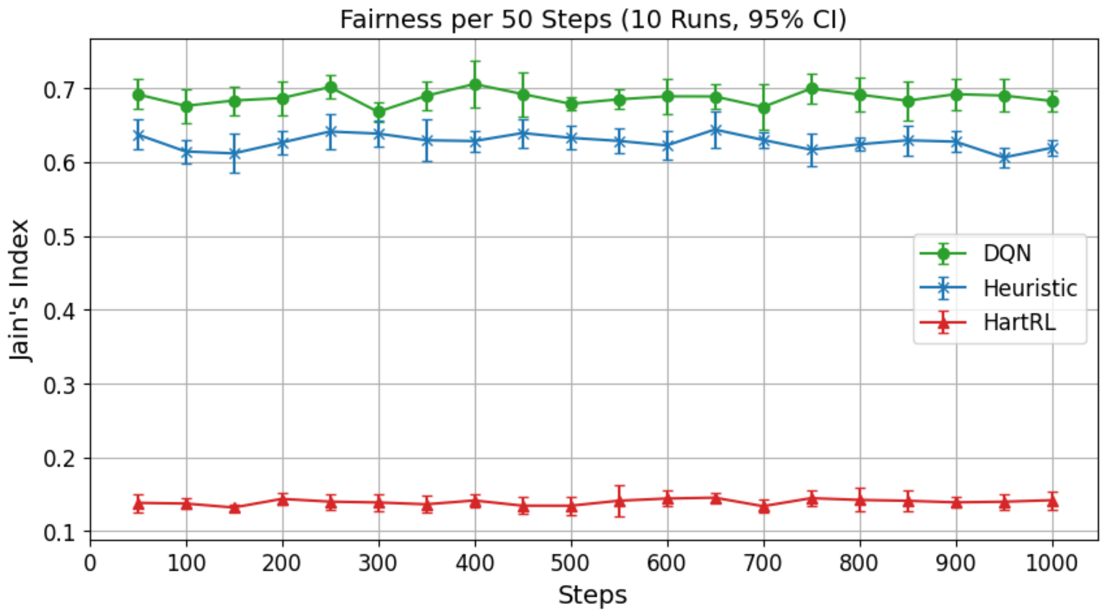

Figure 6.2: Average fairness of the system over time.

The DQN outperforms again the HartRL, and this time clearly outperforms the Heuristic approach. The fairness has been computed every episode, which is a small time scale, just 10 steps, which makes it difficult to obtain higher fairness values, and may give the impression it is not a fair system. To reassure the fairness of the system, you can see in Figure 6.3 the average throughput of each user in the complete simulation. As we can see, in the long term the DQN offers a fair distribution of the throughput among the users, which is also reflected in the Jain index of the overall simulation.

