Novel Deep Reinforcement Learning for User Association in Fog Radio Access Networks

Ignas Laurinavicius, Huiling Zhu, *Senior Member, IEEE*, Yijin Pan, Changrun Chen, Jiangzhou Wang, *Fellow, IEEE*

***Abstract*—As an evolution of cloud radio access network (C-RAN), fog radio access network (F-RAN) becomes promising for future mobile communications by enabling processing and caching at fog access points (FAPs). Different from the centralised C-RAN, F-RAN has a semi-distributed architecture, aiming to alleviate traffic load on the fronthaul links in C-RAN. Under the semi-distributed architecture in F-RAN, which employs a cell-free multiple input multiple output (MIMO) access technique, decisions on the joint user-FAP association and transmit power allocation are made at individual FAPs. To mitigate strong interference, FAPs will need to exchange cooperative status information, such as CSI, user association details or transmission power levels. However, this can lead to significant communication overhead within the network and introduce high complexity in the decision-making process. In this paper, accounting for the semi-distributed nature of the F-RAN architecture, reinforcement learning is leveraged as a potential solution to this kind of problem, and a novel multi-agent dual deep Q-network (MA-DDQN) algorithm is proposed by introducing experience exchange in partially observable Markov decision process environments. The simulation results show that the proposed reinforcement learning based algorithm outperforms the DDQN algorithm as well as the existing low-complexity algorithms.**

***Index Terms*—Reinforcement Learning, Deep Reinforcement Learning, Multi-Agent Deep Reinforcement Learning, Fog Radio Access Networks, Cell-Free MIMO, User-AP Association**

I. Introduction

As mixed reality technologies, such as virtual or augmented reality become more refined, stringent requirements are being placed on the infrastructure of the fifth generation (5G) mobile networks to support them. 5G's share of mobile data traffic is forecast to grow to around 75 percent in 2029 [1]. As 5G data continues to grow, the traditional centralised deployment model of the cloud radio access network (C-RAN), which uses a baseband unit (BBU) pool at the operator side connected to distributed radio radio heads (RRHs) at the edge via fronthaul links, faces challenges in handling the control plane signaling required for optimising radio performance. To tackle this issue, fog radio access network (F-RAN) has been proposed as a potential solution. F-RAN evolves from the traditional C-RAN architecture to address fronthaul congestion and latency issues. It replaces RRHs with fog access points (FAPs) that integrate both caching and application processing capabilities. By bringing caching and cooperative radio resource management to the network edge, F-RAN reduces the need for most data applications to traverse the entire network, thus alleviating pressure on fronthaul links and reserving capacity for latency-sensitive or processing-heavy tasks. This evolution places time-critical computing functions, such as cooperative radio resource management (CRRM) and cell-free MIMO signal processing, at the edge of the network. As a result, F-RAN can significantly extend the capabilities of conventional networks, enabling more efficient resource management and enhancing overall performance [2]–[6].

Furthermore, although the cloud layer in an F-RAN remains responsible for handling resource-intensive tasks and large volumes of data [7], the addition of processing and caching capabilities at FAPs at the network edge significantly changes data flow. By enabling these functions locally, F-RAN reduces the amount of channel information transmitted over long fronthaul links to the cloud for network control tasks, such as resource management. This also allows content acquisition and network control to be coordinated more efficiently at the fog network access layer. As a result, the F-RAN architecture can improve channel accuracy, leading to higher transmission rates and lower latency [6].

Although F-RAN demonstrates promising performance benefits, it also faces unique challenges that must be addressed before it can be fully operational [8]. With radio resource management and content acquisition capabilities enabled at the edge, cooperative methods need to be developed among FAPs for the distributed system. Given that FAPs can be densely deployed to serve a large number of user equipments (UEs), lack of cooperation among FAPs can lead to significant interference in the coverage areas of different FAPs. Therefore, joint UE-FAP association and transmit power management are essential for managing available resources efficiently. The issue of UE-FAP association in F-RANs has been explored in [9] and [10], which considered either the signal to interference plus noise ratio (SINR) [9] or received signal power [10]. However, these studies ignored the fact that each FAP only has limited resources to meet UE demands for services and content. When a single FAP is connected to many UEs, it risks becoming overloaded, which can significantly degrade performance within the fog network. Therefore, implementing load-balancing algorithms is crucial to ensure a smooth user experience [11]. Additionally, most research that focuses on UE-FAP association, including data caching [12], [13], has not considered that a significant portion of Internet content is generated from delay-sensitive applications, such as inter-

I. Laurinavicius is with Ericsson, Sweden, (e-mail: ignas.laurinavicius @ericsson.com), H. Zhu and C. Chen are with the School of Engineering, University of Kent, Canterbury CT2 7NT, U.K. (e-mail: h.zhu,j.z.wang,c.chen@kent.ac.uk), Y. Pan and J. Wang are with Southeast University, Nanjing, China (e-mail: panyj@seu.edu.cn, j.z.wang@kent.ac.uk)
This work was supported in part by the European Commissionâs Horizon Europe Marie SkÅodowska-Curie Postdoctoral Fellowship-UKRI guarantee under Grant EP/Y027558/1 (Corresponding author: Changrun Chen), and by National Natural Science Foundation of China under Grant No. 62001107.

active content and banking services [14]. When UEs that are connected to fog networks need to access this kind of delay-sensitive data, addressing the latency becomes essential in the UE-FAP association.

Considering factors such as the fronthaul load, cache prioritisation, user equipment (UE) performance requirements, network resource constraints, and the impact of edge computing, the association between UEs and FAPs presents a high-dimensional challenge under distributed architecture, which involves co-design of communication, computing and control towards optimal system performance. Existing research solutions do not adequately address this complexity. When the number of UEs is large, achieving optimal performance through traditional algorithms becomes nearly impossible due to the high computational complexity involved. Recent research has studied deep learning-based solutions within the distributed architecture by allowing distributed nodes, e.g. access points, to act as agents and exploring information/experience sharing among agents for decision-making [15]–[17]. In [15], within a distributed mobile edge computing (MEC) environment, MEC nodes, acting as agents, communicate with each other to increase the number of observations available to each agent, which improves the decision-making for optimal association between MEC nodes and devices. In distributed networks, experience sharing among agents has been demonstrated as a powerful tool for enhancing performance [16], [17]. For instance, [16] explored experience sharing by allowing agents to simultaneously perform policy optimisation and evaluation through actor-critic mechanisms. Multiple agents interact independently with separate instances of the same environment, sharing their experiences without directly influencing each other's decisions. This setup enables efficient learning while maintaining simplicity, similar to single-agent learning under the same environment for all agents. Furthermore, recent research considered the environment dynamics for multi-agent reinforcement learning (MARL). Agents actively exchange information about the environment, which allows them to make joint observations and coordinated actions. This form of experience sharing can significantly enhance network performance, especially in distributed systems where multiple agents need to work together effectively. In [17], how agents can learn cooperatively was explored, leading to faster learning and improved decision-making. By sharing observations and coordinating actions, agents in distributed networks can overcome the limitations of isolated learning, leading to more efficient resource management and overall improved performance.

However, these results are only applicable to fully observable or complete Markov Decision Process (MDP) environments, where agents operate independently within separate instances of the same environment. In the aforementioned studies, when agents share the same environment, extensive communication is needed before they can make a decision [18]. In the context of the F-RAN architecture, meeting latency requirements is critical. In practice, FAPs acting as agents can only partially observe the environment, resulting in a partially observable MDP (POMDP). Attempting to exchange observations to form a joint understanding and coordinated action would introduce significant communication overheads.

A potential solution to this problem in MARL would be to train a double deep Q-network (DDQN) to learn about the environment. DDQN is a reinforcement learning (RL) method that employs two networks, an estimation network and a target network. This approach, proposed in [19], was originally designed to address the limitation of classical deep-Q-network learning, which only uses one estimation network to learn. By introducing a target network that is updated gradually, the stability of the learning process improves. The target network is assumed to be a more accurate representation of the ground truth, providing the estimation network with a benchmark for comparison. This setup allows for more stable decision-making. Original DDQN performs well in single-agent environments, where it can effectively estimate MDP state transitions and rewards. However, its performance declines in multi-agent, partially observable MDPs (POMDPs) [20]. In an F-RAN, multiple FAPs create a shared POMDP environment. In this environment where each agent seeks to maximise its own reward, effective cooperation becomes challenging. The estimation network struggles to accurately predict and reflect the actions of other agents, leading to higher uncertainty and reduced stability. This issue remains unresolved even with gradual updates to the target network and extended training times, making it difficult for the agents to learn effectively in these complex environments.

This paper aims to tackle the challenge of the UE-FAP association in the F-RAN architecture by modeling it as a partially observable distributed MARL problem. In the model, each FAP operates as an independent agent, while also collaborating with other FAPs to maximise the number of served UEs under certain constraints. Due to the dynamic nature of the environment, agents must learn that trade-offs are necessary to accommodate the maximum number of UEs and adapt the solution to satisfy UE requirements. The main contributions of this paper are summarised as follows:

1) We analyse the distributed F-RAN architecture and find that due to the low latency requirement in future wireless communications, agents may not have enough time to exchange complete state information. Since a fully observable MDP cannot be developed due to strict constraints, we formulate a POMDP environment. Acting as deep learning agents, FAPs must learn not only how their actions affect the system state, but also how to predict the actions of other agents in the same state. Our analysis shows that traditional low-complexity algorithms perform poorly in the F-RAN environment, as they prioritise greedy UE associations that benefit individual FAPs rather than optimising the entire system's performance.
2) We propose a solution for a POMDP environment called multi-agent distributed double deep Q-network (MA-DDQN), which adopts an experience exchange method to share information, such as the states, actions, rewards, and subsequent states between agents in the environment. This approach is particularly helpful in making deep learning agents diverge from a local optimum solution, as agents can build a larger knowledge base

required for learning than they would independently. Under the system constraints, we demonstrate how agents can effectively use the experience exchange mechanism without modifying the neural network hyperparameters, ultimately achieving better rewards compared to low-complexity, hard-coded methods. The simulation results show that the proposed algorithm outperforms the conventional DDQN algorithm which uses no experience exchange.

The remaining structure of the paper is as follows. In Section II, the system model is introduced as an iteration of C-RAN, with distributed coordinated multi-point radio resource allocation. Section II also formulated the main optimisation problem, including the constraints. In Section III, based on an overview of the RL concepts, we propose a novel deep RL algorithm to solve partially observable multi-agent problems and design an environment which simulates network dynamics. In Section IV, the results of training and simulation are presented and discussed, and the paper is concluded in Section V.