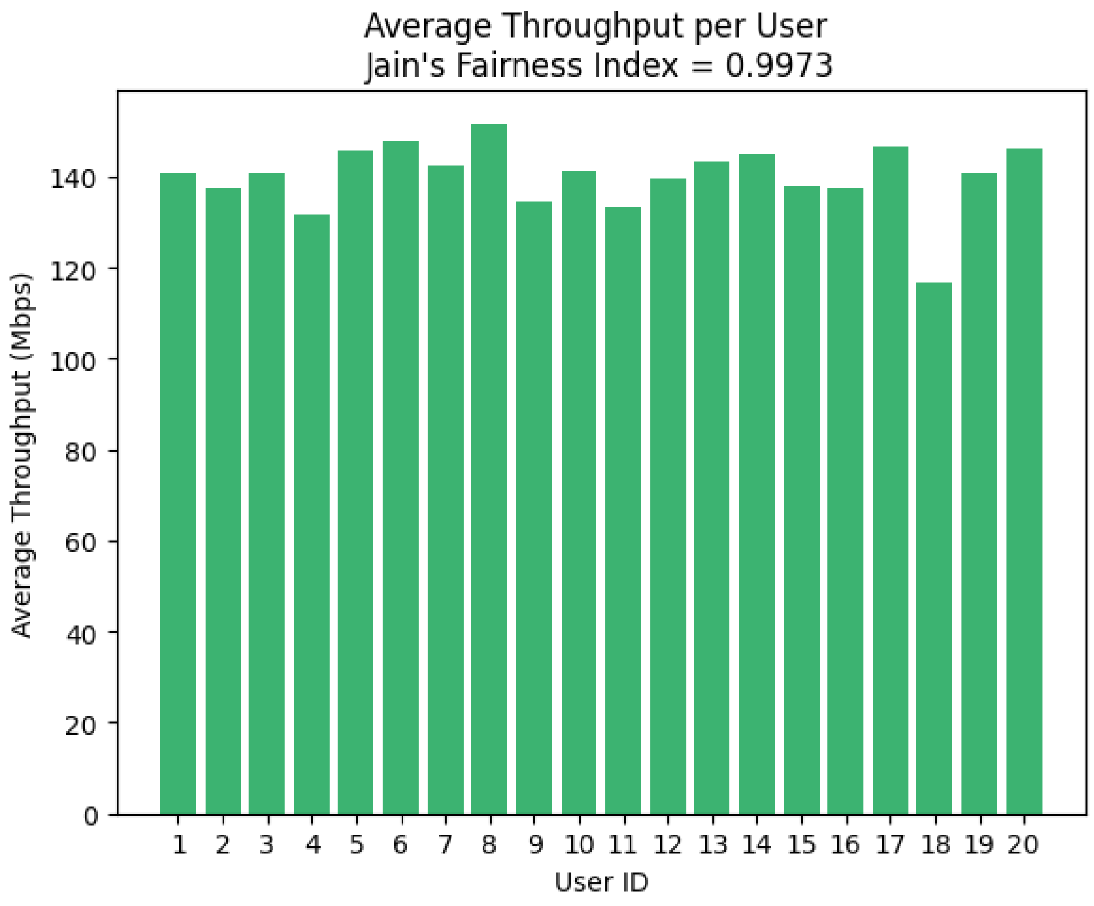

Figure 6.3: User's throughput across the simulation using the DQN.

6.3 Switching frequency

In this section, it is demonstrated how the proposed DQN reduces the number of switches, reducing the cost of performing these handovers, which cause delay, added overhead, etc, reducing the quality of service experienced by the user. The HartRL did not implement any mechanism to avoid switches, which may have not been necessary for a more static and stable scenario, but as we have seen, this does not apply to our dynamic environment. On the other hand, the Heuristic approach implemented a mechanism to reduce the number of switches, based on evaluating switching gains, i.e., the difference in throughput obtained when switching to a new station. They also added a probability of switching to be applied even if the switching gains were positive. In this work, the strategy was to modify the reward function of the DQN to penalize these switches between RATs. As we can see in

Figure 6.4, it has proven to be very effective.

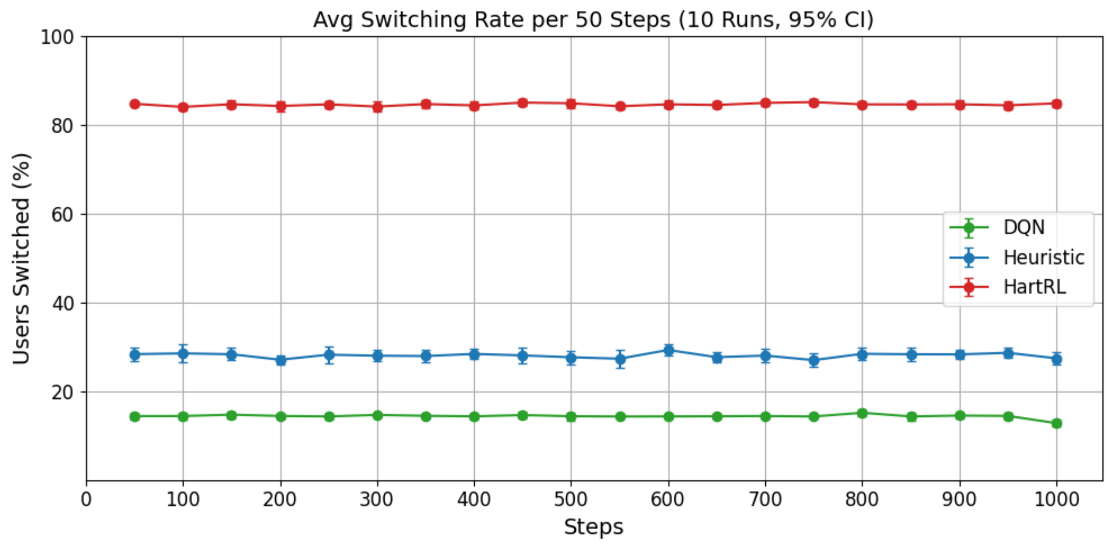

Figure 6.4: Percentage of users that switch stations.

In the graph above, we see that in every step, on average, less than 20% of the users change station, which proves how the proposed solution is able to properly solve the excessive switching problem effectively. It is important to point out that users are still going to switch stations when they move in order to select the best RAT. The goal is to minimize unnecessary switches like the ones we can see in the HartRL where almost 90% of the users switch every step.

6.4 Disconnection rate

One issue in the RAT selection problem is mistakenly choosing a station that the user cannot connect to, or failing to adapt quickly enough to movement by not switching to another station when the one it is currently using cannot provide a connection anymore. These two requirements: good decision making, and high adaptability to the users' mobility, can be measured by looking at the percentage of times users were disconnected from the network. As we can see in Figure 6.5, the DQN meets all the specified requirements as no user was ever disconnected from the network. The same cannot be said for the other approaches.

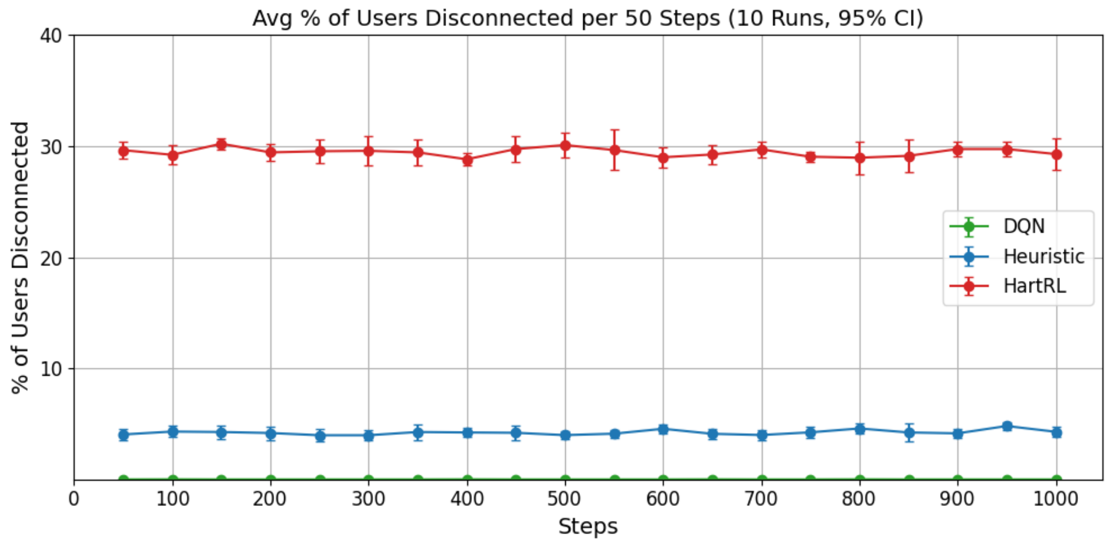

Figure 6.5: Percentage of users that are disconnected from the network.