II. System Model and problem formulation

A. Transmission model

Consider an F-RAN architecture including a set of FAPs $\mathcal{J}$ with a total of $|\mathcal{J}| = J$ FAPs, each of which is equipped with M antennas, as shown in Fig. 1. FAPs are connected to the cloud network through the fronthaul links. They are also connected to each other via high-speed fibre links in a mesh topology [21]. In F-RAN, network access layer requires horizontal connectivity to allow FAPs to exchange information required for cell-free CRRM [22], as such, links between FAPs are utilised to share network information with low latency. FAPs can connect to multiple UEs by using cell-free MIMO methods for joint transmission and coordinated beamforming [23]. The set of UEs to be served is denoted as $\mathcal{K}$, with a size of $|\mathcal{K}| = K$. It is assumed that each UE has a single antenna. Each FAP can be associated with a number of UEs through multiple beams, and the association factor between UE $k \in \mathcal{K}$ and FAP $j \in \mathcal{J}$ is denoted as a variable $x_{j,k}$, given by

$$x_{j,k} = \begin{cases} 1, & \text{if } k \text{ is associated to FAP } j, \\ 0, & \text{if not.} \end{cases} \tag{1}$$

The sum of associations $\sum_k x_{j,k}, \forall j \in \mathcal{J}$ is the cardinality of set $\mathcal{K}^{(j)}$ that includes all the UEs associated with FAP j. The union of all association sets produces a set of UEs associated with the fog $\mathcal{K}_\Phi$ and is shown as $\bigcup_{j=1}^{J} \mathcal{K}^{(j)} = \mathcal{K}_\Phi$.

Note that even if the interest lies in associations within fog with global synchronisation between FAPs assumed, some information may not be known by the rest of the FAPs. In a downlink communication scenario, UEs estimate the channel condition and feed it back. Assuming that the channel condition is perfectly known, UE k requests content $c_k = [0, 1, \cdots, \phi, \cdots, \phi_{\max}]$ of size b_ϕ. If the content request is fulfilled, UE k is considered as served. When the UE service request is done, it is included in a set $\mathcal{K}_d$, and the number of served UEs will be given as $|\mathcal{K}_d| = x_d$. Employing cell-free MIMO [22] [24], the file would be jointly transmitted to the UE in multiple streams of simultaneous message signals s_k. With coordination, interference alignment between APs is possible, and UE k can be associated with multiple FAPs, while UEs that are not associated with all FAPs can also receive a signal with reduced interference from other FAPs [25]. Since the UEs are static, the main source of interference is the inter-stream interference between FAPs. As such the received symbol at UE k is given by

$$y_k = \sum_j x_{j,k} p_{j,k} \boldsymbol{h}_{j,k}^{\mathrm{H}} \boldsymbol{w}_{j,k} s_k + \sum_j \sum_{i \neq k} x_{j,i} p_{j,k} \boldsymbol{h}_{j,k}^{\mathrm{H}} \boldsymbol{w}_{j,i} s_i + n, \tag{2}$$

where n is the additive white gaussian noise (AWGN), satisfying $\mathcal{N}(0, \sigma^2)$. $p_{j,k}$ is the waterfilling power allocation from FAP j to user k. $\boldsymbol{h}_{j,k}$ is the $\mathbb{C}^{M \times 1}$ static complex Gaussian channel model composed of M channel responses, denoted as

$$\boldsymbol{h}_{j,k} = \begin{bmatrix} \mathrm{PL}_{j,k}^{\mathrm{ABG}} h_{j,k,1} & \mathrm{PL}_{j,k}^{\mathrm{ABG}} h_{j,k,2} & \cdots & \mathrm{PL}_{j,k}^{\mathrm{ABG}} h_{j,k,M} \end{bmatrix}^{\mathrm{T}} \tag{3}$$

where each entry $h_{j,k,m}$ represents the channel of UE k to antenna m of FAP j. Defining $\mathrm{PL}_k^{\mathrm{ABG}}$ as a pathloss coefficient that follows the ABG urban micro open space pathloss model [26] for high frequency communications, the pathloss coefficient for frequency f, distance $d_{j,k}$ from UE k to FAP j, with log-normal shadowing $\chi_\sigma^{\mathrm{ABG}}$ is evaluated as follows

$$\begin{aligned} \mathrm{PL}_{j,k}^{\mathrm{ABG}}(d_{j,k}) = & 10 \cdot 2.6 \cdot \log_{10}\left(\frac{d_{j,k}}{1\mathrm{m}}\right) + 24 + \\ & 10 \cdot 1.6 \cdot \log_{10}\left(\frac{f}{1\mathrm{GHz}}\right) + \chi_\sigma^{\mathrm{ABG}}. \end{aligned} \tag{4}$$

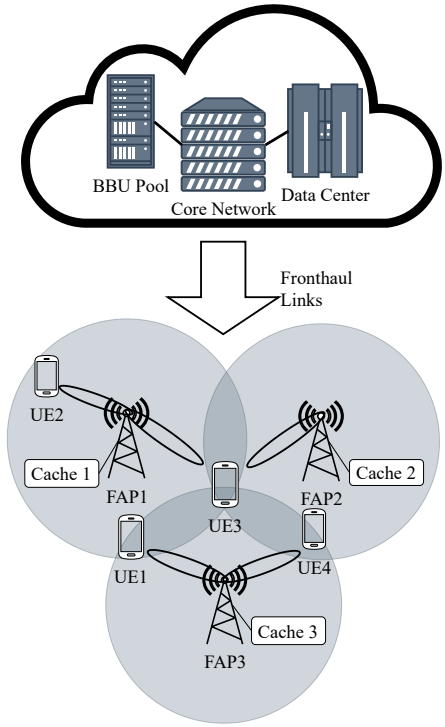


Fig. 1: System model showing a F-RAN deployment as an extension to the cloud. Backhaul links between FAPs (short dashed lines) allow for CRRM.

In (2), $\boldsymbol{w}_{j,k}$ is the $\mathbb{C}^{M\times 1}$ transmitter beamforming weights vector. For simple analysis, zero-forcing is considered for the transmit beamforming vectors [27], given by

$$\boldsymbol{w}_{j,k} = (\boldsymbol{h}_{j,k}^{\mathrm{H}}\boldsymbol{h}_{j,k})^{-1}\boldsymbol{h}_{j,k} \tag{5}$$

FAPs have limited instantaneous transmit power, denoted as P_j^{TRI}, given by

$$P_j^{\mathrm{TRI}} = \sum_k P_{j,k}^{\mathrm{TRI}} = \sum_k x_{j,k}p_{j,k}\left|\frac{\boldsymbol{w}_{j,k}}{|\boldsymbol{w}_{j,k}|}\right|^2, \tag{6}$$

where the precoding vector $\boldsymbol{w}_{j,k}$ is normalised by its magnitude to ensure that only interference cancellation is being performed by the precoding vectors.

The resulting signal to interference plus noise ratio (SINR) experienced by UE k is given by

$$\gamma_k = \frac{\sum_j x_{j,k}p_{j,k}|\boldsymbol{h}_{j,k}^{\mathrm{H}}\boldsymbol{w}_{j,k}|^2}{\sum_j\sum_{i\neq k} x_{j,i}p_{j,i}|\boldsymbol{h}_{j,k}^{\mathrm{H}}\boldsymbol{w}_{j,i}|^2 + \sigma^2}. \tag{7}$$

The corresponding data rate of UE k is denoted R_k, given by

$$R_k = \log_2(1+\gamma_k), \tag{8}$$

B. Cache and Scheduling Model

Consider that the F-RAN has caching capabilities at FAPs for an improved UE experience. FAPs will be located within a small area to support the transmission of delay-sensitive contents. Considering FAPs only have limited caching capability, the caches will be populated by a uniform random distribution. Content availability in FAP j is denoted as a binary value $c_{j,\phi} \in \{0,1\}$, and $\sum_\phi c_{j,\phi} \leq C_j$, where C_j represents the cache capacity. To guarantee the latency requirement, δ_{th}, for the contents, it needs to analyse the user experienced delay, depending on whether content is present in the caches. As a UE demands content ϕ of size b_ϕ, each FAP is aware if it is present in the cache, as such, the delay for UE k is split into three cases

$$\delta_k = \begin{cases} b_\phi\left(\frac{1}{\Psi_j}+\frac{1}{R_k}\right), & \text{if } c_{j,\phi}=1 \\ b_\phi\left(\frac{1}{\Psi_j}+\alpha\frac{1}{R_{\text{bh}}}+\frac{1}{R_k}\right), & \text{if } c_{l,\phi}=1 \text{ , } l\neq j \\ b_\phi\left(\frac{1}{\Psi_b}+\frac{1}{R_{\text{fh}}}+\alpha\frac{1}{R_{\text{bh}}}+\frac{1}{R_k}\right), & \text{otherwise.} \end{cases} \tag{9}$$

In the first case in (9), Ψ_j denotes the processing speed of FAPs. Ψ_b is the processing speed of the BBU pool and it is going to be assumed that $\Psi_b > \Psi_j$. In the second case, the content is not available in the immediate FAP, but can be acquired from adjacent FAP l. R_{bh} is the backhaul link rate, and α denotes the minimum number of hops required to acquire the content. In the final case, the content is unavailable in the adjacent FAPs, as such the content has to be acquired from the cloud through the fronthaul. Fronthaul rate is defined as R_{fh} and the typical relationship among the transmission, backhaul, and fronthaul rates is given by

$$R_k \leq R_{\text{bh}} \leq R_{\text{fh}} \tag{10}$$

where R_{fh} and R_{bh} are assumed to be constant, and if in use, they can be close to their capacities. Since δ_k is dependent on R_k and, as a result on how many FAPs are associated with user k, some FAPs may need to wait before content is acquired before transmitting the content. As such, if the FAPs associated with the UEs have different contents in caches, the highest delay value is used.

Corresponding to three cases, FAPs that are associated with UE k will consume a different amount of power to acquire the content through the network. Fronthaul and backhaul consume power at a time instant, modeled as

$$P_{j,k}^{\mathrm{XH}} = \begin{cases} 0, & \text{if } c_{j,\phi}=1 \\ \alpha\left(P_{\text{ONU}}+P_{\text{LS}}\right)+N_{\text{LC}}P_{\text{LC}}, & \text{if } c_{l,\phi}=1 \text{ , } l\neq j \\ \alpha\left(P_{\text{ONU}}+P_{\text{LS}}\right) & \\ +N_{\text{LC}}P_{\text{LC}}+mP_{\text{SW}}+P_{\text{SWB}}, & \text{otherwise.} \end{cases} \tag{11}$$

In the first case in (11), content does not need to traverse the network and is present in the cache, meaning no x-haul power is being consumed. A linecard (LC) is a piece of x-haul hardware used for a multitude of functions, such as digital-to-analog conversion within switches and local communication between FAPs and the cloud. In the second case, N_{LC} and P_{LC} represent the number of LCs used in the network and the power consumed by each LC, respectively. LCs can be shared in the scenario of sharing the same wavelength, as such, no more than one LC will be used in this model. In the same case, P_{ONU} and P_{LS} are the power consumed by optical network units, used for high-speed data transmission, and each switch along the path in the backhaul, respectively [28]. In the final case, P_{SW} denotes consumed power at each switch port in the cloud and m is the number of switch ports used in the content's path. P_{SWB} is the baseline power consumption of the cloud switch.