In the HartRL, around 30% of the users are disconnected at every iteration. This is due to the fact that not all stations are available at all times. As this is a stateless algorithm, it does not adapt properly to highly variable states where the available stations are constantly changing.

In the case of the heuristic approach, around 5% of the users are not able to connect to the network. The problem with this algorithm is that the probability of switching may sometimes keep users from switching even when they really need it, because it does not make any distinction for the cases in which the user may be about to lose the connection to its current station.

Also referencing the previous section where the amount of switches was measured, this disconnection graph shows that the heuristic algorithm is not switching enough. It would require more switches to avoid disconnections and properly work, even though the users are already switching 10% more than in the DQN. This shows once again how efficient the switching of the DQN is, switching always if needed but avoiding a lot of unnecessary switching.

6.5 RAT distribution

In this section, it is evaluated whether or not the algorithm has learned to choose both LTE Stations and WiFi APs, and check if the load of users between these two technologies is more or less balanced. In the environment, these technologies offer different advantages: WiFi APs offer higher rates, but their range is limited to 12 m. On the other hand, LTE stations offer lower rates but higher coverage. In an optimal solution, it would be expected for the user to balance these 2 technologies to maximize the advantages that each provides and maximize the overall reward. In Figure 6.6, we can see that the DQN meets our expectations and properly adapts to the environment, learning to use both RATs.

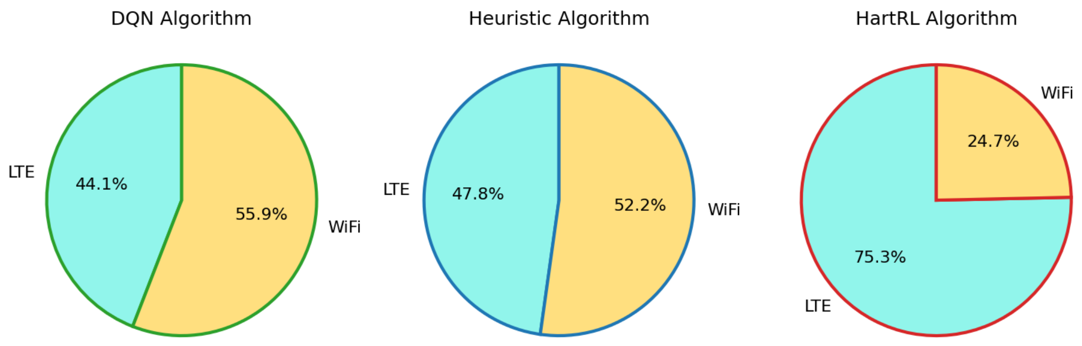

Figure 6.6: Percentage of users in each RAT.

The HartRL shows an example of a situation of suboptimal convergence. LTE stations offer a wider coverage than WiFi APs, so if the user mostly connects to LTE stations, it will be easier to avoid disconnections. However, this will also limit the throughput, as the LTE stations will be more congested. Additionally, users will not take advantage of the potential higher throughput that the APs can provide.

6.6 Congestion analysis

It is also important to ensure that the network is not congested, meaning that the users are fairly distributed across different stations and not connected to the same one. To check it, we can visualize the stations that have the most users connected over time, to visualize how much of the network total load is being taken by a single station. If we check Figure 6.7, we can see that in the worst case, the load supported by one single station will not be more than 25% of the users, and the rest will be distributed across the other stations. There will not be any case of heavy congestion where a high percentage of the users are connected to the same station, which would obviously degrade the throughput obtained by the user. Comparing it with the other different algorithms, we can see that the DQN has approximately the same behavior as the Heuristic approach, and again a better behavior than the HartRL approach.