C. Problem Formulation

Through the UE-FAP association, which generates a set $\mathcal{K}_\Phi$, the performance objective is to maximise the number of UEs x_d, which are served by the fog, with their latency requirement, δ_{th} being satisfied. Since the power consumed is different for all UEs, due to varying delays, energy consumption will provide some trade-off if the UEs' requested content is unavailable, but they are experiencing high SINR conditions. The total power consumption can be derived as a sum of the x-haul power consumption and FAP beamforming power consumption to serve all selected UEs and is expressed as

$$P_{\text{tot}} = \sum_j\left(P_j^{\mathrm{TRI}} + \sum_k P_{j,k}^{\mathrm{XH}}\right). \tag{12}$$

Finally, the problem formulation of user-FAP association is

given by

$$\text{O1:} \quad \arg\max_{\mathcal{K}_\Phi} x_d, \tag{13a}$$

$$\text{s.t.} \quad P_j^{\text{TRI}} \leq P_T, \forall j \tag{13b}$$

$$R_k \geq R_{min}, \forall k \tag{13c}$$

$$\delta_k \leq \delta_{\text{th}}, \forall k \tag{13d}$$

$$P_{\text{tot}} \leq P_{\text{th}} \tag{13e}$$

where the first constraint (13b) limits the transmit power of FAPs, which also limits how many UEs can be served by individual access points. Constraint (13c) guarantees that the UE sets selected by the system have satisfactory rate performance for serving groups of FAPs. Since fronthaul is used for accessing data that is not present within the fog, the latency constraint δ_{th} shown in (13d) can help locate UEs with low transmission rate to benefit from files located within the caches in the fog, and place high rate UEs to the fronthaul, in which case the delay introduced by the fronthaul links is endurable. δ_{th} can be chosen differently for different types of contents. Fronthaul links also consume power, which introduces P_{th} as the power constraint in (13e). Fronthaul consumes much more power than radio transmission, if unconstrained, selected UE sets can consume tremendous amounts of power [29]. Reducing the amount of power consumed by the UEs means sustaining as many communications within the fog as possible. This benefits the overall goal of improving energy efficiency in the upcoming generations of mobile communications [30]. Further, looking at the delay characteristics in (9), if all possible requested UE content is present in caches, the selection becomes rate dependent, rather than relying on reducing the number of links being used for fronthaul transmissions. As such, a fixed solution is not beneficial and must be adaptable to many different factors in the network.

Due to the presence of an integer optimisation variable and multiple floating-point constraints, this problem becomes a non-convex mixed-integer non-linear programming problem. Consider the following factors: 1) there are K UEs, 2) each UE can associate with more than one FAP, 3) a total of M UEs can be admitted at one FAP with M antennas, and 4) the sets of selected UEs can be different at each FAP. Therefore, the computational complexity for an optimal brute-force solution at each FAP j is $O(JK^M)$. With this combinatorial explosion, finding an optimal solution is difficult, and using brute-force search methods is computationally infeasible even at relatively small networks ($M \approx 8$, $K \approx 16$). To find feasible solutions to the problem, we will adopt a deep RL-based method, which has high potential in dealing with dimensionally complex problems.

III. Distributed Double Deep Q-Neural Network

This section presents how RL can be evolved and applied to the F-RAN. When solving problems using RL in communication problems, the solution is generally based on a fully observable MDP. While MARL can be easily applied to networks with full knowledge of the system state, it requires a high amount of communication between agents. Due to a low latency requirement raised for F-RAN, applying RL to solve the F-RAN optimisation problem in (13) using a fully observable MDP is infeasible. Therefore, we formulate the environment as a POMDP. Then, a modification to the state-of-the-art deep RL algorithm is proposed to solve POMDP problems based on designing the environment which simulates the network dynamics.

A. Deep Reinforcement Learning

Deep RL is termed "trial and error learning". FAPs act as agents. Therefore, there are J agents in the network. An agent, which is an intelligent entity represented by a neural network, interacts with an environment and learns how to map rewards from actions taken in states [31]. The UE-FAP association in F-RAN can be represented as an MDP with state and action spaces, denoted $\mathcal{S}$ and $\mathcal{A}$, respectively. The smallest time unit experienced in the network is called a timestep t, and T timesteps are defined as an epoch. A snapshot at t is known as an experience $E_j(t) = \{o_j(t), a_j(t), r_j(t), o_j(t+1)\}$, in which an agent j observes a part of the state $o_j(t) \in s(t)$, where $\mathbf{s}(t)$ is the full state of the environment at timestep t. Observations $o_j(t)$ can be continuous and infinite [32], which leads to a requirement of a large number of experiences to learn about the environment behaviour. Since each agent has a limited observation $o_j(t)$ of the system state, $\mathbf{s}(t)$ is composed of multiple observations experienced by all agents individually at timestep t i.e. $\mathbf{s}(t) = [o_1(t) \quad o_2(t) \quad \cdots \quad o_j(t)]$. Using this information, the agent chooses an appropriate action $a_j \in \mathcal{A}$, resulting in a new state $\mathbf{s}(t+1) \in \mathcal{S}$ and observation for each agent $o_j(t+1)$. Each agent takes actions on their respective partial observations, and the action vector is given by $\mathbf{a}(t) = [a_1(t) \quad a_2(t) \quad \cdots \quad a_j(t)]$. Unlike in a deterministic environment, it cannot be guaranteed that an agent taking an action will lead to the expected state. An example in our environment is to decide to associate a FAP with one UE. Because of lack of inter-agent communication prior to deciding, the UE may be overlooked by other agents. Furthermore, channel conditions can be random, and not dependent on the agents' previous actions. When an action is chosen at every agent, the agents perceive state transitions from state $\boldsymbol{s}(t)$ to $\boldsymbol{s}(t+1)$ with a time-varying probability, which is denoted as $\Pi(\boldsymbol{s}(t+1), r(t) | \boldsymbol{s}(t), \boldsymbol{a}(t))$. With the state transitions, a reward $r_j(t)$ is distributed equally among the agents. The reward determines how good or bad the agents' decision is and is closely related to the optimisation objective. The purpose of an RL agent is to maximise the long-term rewards. For example, choosing to associate FAPs with UEs that historically have not given good rewards may result in suboptimal performance and it would be expected that the agents will be less likely to associate with those UEs. The transition probabilities are dependent on the state that all the agents are in. However, it still applies that the sum of these probabilities is as follows

$$\sum_{\boldsymbol{s}(t+1) \in \mathcal{S}} \Pi(\boldsymbol{s}(t+1), r(t) | \boldsymbol{s}(t), \boldsymbol{a}(t)) = 1, \quad \forall \boldsymbol{s} \in \mathcal{S}, a \in \mathcal{A}. \tag{14}$$

Furthermore, the connection between t and $t+1$ is temporal and the timestep progresses once an action has been made

by all agents. Since the goal of each agent is to maximise the combined expected reward $\max \mathbb{E}[r(t)|s_j(t)]$, the agents have to work collaboratively to learn a method to achieve the optimal outcome. The strategy is an approximated function or a neural network map β of taking any action a_j under a given state o_j. Since we want the agent to learn about the environment to maximise the possible reward, we are interested in the value of taking an action while in a given state. This is denoted as a Q-value and is derived as

$$Q_\pi\left(s_j(t), a_j(t)\right) = \sum_{s_j(t+1)} \Pi\left(s_j(t+1), r(t) | s_j(t), a_j(t)\right) \cdot \cdot \left[r(t) + \rho \max_{a_j(t+1)} Q_\pi(s_j(t+1), a_j(t+1))\right], \tag{15}$$

where $\rho \in [0, 1)$ is introduced as the discount rate, choosing values closer to 1 makes the agent more exploratory, whereas values closer to 0 encourage the agent to be more greedy with the actions.

Given sufficient experience, an RL agent following (15) can find the optimal solution to any targeted problem. Explicitly following the optimality equations is not feasible, as the complexity is similar to exhaustive search. Looking ahead to find the best options for each state results in a number of assumptions as well, which are very unlikely to occur in practice. Deep neural networks (DNN), which is also referred to as deep Q-networks (DQNs) [32] can be used to approximate working solutions. This approximation of Q-values is denoted as $Q(s_j(t), a_j(t); \beta)$, where β represents the trained parameters of a deep network. It is commonly known that DQNs can diverge in performance over a long period of time, due to the correlation of samples [19]. Double Deep Q-networks [19] and experience replay [33] have been used to reduce the correlation between samples, and more importantly remind the networks of rare states that they can appear in, preventing divergence of the possible solution and overestimating the predicted reward. The trained parameters in the learning network are estimated by optimising a system of loss functions for each time step. A loss function for a DDQN [19] is given by

$$\mathcal{L}(\beta) = \mathbb{E}\Big[\Big(r(t) + \rho Q\left(s_j(t+1), \arg\max_a Q(s_j(t+1), a|\beta)|\beta'\right) - Q(s_j(t), a_j(t)|\beta)\Big)^2\Big], \tag{16}$$

where a new variable β' represents the target network parameters. As shown in Fig. 2, the structure of the target network is the same as the estimation network with parameters β, with the main difference being that the target network is updated every N steps with $\beta' = \beta$. In this way, the neural network will produce the action that provides the least predicted loss in the environment.

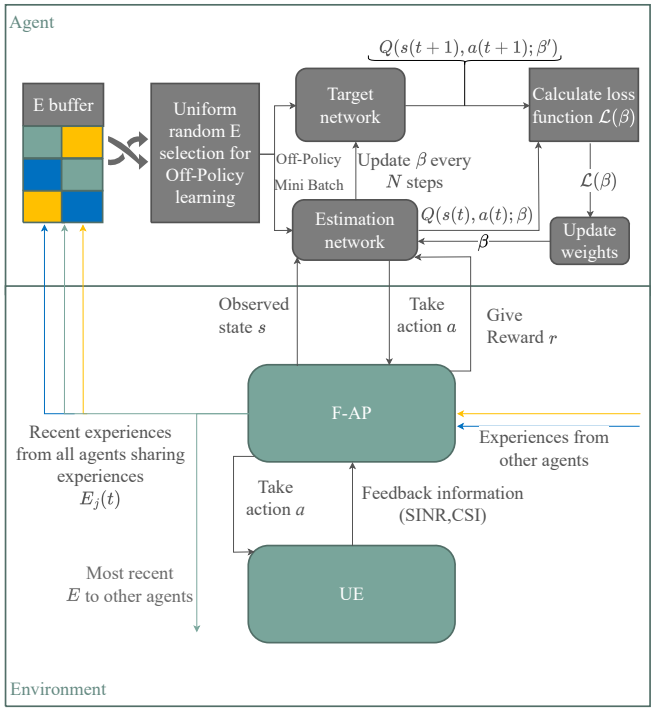


Fig. 2: A Markov chain showing interaction between a learning agent and environment, with multiple agents sharing experiences for replay .