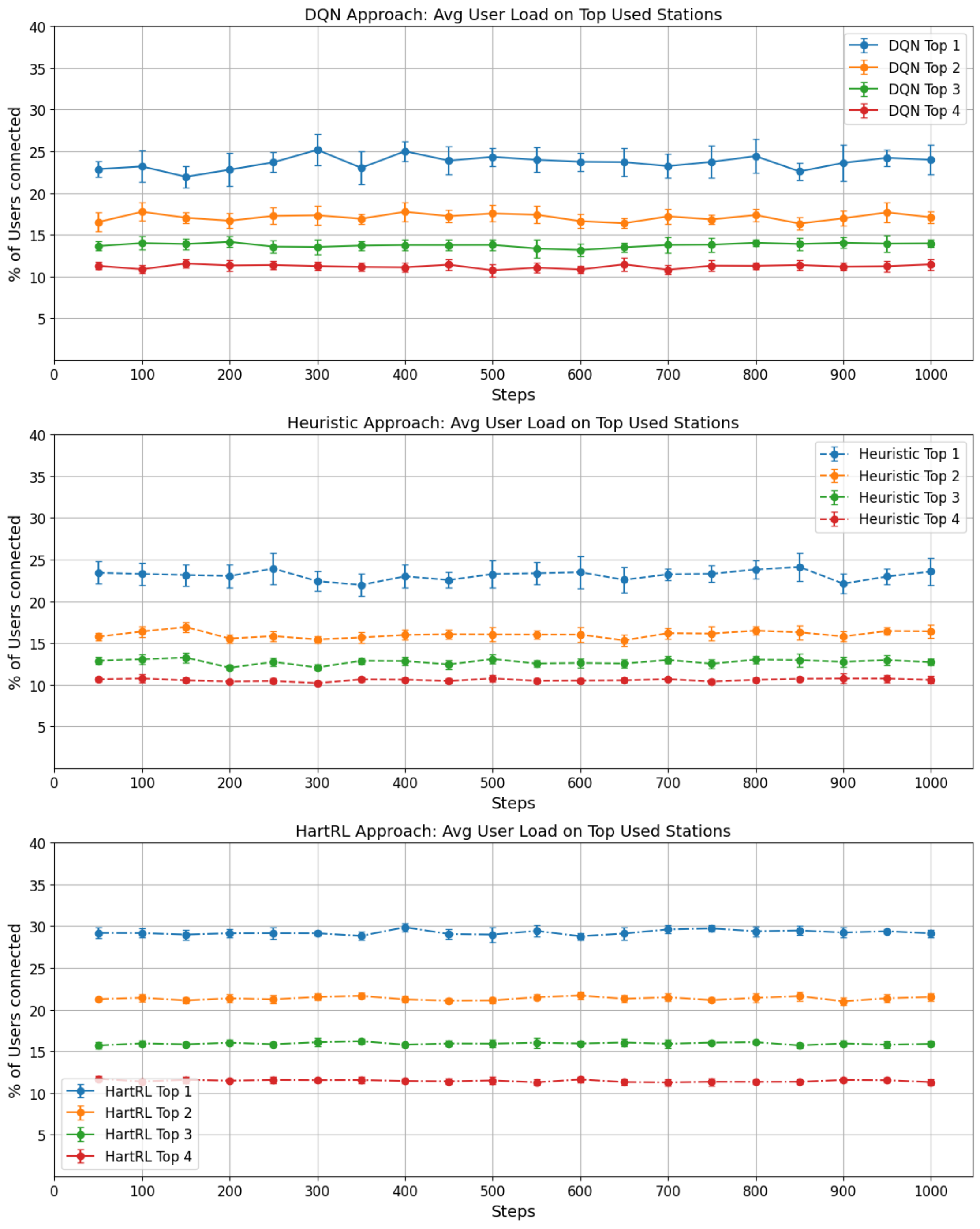

Figure 6.7: Percentage of users in top used stations.