B. Environment Design

Consider an environment following the model defined in Section II. In this section, the states, actions and rewards which are available to each agent are defined in more detail. It is assumed that all UEs enter and remain within the range of their respective FAPs until their content requirement $c_k = \{0, 1, \cdots, \phi_{max}\}$ is met. As stated in the system model, UEs provide feedback with CSI holding information about SINR that they are experiencing γ_k, based on the estimated channel strength $|\mathbf{H}_{j,k}|$. The FAP itself holds information about the power P_j, the availability of the content $C_{j,\phi}$, and the expected delay under current conditions $\delta_{j,k}$. In this scenario, an agent is located within each FAP, accessing the required observations. An agent navigates the state-space observing one UE at a time, and a single epoch lasts until all UE-FAP associations have been determined, which is defined as an endgame. The specific variables and values are elaborated as follows

- An endgame $D \in \{0, 1\}$ is said to be reached ($D = 1$) when all users in set $\mathcal{K}$ have been acted on, timestep $t = T$, initially $D = 0$.
- The state observation vector for a single agent j is $o_j(t) = \{\alpha, t, |\mathbf{H}_{j,k}(t)|, R_k(t), \max \delta\}$ and a collection of observations from all agents is the full state of the system $\mathbf{s}(t) = [o_1(t) \quad o_2(t) \quad \cdots \quad o_j(t)]$. This ensures that the algorithm is flexible for a changing number of UEs.
- An action $a_j(t) = \{0, 1\}$ is taken by all agents for each UE in range at the same time, where action indices correspond to "do not associate with current

user", "associate with current user", respectively. A collection of actions from all agents will be $\mathbf{a}(t) = [a_1(t) \quad a_2(t) \quad \cdots \quad a_j(t)]$.

- The reward $r(t)$ is handed out to the agent after each action and depends on the performance metrics resulting from the decision. The reward function is defined as

$$r(t) = \begin{cases} -\frac{P_{\text{tot}}}{P_{\text{th}}} - \sum_k \frac{\delta_k}{\delta_{\text{th}}}, & \text{if } D = 0, \\ x_d(t) - \frac{P_{\text{tot}}}{P_{\text{th}}} - \sum_k \frac{\delta_k}{\delta_{\text{th}}}, & \text{if } x_d > 0 \text{ and } D = 1, \\ r_{min}, & \text{if } x_d = 0 \text{ and } D = 1, \end{cases} \tag{17}$$

  where statistic of all UEs within a fog is readily available to the agents.

The way the reward is formulated improves the prospects of cooperation. For the first two conditions, performance metrics are weighted by power and delay thresholds, P_{th} and δ_{th}, respectively. This is used to normalise the impact of each performance indicator in the rewards. Power consumption can be higher by a factor of a thousand than the number of UEs associated with and significantly larger than the delay. The first condition grants the agent intermediary rewards based on power consumption and expected delays for the current number of selected UEs, and grants the agent with an ability to familiarise itself better with state transitions. The second condition assumes an end of association period and grants the rewards based on the end result of the agent's decisions. The first term is the term that satisfies the primary objective of a number of associations, and the second term describes the condition of energy consumption within the network. The last element of the equation is the sum of the delays that the UEs are experiencing at the time of reward. The third condition prevents the agent from not selecting any UEs, by using a minimum reward, r_{min}, as this undesirable solution theoretically grants null power consumption and does not introduce any delays to other UEs. Hence, a variable reward is chosen that would typically be lower than the rewards from the top two cases. While the first case is useful in helping the agent navigate the state-space, only the second condition is important and shows how good the performance is when the final decision is made. A high reward means that a large number of UEs have been served with power consumption and delay close to the threshold.

C. Constraint Handling

Agents only take actions based on the system observation o_j, and are not explicitly aware of the system constraints defined in (13b)-(13e). To support system constraints, algorithms implementing RL must have approaches to address the constraint handling. The constraint handling method is included in Algorithm 1. To make sure that constraints (13b)-(13d) are satisfied, the algorithm verifies them whenever an agent makes an action that would alter the selected user set. Initially, if the FAPs are informed of the association decision, they must send an association signal to UE k. Upon collection of association signals, the number of FAPs wanting to associate is fed back to each FAP. The outcome for a new user set is found by calculating the expected R_k, P_{tot}, and δ_k for all K users. If constraints corresponding to these values, $R_{\text{min}}, P_{\text{th}}, \delta_{\text{th}}$, respectively, would be breached, an action must be prevented and no association takes place. If these conditions are met, the FAP must check if the new user set will breach the association constraint, as the number of served users for a single FAP is constrained by the number of antennas M.

D. Neural Network Workflow

When more than one access point can communicate with the same UEs, coordinating their efforts, as opposed to competing to provide resources to the same UE, provides a significant performance boost [34]. Coordinated beamforming is typically performed at a centralised node, such as the BBU of the network, providing the highest boost in terms of sum-rate. The drawback of using centralised coordinated beamforming is the resulting long latency because of high dependency on fronthaul rates [35]. Hence performing beamforming, by aligning beamforming vectors at edge nodes, results in reduced latency, due to more accurate channel information achieved at the edge, which can equivalently achieve better performance than centralised beamforming. The issue imposed by enabling FAPs or edge-nodes cooperation is that overhead is required in exchanging UE and decision information before UEs can be served. This can be solved by predicting other FAPs' decisions using a deep learning method.

Most previous solutions focused on learning locally and then sending data to a global aggregator, e.g. federated learning [36], distributed learning [37]. Keeping data centralised has its cons like reduced performance due to capacity constraints and concerns, such as privacy and security. And, classical distributed learning suffers from having to share model or gradient data between agents, which can grow large with more advanced neural networks present in access points. Solutions can also consider agent placement inside UEs [38]. In such scenarios, the agents may not be aware of the placement of content or the network load, which would lead to degraded performance. A more significant issue is that agents in UEs might not be aligned with network goals, such as minimising latency or maximising throughput, which would impact their performance severely [39].

In our research, learning happens by agents exchanging raw experience information $E_j(t) = \{o_j(t)), a_j(t), r(t), o_j(t+1)\}$ with each other over backhaul links. This is most like collaborative evolutionary learning, as such information includes essential parameters allowing FAPs to assess the environment before making an action. Exchanging such data between FAPs instead of sending it all to the core network or an aggregator somewhere at the edge reduces loads on fronthaul links. Models are also allowed to develop independently to maximise their performance among themselves. In this method, agent cooperation is not only implicit, but also explicit, by introducing experience sharing as shown in Fig. 2. Experience replay buffers are used for single-agent off-policy learning to prevent a high temporal correlation from recent samples in learning and reduce variance in predicted rewards [32]. Sampling experiences in mini-batches using any probability

distribution improves the agents' ability to adapt the current policy to previously experienced scenarios. This method of learning is called off-policy learning since the data that is sampled is not representative of the current policy used by

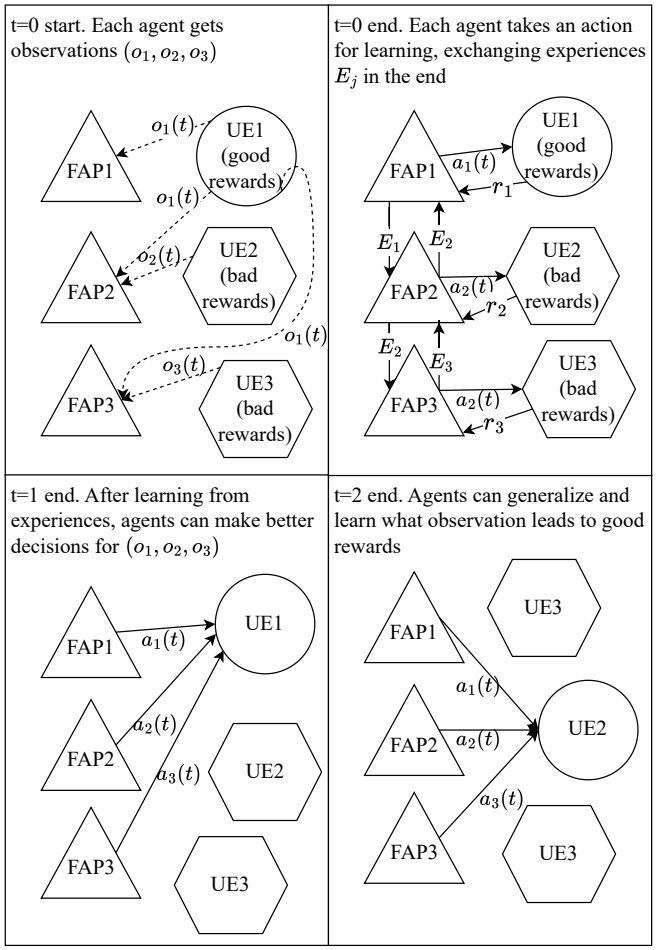


Fig. 3: Workflow concept diagram showing how experience sharing helps agents learn from other agents' experiences as if they were their own through off-policy learning.

Since, by definition, off-policy learning can be achieved by sampling decisions made by other policies, we propose to extend the algorithm by introducing experience exchange between the learners. When the environment states are not defined by finite integers or finite names, the state-space can become exponentially large or infinite. Even with a limited range, floating-point descriptor values can have infinitesimal, but meaningful contributions to finding optimal or better solutions. Additionally, with a large number of features, the curse of dimensionality becomes prominent [40]. It becomes difficult for agents to learn about the environment efficiently if they learn alone. Additionally, due to multiple agents working in the same environment and affecting the reward, the partially observable property makes the environment even more difficult to resolve [41], as the agents have to predict what actions other agents would make, to maximise the reward. Refer to Fig. 2 for an example flow diagram of a multi-agent deep Q-network RL algorithm. If the agents learn individually, the algorithm could provide good results eventually, with little convergence guarantee. Without having enough data to learn, agents are likely to perform actions that provide a local optimum solution or no solution for an unexplored state. Additionally, independent Q-learning struggles with learning in multi-agent environments [20]. Learning efficiency reduces with an increase of experience replay size, due to outdated policies being present in the experience replay buffer for a long period of time. On the other hand, a reduced buffer size reduces the stability of learning, as the samples become closely temporally correlated, which results in a high variance in rewards, similar to the problem in [19]. Having agents exchange experience information increases the amount of experiences with policies that are different from the current one and helps agents learn about states they could have previously not visited. An example of this is shown in the model for experience exchange in Fig. 3. At $t = 0$, agents only have clean experience buffers or experience buffers with their own previous experiences. The UEs contact all FAPs with information relevant to the FAP, the FAP adds on additional information about the state of the FAP and this constitutes the observation $o_j(t)$. For the observation, each agent j will make an action $a_j(t)$ at the same time. For each agent's action, a reward $r_j(t)$ will be estimated and granted to the agent together with the next user's state and this creates the experience $E_j(t) = \{o_j(t), a_j(t), r_j(t), o_j(t+1)\}$. At the end of the step, each agent exchanges experience information with every other agent. As time goes on, agents share raw experience data with each other. With the number of shared experiences increasing, agents' decisions that lead to solutions matching other agents with good rewards get reinforced. On the other hand, solutions that would lead to local optimum solutions, but do not benefit the overall goal as much, get pruned, as shown in the $t = 1$ section of the same figure. Furthermore, at $t = 2$, we show that due to the generalisation property of Q-learning, agents do not need to get consistently retrained on similar properties. If an observation matches a previously observed solution, agents intuitively find a good solution. In this case, not all agents have to test the experiences individually. The experience information gets sampled in mini-batch testing as shown in the Experience Handling plane in Fig. 2.

E. MA-DDQN Algorithm Design

Off-policy learning is a powerful tool that improves convergence speed and guarantees robustness in novel deep RL algorithms. Similar to (16), the loss function with experience replay can be extended as

$$\begin{aligned}\mathcal{L}(\beta) = \mathbb{E}_{\mathcal{U}\sim(\mathcal{E})}\Big[\Big(r(t)+ \\ \rho Q\left(s_j(t+1), \arg\max_a Q(s_j(t+1), a|\beta)|\beta'\right) \\ - Q(s_j(t), a_j(t)|\beta)\Big)^2\Big],\end{aligned} \tag{18}$$

where the addition to the function appears next to the expectation operator. $\mathcal{U} \sim (\mathcal{E})$ is defined as a uniform distribution over the experience database $\mathcal{E}$. The uniform distribution also includes the current state that the agent is experiencing.

This solution was fine for single-agent environments, but was not well explored in multi-agent environments. Usually, this was circumvented in literature by implementing more complex algorithms, allowing full tracking of historical states, like DRQN [42], or by performing centralised training and executing distributively [43], which led to large overheads in the network due to the sheer data requirement to download the new models. The proposed Multi-Agent Distributed DDQN Algorithm addresses these issues by decentralising experience sharing. It enables agents to exchange compact experience tuples, enhancing generalisation while maintaining computational and communication efficiency.

Algorithm 1 Multi-Agent Distributed DDQN in F-RANs

1: Initialise environment which will run for T time steps and has K UEs, J FAPs with M antennas, J agents $\beta_J = [\beta_1, \beta_2, \cdots, \beta_J]$, and J empty experience replay buffers $\mathcal{E}_j = \emptyset \quad \forall j$
2: **for** $t < T$ **do**
3: **for all** agent j **do**
4: Agent receives $o_j(t)$ and performs action $a_j(t)$
5: **if** $a_j(t) = 1$ **then**
6: FAP j predicts R_k, P_{tot} and δ_k $\forall k$
7: **if** $R_t > R_{\text{min}}$ and $P_{\text{tot}} < P_{\text{th}}$ and $\delta_t < \delta_{\text{th}}$ **then**
8: **if** $a_j(t)$ = 1 and $\sum_t x_{j,k(t)} < M$ **then**
9: Associate with UE $k(t)$
10: **else**
11: Remove UE $k(t-1)$ associate with UE $k(t)$
12: **end if**
13: **end if**
14: **end if**
15: Agent receives reward $r(t)$, stores result in $E_j(t) = \{o_j(t), a_j(t), r(t), o_j(t+1)\}$ and updates $\beta_j \leftarrow$ (18)
16: Agent appends $\mathcal{E}_j \cup E_j(t)$ and transmits $E_j(t)$ to adjacent agents
17: **end for**
18: **for all** agent j **do**
19: j receives experiences and appends them to its own buffer $\mathcal{E}_j \cup \{E_{j-1}(t), E_{j+1}(t)\}$
20: **for all** o' sampled by $U \sim (\mathcal{E}_j)$ **do**
21: j updates $\beta_j \leftarrow$(18)
22: **end for**
23: **end for**
24: **end for**

Algorithm 1 demonstrates how experience sharing is implemented in a multi-agent reinforcement learning (MARL) system operating in fog radio access networks (F-RANs). In this setup, J agents (deployed at fog access points, or FAPs) interact with their respective environments, taking actions $a_j(t)$ based on observations $o_j(t)$ and receiving rewards $r_j(t)$. Each agent maintains an experience replay buffer $\mathcal{E}_j$, which is populated with tuples (o_j, a_j, r_j, o'_j) from its own interactions. Additionally, agents exchange these experiences with neighboring agents to enrich their buffers with diverse off-policy samples.

The algorithm operates as follows:

1) Each agent observes its local environment, selects an action, and stores the resulting experience tuple in its replay buffer (Lines 3–14).
2) Agents share their experiences with neighbors, allowing the buffers to incorporate information from multiple perspectives (Line 15).
3) Using the shared buffer, each agent samples a batch of experiences to update its policy via the loss function in Equation (18). This enables agents to learn from both on-policy and off-policy experiences without incurring significant computational or communication overhead (Lines 16–20).

This decentralised approach avoids the large overheads associated with centralised training methods by limiting communication to compact experience tuples instead of full model or state data. By increasing the diversity of experiences available for training, the algorithm improves generalisation and sample efficiency, which we formalise in the following theorem.

Theorem 1. *Let all agents have identical action spaces, i.e. $\mathcal{A}_i = \mathcal{A}_j$ for any $i, j \in \mathcal{N}$, and observation spaces $\mathcal{O}_i = \mathcal{O}_j \subseteq \mathcal{S}$, where $\mathcal{S}$ is the global state space. At any time t, each agent j observes $o_j(t) \in \mathcal{O}_j$ and takes an action $a_j(t) \in \mathcal{A}_j$. If δ_{sim} is the difference between environment transitions, the environments $\mathbb{E}_i$ and $\mathbb{E}_j$ of agents i and j are sufficiently similar – defined as $\|P_i(o'|o,a) - P_j(o'|o,a)\| \leq \delta_{sim}$ for all $o \in \mathcal{O}, a \in \mathcal{A}$ – sharing experiences improves generalisation, which is quantified as a reduction in the expected value function error (16). This is achieved by filling an agent's local experience replay buffer $\mathcal{B}$ with diverse solutions $a_j^*(o,t)$ for the same observations o from multiple agents. Experience sharing does not require the agents to exchange large model parameters, reducing bandwidth costs. Raw observational data does not need to be exchanged when making a decision either ensuring minimal communication overhead and computational efficiency.*

Proof. This theorem is based on the fact that sharing diverse experiences between agents can improve generalisation in multi-agent partially observable Markov decision processes (POMDPs). While similar results are known for full MDPs, as in A3C [17]. These results do not directly extend to POMDPs due to the interaction of agents with a shared environment. In POMDPs, actions of one agent will affect the environment's dynamics, influencing other agents' later observations and decisions. This interdependence introduces complexities that require further exploration.

Let $\mathcal{O}_j$ denote the observation space for agent j and $\mathcal{B}_j$ represent its local experience replay buffer, where experiences are stored as tuples $E_j(t) = \{o_j(t), a_j(t), r_j(t), o_j(t+1)\}$. In this scenario, each agent relies on its own experience replay buffer $\mathcal{B}_j$, which is limited by the trajectory the agent takes in the environment. This is can cause agents to overfit, leading to bad generalisation when the environment or other agents' behavior changes.

Experience sharing mitigates this issue by exchanging tuples $E_j(t)$ between agents with sufficiently similar environments. Including shared experiences into each agent's replay buffers increases diversity of training samples available for each agent.

This diversity ensures better approximation of the true Q-value function $Q(o,a)^*$, as the expected error in Q-value updates decreases due to a better representation in sample distribution.

From a theoretical perspective, adding diverse off-policy samples improves sample efficiency and reduces the variance of Q-value updates, as shown in prior studies [19]. Since uniform sampling is used in the shared buffer, the computational complexity remains unchanged. Moreover, unlike prior works [18], where off-policy samples led to instability due to outdated policies, our method benefits from sharing experiences generated under policies closer to the current one, as agents operate in sufficiently similar environments.

Algorithm 1 demonstrates how this process is implemented in practice. The results show that shared experiences enable agents to make decisions that lead to a higher reward and converge faster in training. This shows that using shared experiences is a robust approach for improving multi-agent system performance without introducing additional computational overhead.

F. Complexity Analysis

To analyse the complexity of Algorithm 1, we consider the per-agent computational complexity of neural network updates and experience replay, as well as the total communication overhead introduced by experience sharing.

Computational complexity is introduced from experience replay and neural network updates. The experience replay mechanism contributes additional complexity, as each agent samples B experiences from its local replay buffer $\mathcal{E}_j$ of size E_{replay}. If uniform random sampling is used, however, this corresponds to $\mathcal{O}(1)$. For each agent, updating the Double Deep Q-Network involves forward and backward passes through the network. If the network has κ connections and ψ parameters, and processes batches of size B, the per-agent update complexity is:

$$\mathcal{O}\left(B(\kappa+\psi)\right). \tag{19}$$

The communication overhead arises from agents sharing their experience tuples (o_j, a_j, r_j, o'_j) with their neighbors. If each tuple has a size of Tuple_size, and each agent shares B tuples with its $J-1$ neighbors, the communication cost per agent is $\mathcal{O}\left((J-1)B \times \text{Tuple\_size}\right)$. For all J agents, the total communication cost per step is $\mathcal{O}(J(J-1)B \times \text{Tuple\_size})$.

When accounting for bandwidth constraints, and if at each step, an agent will share one tuple ($B = 1$), the overhead introduced by communication scales as:

$$\mathcal{O}\left(\frac{J(J-1) \times \text{Tuple\_size}}{B_{\text{comm}}}\right), \tag{20}$$

where B_{comm} is the available communication bandwidth.

The computational complexity of the algorithm changes linearly with the batch size B, and also depends on the neural network's structure, such as connections κ and parameters ψ. The communication overhead, however, scales quadratically with the number of agents J, since each agent exchanges information with every other agent. Sufficient communication bandwidth can help mitigate delays, but in bandwidth-constrained scenarios, communication overhead may become the bottleneck. Even still, the communication cost does not scale as high as algorithms that exchange model information, such as weights or gradients [36], [44], as the experience tuple is significantly smaller than the network that will be processing it.