6.7 Initial attempt and alternative approach

Before implementing the DQN, a different strategy was attempted to solve the RAT selection problem: using RL with neural networks to achieve the Nash Equilibrium. This involved training the agents to reach a policy where no individual user could unilaterally change their strategy and obtain better results. Mathematical formulations proposed in [5] were integrated into the loss function to enforce this equilibrium. However, this approach was ultimately abandoned for several reasons:

- **Continuous action space mismatch:** The Nash Equilibrium equations were designed for a continuous action space. However, in our environment, the action space is discrete (e.g., selecting station 1, 2, ...). While the network could be modified to output a continuous value between 0 and N stations, and then round the output to choose a station, this strategy proved suboptimal. Stations are represented by numbers, but these numbers have no real order or distance e.g station 2 is not physically "close" to station 1 or has any relation with it. In addition, normalizing these values between 0 and 1 led to very small numerical differences between stations, requiring high output precision from the model. These factors made learning more difficult and unstable.
- **Convergence to suboptimal behavior:** Even after tuning the network, the loss eventually converged and the algorithm did learn. However, the learned behavior was suboptimal: users mostly selected LTE stations due to their higher coverage stability. This represented a local minimum that satisfied the loss function mathematically but did not reflect meaningful or balanced behavior. Alternative attempts to adapt the equations to a discrete action space were also unsuccessful, leading again to poor exploration and conservative choices.

Several tuning strategies were applied in an effort to solve these issues:

In the machine learning side: experimenting with different weight initializations, activation functions, architectures, numbers of layers and neurons, etc.
In the RL framework: applying action masking, injecting Gaussian noise for continuous exploration, varying the number of users and serving stations, and more. Despite these efforts, none of the strategies produced the desired results.

As a result, attention shifted to alternative methods for solving the RAT selection problem. The DQN approach, which is presented as the main contribution of this work, was found to be a much more effective solution in this context.The integration of the Nash Equilibrium remains an interesting line of future research that could potentially provide a better solution for the RAT selection problem.

Chapter 7

Conclusion

This work has presented a way to effectively solve the RAT selection problem with a realistic approach, where users are constantly moving around the area, and making decisions provided only with self-obtained information. The DQN has proven to be an optimal method for solving this problem, specially by training the network with a centralized approach, using a global loss function, and then evaluating it in a user-centric and decentralized manner.

A realistic scenario where this solution could be deployed includes large-scale, high-density environments such as university campuses, airports, convention centers, shopping malls, etc. In these settings, users often move throughout a heterogeneous network covered by both LTE/5G and WiFi. These environments present the exact challenges addressed in this work: varying signal quality, uneven user distribution, limited user knowledge of the global network state, and the risk of congestion or disconnection. Additionally, in wide-area network (WAN) scenarios, where users have only partial visibility of the communication path, proper RAT selection becomes essential. In both cases, the proposed user-centric DQN-based framework allows devices to autonomously learn and adapt their RAT selection strategies to maximize throughput and ensure fair resource distribution, all without requiring any coordination. This makes the proposed approach both scalable and practical for real-world deployment.

Nevertheless, this work may have some potential limitations that have not been verified yet. Due to limited computational resources, the implementation of the model was done in a network with 20 users, which, although it is enough to model situations where multiple users share and compete for the network resources, it does not reflect the behavior of user-dense networks with hundreds or thousands of users.

Additionally, in this approach, the user is always moving across the same area that has the same number of LTE stations and APs. This allows the user to learn how to behave in a Heterogeneous network, where although stations and users may change position, the number of stations and users never changes. A more realistic approach could involve studying this solution under scenarios with a variable number of stations and users.

Finally, alternative mathematical approaches could be explored, such as incorporating game-theoretic concepts like the Nash Equilibrium into this framework. This would aim to reach a policy where no user can unilaterally improve their performance without affecting others. Whether this could lead to improved performance or higher fairness remains an open question for future investigation.