IV. Simulation Results

In this section we present the performance of the proposed algorithm compared to a multi-agent individual DDQN (IDDQN) learning method. Both IDDQN and MADDQN agents are defined as neural networks and are implemented using PyTorch. Both consist of a target and estimation neural network, as defined in Fig. 2, which are fully interconnected neural networks as shown in Fig. 4 with an input layer of 5 neurons, 4 hidden layers composed of 64, 128, 3136, 256 neurons in each layer respectively, with the output layer mapping the Q-values $Q(o,a)$ calculated by the hidden layers to the number of actions a and the action with the highest Q-value is chosen. The neural network and can be altered in the process of optimisation to reduce resource requirements and achieve different results. The hidden layers are composed of rectified linear unit (ReLU) neurons, where ReLU is a piecewise linear function. These units are standard for deep learning algorithms as they do not suffer from the vanishing gradient problem, which is common in multi-layer learning algorithms. When inputs arrive at neurons they are summed, and the ReLU function will output the input if it is positive, otherwise, the output will be zero. The final layer results are then split into a value layer, the sum of which provides the predicted Q-value for a decision. Results have been presented for the proposed algorithm after 50,000 training timesteps, as specified in figure captions. At the end of training, the network

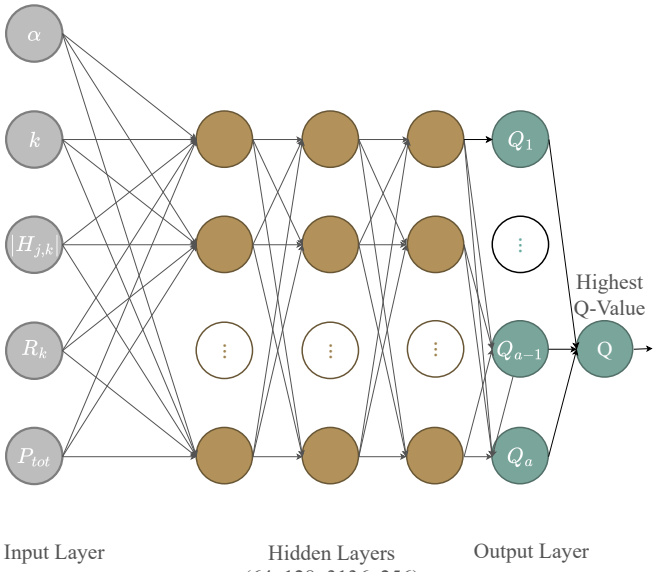


Fig. 4: Neural network used for DDQN and MA-DDQN algorithms.

shows predictable converged behaviour and is operating with 5% chance of taking a random action. An additional corrective experiment has been carried out, with a 3GPP Max-SNR method, as well as by performing corrections to the PyTorch implementation of the agents, which is presented in Figures 5(b), 6, 7(b).

A. Training Results

Agents are deployed in the environment as described in III.B and their neural networks are initialised with arbitrary random parameters at $t = 0$. The learning results are achieved with a discount rate $\rho = 0.99$, learning rate $\mu = 0.00025$, experience batch size $S_{\mathcal{E}} = 512$, pre-training steps $T_{\text{pretrain}} = 512$, probability of random action during training ϵ is chosen in a reducing value over time between the values $[0.99 : 0.05]$, the minimum reward is $r_{\text{min}} = 0$. Simulation parameters for both algorithms are chosen to include $J = 3$ FAPs and agents in the environment, with a max transmit power of $P_T = 30$W, the number of users is $K = 20$ to ensure a quick feedback loop and each user has a minimum rate requirement of $R_{\text{min}} = 5$ Mb/s. The network can consume $P_{\text{tot}} = 1000$ W of power and introduce a delay of $\delta_{\text{th}} = 0.1$s. Training results are shown in Fig. 5 which are achieved from running 2 different training scenarios. In scenario shown in Fig. 5(a) the agent updates the target network every $\tau = 119$ steps. This is done deliberately. High density of target network updates destabilises learning because of high temporal correlation between the two networks. We wish to see if our algorithm accommodates training with inherent lack of stability. It can also be observed that the agent performance drops at the end of training, this has been investigated and was caused due to the agents converging to a local optimum solution, which can be resolved by introducing additional exploration strategy into the behaviour of the agents. This is showcased in the second scenario in which some corrections are perfomed that improve the performance of the deep learning agents. The corrections are as follows:

1) To encourage more diversity in actions, the action space is expanded to be 4 actions [1. Skip, 2. Do not Associate, 3. Associate with 50% probability, 4. Prioritise associate]. While "Skip" and "Do not Associate" both lead to the same outcome, it prevents agents from getting stuck in a suboptimal policy in a stochastic environment [31].
2) Agents in this environment benefit from exploration. The deterministic choice over the actions where the highest Q-value action is chosen is removed. The agents use the Q-values to compute Boltzmann probabilities for each action using a softmax function, which is widely cited in [31].
3) Agents update their estimation networks β every 500 steps. Since the agents are training for 200,000 steps, the rule of thumb is to set the experience replay buffer to be a tenth of the training steps. For a large experience replay buffer, it is more efficient to learn once enough data has been collected. The loss function calculation in Fig. 6 shows how updating every 500 steps improves agent stability in learning.

Algorithm 2 Cache Greedy Algorithm (Pseudocode)

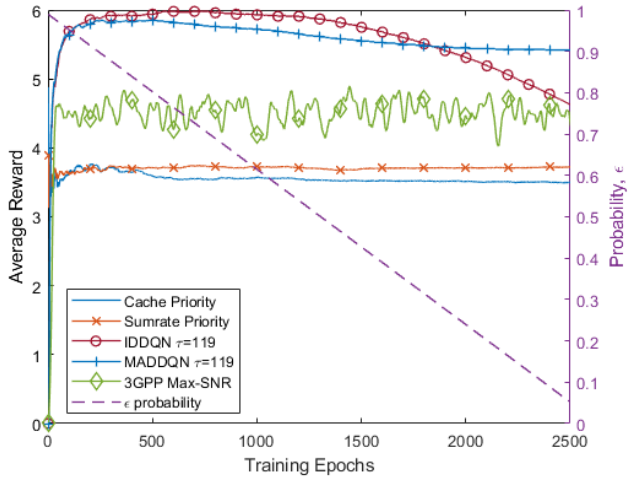
Input: Cache Information (*FileCache*), User File Requests (*indices_user*), List of Access Points (*APs*), Channel Information (*Channel*), Maximum Cache Size.
Output: Selected User Set for Each AP (*SelectedUserSet*).
for each user u in the system **do**
  for each AP n in the system **do**
    if $u \notin SelectedUserSet_n$ **and** *fileRequest*$[u] \in FileCache_n$ **and** Size of $SelectedUserSet_n <$ Max Cache Size **then**
      Update *SumRate* with the new sum rate.
      Update TrialUserSet$_n$: Temporarily add user u to user set for AP n.
      Update *SelectedUserSet*$_n$: Finalise user set associated with AP n.
    end if
  end for
end for

Algorithm 3 Sumrate Greedy Algorithm (Pseudocode)

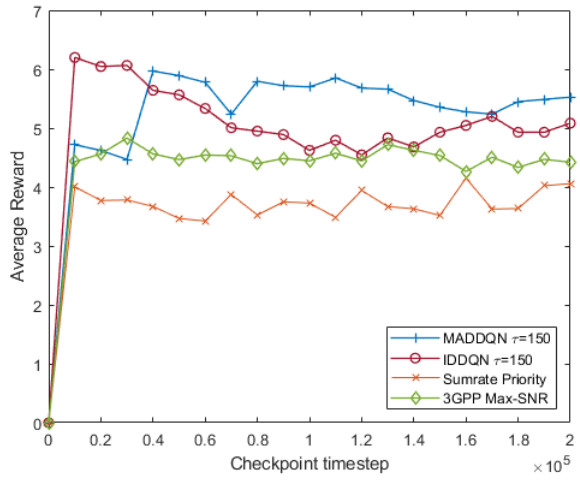
Input: Channel information *Channel*, Signal-to-Noise Ratio (*SNR*), List of Access Points (*APs*), Maximum Power P_{max}.
Output: Selected User Set for Each AP (*SelectedUserSet*).
for each user u in the system **do**
  for each AP n in the system **do**
    if $u \notin SelectedUserSet_n$ **and** $rates\_ZF[u] > rateThreshold$ **and** Sum of Rates $>$ *SumRate* **then**
      Update *SumRate* with the new sum rate.
      Update TrialUserSet$_n$: Temporarily add user u to user set for AP n.
      Update *SelectedUserSet*$_n$: Finalise user set associated with AP n.
    end if
  end for
end for

Results are smoothed out with a cumulative average of the achieved rewards, to eliminate randomness of the results and observe the trend in convergence behaviour of the algorithms. The proposed MA-DDQN algorithm is compared to the individual learning DDQN (IDDQN) algorithm, which is the only contender capable of solving the problem and is used widely in other works. At the same time, suboptimal greedy solutions and a 3GPP Max-SNR algorithm, which chooses the users that have the highest SNR, are provided as a benchmark in exactly the same random environment determined by an equivalent seed. The greedy solutions are implemented as shown in Algorithms 2 and 3. The main difference is the objective of the algorithm – a cache greedy algorithm will greedily choose users that it can minimise the number of hops to the requested content for, while a sumrate greedy algorithm will greedily choose users as long as it can increase the achievable sumrate.

Since the base algorithm that is used for both IDDQN and MA-DDQN is the same, a similar pattern in training is noticeable in the beginning in both Figs. 5(a) and 5(b). It can



(a) Test results for training steps T =50,000.



(b) Test results for training steps T =200,000 with corrections.

Fig. 5: Training results over entire training period for DDQN and MA-DDQN algorithms, higher is better. 1 epoch is equal to 20 training steps.

be seen that the proposed algorithm shows faster convergence and a higher overall reward at the end of training in both scenarios, when probability ϵ is close to 0, while the standard IDDQN algorithm shows a reduced average performance. As a result of more experiences being available, for the same training scenario as IDDQN, the MA-DDQN algorithm shows convergence to a higher reward as training continues. It can be seen from Fig. 5(a) that IDDQN agents diverge from the solution more than they do for the scenario with more stable target network updates. The proposed algorithm does not show a similar deterioration, meaning that experience sharing provides an extra layer of stability when learning. Comparing the results to the Max-SNR algorithm which is closer than the other greedy solutions, it is visible that choosing highest SNR users results in a higher variance. While the number of users oscillates a lot, when smoothed out, it can be seen that on average, the number of users selected by the 3GPP algorithm is lower than by the RL algorithms. This could be explained by the fact that if the users are between multiple FAPs, they have similar SNR, which causes the FAPs have significant overlap between their selected user sets.

In Fig. 5(b) before-mentioned corrections have been performed on the agents. The corrections have successfully achieved a better performance in terms of rewards and users served compared to the initial results. Previously, as seen in 5(a), as the epsilon value reduced, the agents' performance reduced as well, which meant that the agents were converging towards a suboptimal solution. With the corrections, as the epsilon value reduces, the agents have a slight decrease in performance initially, but begin improving when the agents are reliant on their learned solution (around the checkpoint at $t = 180,000$). The results of the corrected agents are slightly larger at the end of training, with IDDQN performing better as well. Compared to Max-SNR, the corrected agents perform 20% better. The corrected results in Fig. 7(b) follow this trend.

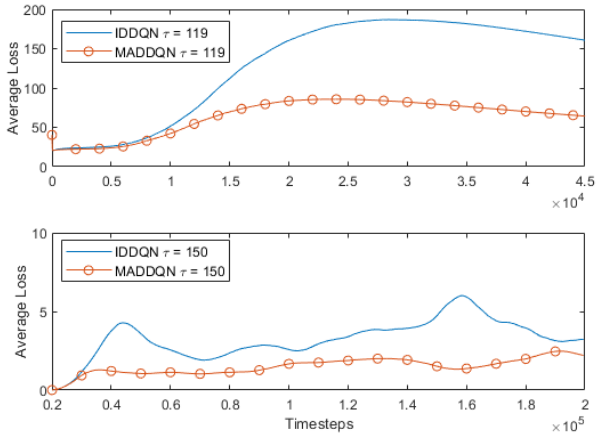


Fig. 6: Difference between predicted reward and received reward over entire training period for DDQN and MA-DDQN algorithms.

Fig. 6 shows a significant statistic to compare the used RL algorithms, which is the learning loss. Learning loss, as stated in (16) shows how much the agent deviates from the predicted reward on a new decision. We see our algorithm show a better average reward at the end of training in Fig. 5, however Fig. 6 shows an approximately 25% better learning performance due to a smaller loss across the entire training period. The two figures are connected. Since loss shows how well the agent has generalised its previous experiences, an algorithm with lower loss is less likely to overestimate or underestimate its future solutions. At the end of training, when the number of target update steps is small ($\tau = 119$), the performance of IDDQN is about 3 times worse than our proposed algorithm. We see a lower loss of our algorithm at the end of training being represented in Fig. 5. When agents are making decisions with $\epsilon = 0.05$ our algorithm trains agents to associate with more users. This proves that exchanging experiences between agents in a POMDP, thus increasing the number of experiences in recent policies stabilises and improves training. We also test how altering the target network update can affect the two

TABLE I: Training results comparison for a large and a small network using MADDQN

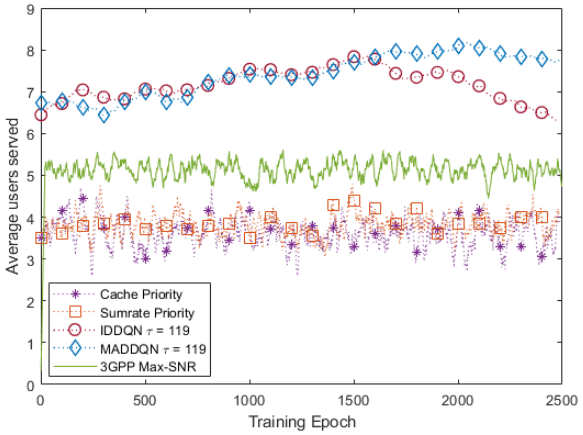
	Users Served	Average Duration	Loss	Single Episode Learning Time
Large Network	7.449	0.0104s	40.1631	0.5625s
Small Network	7.551	0.0109s	42.9084	0.5000s

training algorithms, as it is known that larger time difference between target network updates can stabilise learning further. By performing the proposed corrections to the algorithm we see that the performances of both algorithms improve, the stability increases and the average losses decrease. Although more stable when the target network update happens less often, IDDQN does not catch up to the performance of MADDQN and shows that IDDQN tends to overestimate the reward compared to MADDQN.

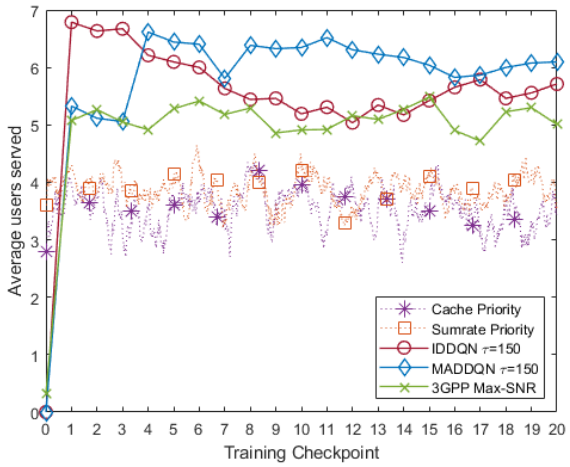
Deep learning network size is a parameter that can be tuned to achieve different results. Using a larger neural network means that the results will take longer to be achieved. It can take careful optimisation to achieve the best results when considering the performance and size trade-off. Table I shows the comparison of results when using a smaller neural network running a MADDQN method. The results labeled "Large Network" are using the neural network described in the preface to the section and shown in Fig. 4. The other results are from a smaller neural network, having only 3 hidden layers between the input and output and they are composed of 64, 128, 256 neurons each. It can be seen in the table that the simulation results do not change significantly. The larger network serves a smaller number of users, but chooses users more accurately than the small network. This is backed up by the fact that the duration that each user experiences is lower than that of the small network, as well as the fact that the loss is lower, meaning that the larger network is making the decision while estimating the result more accurately. Depending on the application, the decision on the size can be more impactful. In the environment that we train the networks in, the extra hidden layer introduces a $0.0625s$ overhead.

B. User selection results

In this and following sub-sections we take a look at practical results. In this sub-section we explain how the reward in Fig. 5 gets influenced by the number of users that have been successfully served by the system. The system has 20 UEs in total. In this scenario, FAPs have a static cache holding 3 possible cacheable objects out of a total 4. UEs randomly choose 1 out of the 4 files and make a request to the system. In this scenario with a limited size cache, some files are not present at some FAPs and they have to be acquired from the cloud/Internet. If the file reaches the UE completely with specified constraints, the UE is denoted as being served by the system. IDDQN algorithm is deployed in the environment for comparison with MADDQN algorithm as well as simple greedy methods that are aiming to serve users with close cache proximity or with higher sumrate. Greedy algorithms fail to adapt to the environment and converge to an average 4 users served despite what is being prioritised. Despite



(a) Test results T =50,000, $\tau = 119$



(b) Test Results T =200,000, $\tau = 150$

Fig. 7: Simulation output showing the number of UEs served under required constraints.

seemingly random performance looking at the greedy algorithms we can see when we would expect better performance from deep learning methods. A peak in greedy algorithm performance is indicative of favourable conditions for deep learning algorithms as well, since the environment seed is the same for all algorithms. In Fig. 7(a), as in the reward for 3 cacheable objects (c.o.), the number of users served converges to a solution at the end of training. When the updates of target network are frequent, i.e. $\tau = 119$ the MADDQN algorithm learns about the environment better. Because of a higher loss overall in this scenario, the opportunities to diverge benefit MADDQN algorithm and provide it with more learning experiences. DDQN performs about 40% worse at the end of training, showing that the algorithm finds some strategies for associating with different users across 3 FAPs, but the strategy is not as good as the one found by MADDQN algorithm. A similar conclusion can be drawn for Fig. 7(b). The IDDQN algorithm comes close to the number of users served, while the Max-SNR algorithm is about 20% worse due to its inability to account for the cache placement in FAPs. When the number of users served is compared to the reward, it can be seen

that there is a bigger difference in the reward than there is in the users selected. This is caused by the fact that IDDQN chooses less optimal sets of users and results in higher delay and power consumption compared to MADDQN. This is in line with Fig. 6, where it is shown that IDDQN has worse estimation capabilities than MADDQN.

In addition to exploring how changing hyperparameters of training can affect the agents, it is also possible to change the parameters of the environment. One of the significant parameters to the system is the cache availability. We can alter the number of cached objects in the FAPs to observe if that improves or degrades the performance of deep learning algorithms. We already have a reference for greedy algorithms, so we remove their curves with curves of algorithms under test for a reduced number of c.o, 1 c.o. instead of 3. The results for this, shown in Fig. 8 are compared to the performance of 3GPP standard Max-SNR algorithm. When algorithms converge, the trend follows the fact that when the number of cached objects decreases, the performances of both algorithms decrease as well. However, the significant improvement of MADDQN is easily visible. When the number of c.o. decreases from 3 to 1, the performance falls only slightly from 7.75 users served to 7.5. That is a large performance increase over IDDQN, showing that the performance decreased from about 6.5 users to less than 5 users served as the cached object number decreases. We also see that just like in Figure 5(b) the proposed algorithm outperforms the industry standard Max-SNR algorithm. During the training when exploration takes place, the proposed algorithm can find solutions that significantly outperform Max-SNR and have less variance between the number of users served step-by-step. At the end of training the proposed algorithm serves as many users as the Max-SNR algorithm.

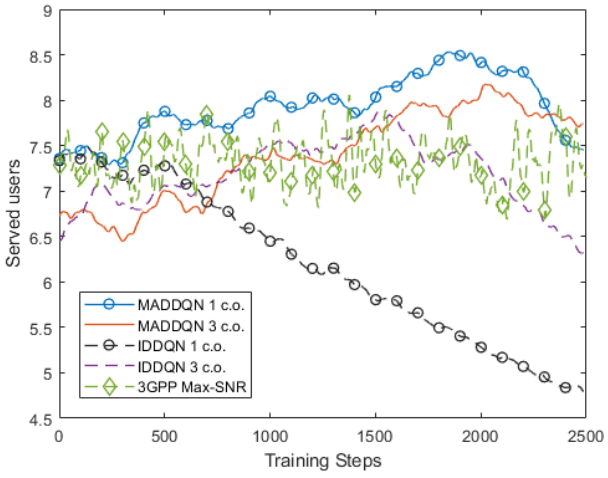


Fig. 8: Simulation output showing the number of users selected when the number of cached objects changes.

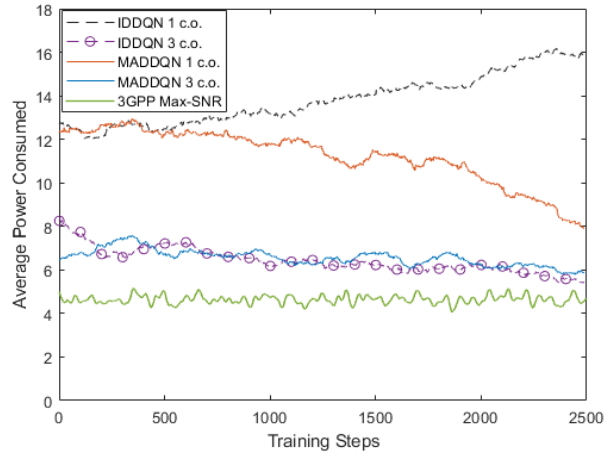


Fig. 9: Simulation output showing the average powers consumed per user. Training steps T = 50000, $\tau = 119$.

C. Power consumption and service delay results

Power consumption efficiency of deep learning algorithms for different numbers of c.o. is shown in Fig. 9. To generate these results, the instantaneous power consumption of each FAP is divided by the number of users served

$$P_{avg} = \frac{\left(P_j^{\text{TRI}} + \sum_k P_{j,k}^{\text{XH}}\right)}{x_d}. \tag{21}$$

This normalisation shows how well the solution of each algorithm chooses the users to meet the power consumption constraint. As the cache availability increases, the power consumption performance of both deep learning algorithms converges. The performance of the IDDQN algorithm is nearly identical to MADDQN, however referring to Figs. 7(a) and 7(b) it will become obvious that while the power consumption for IDDQN is identical to MADDQN, it does have additional power to spend to associate with more users. However, MADDQN performance is significantly better when the number of c.o. is lower. The performance of IDDQN degrades as the training continues and when the agent is making most of the decisions it fails to complete requests for users while maintaining a good power efficiency. This is consistent with Figs. 5 and 7, as the reward begins to drastically fall at the end of training, while the agent is making overestimations for its decisions. As Max-SNR algorithm is comparable in performance for selected users, we can see that the small number of selected users by the algorithm results in a lower average per-user power consumption compared to both RL algorithms. This means that the algorithm, even if it chooses less users, the users selected do not require as much power as the users selected by RL algorithms.