In summary, the results obtained in this work demonstrate the advantages of using a DQN-based approach for RAT selection in realistic, user-centric scenarios. By effectively balancing throughput, fairness, and system stability, the proposed method exhibits strong potential for practical deployment in real-world networks. The solution lays the groundwork for more intelligent, user-centric network management strategies as heterogeneous wireless networks continue to evolve.

Bibliography

[1] C. Joe-Wong, S. Ha, S. Sen, and M. Chiang. «HetNets Selection by Clients: Convergence, Efficiency, and Practicality». In: *Proceedings of IEEE INFOCOM.* 2013, pp. 862–870 (cit. on pp. 5, 22, 27).

[2] Y. Zhang, L. Zhang, J. Liu, Y. Shu, and H. Wu. «Intelligent User-Centric Network Selection: A Model-Driven Reinforcement Learning Framework». In: *IEEE Transactions on Vehicular Technology* 68.5 (2019), pp. 5031–5043 (cit. on pp. 5, 18, 22).

[3] Z. Yang, M. Sheng, X. Wang, J. Li, and Y. Zhang. «Reinforcement Learning with Network-Assisted Feedback for Heterogeneous RAT Selection». In: *IEEE Transactions on Wireless Communications* 19.9 (2020), pp. 5828–5843 (cit. on pp. 6, 22, 28).

[4] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* 2nd. Accessed: 2025-06-11. Cambridge, MA: MIT Press, 2018. URL: `http://incompleteideas.net/book/the-book-2nd.html` (cit. on p. 9).

[5] P. Casgrain and P. L. Bartlett. «Deep Q-Learning for Nash Equilibria: Nash-DQN». In: *Advances in Neural Information Processing Systems (NeurIPS).* 2021 (cit. on p. 37).

[6] C. Comaniciu, H. V. Poor, and A. Garcia. «User-Centric Radio Access Technology Selection: A Survey of Game Theory Models and Multi-Agent Learning Algorithms». In: *IEEE Communications Surveys & Tutorials* 21.3 (2019), pp. 3003–3031.

[7] H. Su, X. Wang, and M. Wu. «Beamforming Transmission in IEEE 802.11ac under Time-Varying Channels». In: *IEEE Transactions on Wireless Communications* 15.9 (2016), pp. 6210–6222.

[8] 3GPP. *Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Layer Procedures (3GPP TS 36.213 version 15.7.0 Release 15).* Tech. rep. Accessed: 2025-06-11. 3rd Generation Partnership Project (3GPP), 2019. URL: `https://www.etsi.org/deliver/etsi_ts/136200_136299/136213/15.07.00_60/ts_136213v150700p.pdf`.

[9] T. Bai and R. W. Heath Jr. «An Accurate Approximation of Resource Request Distributions in Millimeter Wave 3GPP New Radio Systems». In: *IEEE Wireless Communications Letters* 9.4 (2020), pp. 489–493.

[10] GeeksforGeeks. *Deep Q-Learning*. Accessed: 2025-06-11. 2023. URL: https://www.geeksforgeeks.org/deep-q-learning/.

[11] Arjun Sarkar. *Reinforcement Learning for Beginners: Introduction, Concepts, Algorithms, and Applications*. Accessed: 2025-06-11. 2022. URL: https://arjun-sarkar786.medium.com/reinforcement-learning-for-beginners-introduction-concepts-algorithms-and-applications-3f805cbd7f92.

[12] vwxyzjn. *CleanRL DQN Implementation (GitHub)*. Accessed: 2025-06-11. 2023. URL: https://github.com/vwxyzjn/cleanrl/blob/master/cleanrl/dqn.py.

[13] P. Casgrain. *Nash-DQN GitHub Repository*. Accessed: 2025-06-11. 2021. URL: https://github.com/p-casgrain/Nash-DQN/tree/master/Nash%20DQN%20-%20Updated.