The delay experience by the users shows more performance gain when MADDQN is used. When the number of cached objects is high and low, MADDQN finds users to serve to minimise the possible delay which increases the resulting reward. IDDQN fails to find similar user sets. When comparing the average durations, IDDQN is $2ms$ behind MADDQN on convergence and when the number of cached objects is low, IDDQN performance is drastically decreased. As the number of cached files increases, the number of good solutions becomes closely linked to the channel quality experienced by the UEs. However, the delay will also be reduced as it is completely dependent on the high-speed wireless transmission

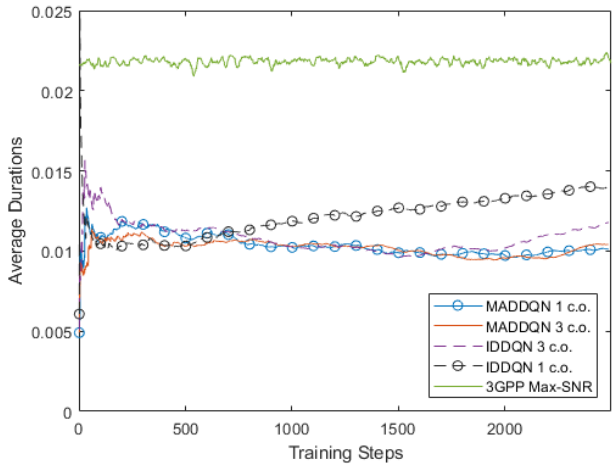


Fig. 10: Delay performance for 1 and 3 cacheable objects (c.o.).

links and is not bottlenecked by the x-haul processing. Consequently, it becomes harder to find optimal UE combinations that would provide the smallest delay. How this principle affects different algorithms is shown in Fig. 10. The delay is related to the number of UEs served, as an increase in the size of the served UE set means that the power (and achieved rate) is split between the served UEs, consequently increasing the delay. As shown for greedy algorithms (cache priority and sum-rate priority in the figure), when the number of UEs in the system is large, the delay increases. Just as for power consumption analysis it is preferred that if the number of served UEs is large, the delay should be large as well, as it shows how well the algorithms use the allocated thresholds. The performance of Max-SNR algorithm is opposite of what's observed in the power consumption figure. Max-SNR seems to miss out on users that would benefit from cached objects. Since APs have only 1 c.o. in that scenario, the Max-SNR users are less likely to be served by an AP that has their desired content.

D. Latency threshold impact

This subsection highlights how the system performs under varying latency constraints using the different methods. Values for the proposed deep learning method are using the model that was generated at the last training checkpoint T =200,000. For the Max-SNR algorithm, the results are the average number of users served over 20 runs. The results can be used to evaluate the trade-offs between different algorithms and how well they can adapt to latency tolerances. It would be expected that as latency increases, so does the number of users served. The results show that the proposed algorithm, MADDQN, outperforms Max-SNR for all latency thresholds, especially when the latency threshold is larger. Combining the results in Table II and Fig. 8 mean that using the proposed method, agents generalise better under various network conditions, whether it is the latency or number of cached objects.

Latency Threshold $\delta_{\text{th}}(s)$	Max-SNR	MADDQN
0.001	5.100	5.346
0.05	5.100	6.519
0.1	5.128	6.096

TABLE II: Comparison of Users Served under Latency Threshold with Max-SNR, and MADDQN

V. Conclusion

In this paper, we have presented the F-RAN system as a POMDP environment, which has stochastic outcomes based on agent decisions. It has been shown that low-complexity mechanisms that would typically be used to save computation and transmission overheads do not perform well, as they result in optimising local greedy solutions. To improve the system performance, we have proposed a novel method called MADDQN to reduce training times, improve learning stability and improve the reward received on convergence. We showed that our method performs better than greedy algorithms and the DDQN algorithm, when deployed in the same environment. The algorithms were simulated in a network with a small number of antennas and a comparably large number of UEs, a varying number of cached objects and by altering the frequency of target network updates to measure the stability of our algorithm. The simulation results have validated that the proposed method performs better than the DDQN algorithm, improving the reward at the end of training and following the results of training losses we presented that MADDQN is significantly more stable during training. MADDQN outperforms DDQN in all metrics, i.e. number of users served, power consumption and user experienced delay.

References

[1] "Mobile data traffic outlook." [Online]. Available: https://www.ericsson.com/en/reports-and-papers/mobility-report/dataforecasts/mobile-traffic-forecast

[2] C. Pan, M. Elkashlan, J. Wang, J. Yuan, and L. Hanzo, "User-centric c-ran architecture for ultra-dense 5g networks: Challenges and methodologies," *IEEE Communications Magazine*, vol. 56, no. 6, pp. 14–20, 2018.

[3] G. Interdonato, E. Björnson, H. Quoc Ngo, P. Frenger, and E. G. Larsson, "Ubiquitous cell-free massive mimo communications," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, no. 1, p. 197, Aug 2019. [Online]. Available: https://doi.org/10.1186/s13638-019-1507-0

[4] X. Huang, G. Xue, R. Yu, and S. Leng, "Joint scheduling and beamforming coordination in cloud radio access networks with qos guarantees," *IEEE Transactions on Vehicular Technology*, vol. 54, no. 7, pp. 5449–5460, 2016.

[5] C. Pan, H. Zhu, N. J. Gomes, and J. Wang, "Joint precoding and rrh selection for user-centric green mimo c-ran," *IEEE Transactions on Wireless Communications*, vol. 16, no. 5, pp. 2891–2906, 2017.

[6] X. Li, K. Jiao, X. Chen, H. Ding, J. Wang, and M. Pan, "Demand-oriented fog-ran slicing with self-adaptation via deep reinforcement learning," *IEEE Transactions on Vehicular Technology*, vol. 72, no. 11, pp. 14 704–14 716, 2023.

[7] F. Guo and M. Peng, "Efficient mobility management in mobile edge computing networks: Joint handover and service migration," *IEEE Internet of Things Journal*, vol. 10, no. 20, pp. 18 237–18 247, 2023.

[8] M. You, G. Zheng, T. Chen, H. Sun, and K.-C. Chen, "Delay guaranteed joint user association and channel allocation for fog radio access networks," *IEEE Transactions on Wireless Communications*, vol. 20, no. 6, pp. 3723–3733, 2021.

[9] T. H. L. Dinh, M. Kaneko, E. H. Fukuda, and L. Boukhatem, "Energy efficient resource allocation optimization in fog radio access networks with outdated channel knowledge," *IEEE Transactions on Green Communications and Networking*, vol. 5, no. 1, pp. 146–159, 2021.

[10] M. Wang, J. Wang, Y. Kai, F. Xia, X. Zeng, and F. Liu, "User association and power allocation in multi-connectivity enabled millimeter-wave networks with limited backhaul," *IEEE Open Journal of the Communications Society*, vol. 4, pp. 1761–1773, 2023.

[11] A. Nassar and Y. Yilmaz, "Deep reinforcement learning for adaptive network slicing in 5g for intelligent vehicular systems and smart cities," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 222–235, 2022.

[12] X. Cao, C. Hu, and S. Yan, "A matching game approach for joint resource allocation and user association in fog radio access networks," in *2018 IEEE/CIC International Conference on Communications in China (ICCC Workshops)*, 2018, pp. 277–281.

[13] S. Yan, M. Jiao, Y. Zhou, M. Peng, and M. Daneshmand, "Machine-learning approach for user association and content placement in fog radio access networks," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9413–9425, 2020.

[14] F. Tütüncüoğlu and G. Dán, "Optimal service caching and pricing in edge computing: A bayesian gaussian process bandit approach," *IEEE Transactions on Mobile Computing*, vol. 23, no. 1, pp. 705–718, 2024.

[15] C. Liu, F. Tang, Y. Hu, K. Li, Z. Tang, and K. Li, "Distributed task migration optimization in mec by extending multi-agent deep reinforcement learning approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1603–1614, 2021.

[16] L. Oliveira Souza, G. de Oliveira Ramos, and C. Ghedini Ralha, "Experience sharing between cooperative reinforcement learning agents," in *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2019, pp. 963–970.

[17] F. Christianos, L. Schäfer, and S. Albrecht, "Shared experience actor-critic for multi-agent reinforcement learning," in *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*. Curran Associates Inc, Dec. 2020, pp. 10 707–10 717, thirty-fourth Conference on Neural Information Processing Systems, NeurIPS 2020 ; Conference date: 06-12-2020 Through 12-12-2020. [Online]. Available: https://nips.cc/Conferences/2020

[18] S. Ahilan and P. Dayan, "Correcting experience replay for multi-agent communication," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=xvxPuCkCNPO

[19] N. Zhao, Y.-C. Liang, D. Niyato, Y. Pei, M. Wu, and Y. Jiang, "Deep reinforcement learning for user association and resource allocation in heterogeneous cellular networks," *IEEE Transactions on Wireless Communications*, vol. 18, no. 11, pp. 5141–5152, 2019.

[20] J. N. Foerster, N. Nardelli, G. Farquhar, T. Afouras, P. H. S. Torr, P. Kohli, and S. A. Whiteson, "Stabilising experience replay for deep multi-agent reinforcement learning," in *International Conference on Machine Learning*, 2017.

[21] J. Yu, S. Zhu, and D. Kilper, "Evolution to mesh 5g x-haul networks," in *2020 Optical Fiber Communications Conference and Exhibition (OFC)*, 2020, pp. 1–3.

[22] H. Q. Ngo, A. Ashikhmin, H. Yang, E. G. Larsson, and T. L. Marzetta, "Cell-free massive mimo versus small cells," *IEEE Transactions on Wireless Communications*, vol. 16, no. 3, pp. 1834–1850, 2017.

[23] M. M. Do and H. J. Son, "CoMP (1): CoMP Types - CS, CB, JT and DPS," 2014, Accessed 20 Feb, 2020. [Online]. Available: https://www.netmanias.com/en/post/blog/6558/comp-lte-lte-a/comp-1-comp-types-cs-cb-jt-and-dps

[24] S. Schneider, H. Karl, R. Khalili, and A. Hecker, "Multi-agent deep reinforcement learning for coordinated multipoint in mobile networks," *IEEE Transactions on Network and Service Management*, vol. 21, no. 1, pp. 908–924, 2024.

[25] W. Wong and S. . G. Chan, "Distributed joint ap grouping and user association for mu-mimo networks," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, April 2018, pp. 252–260.

[26] S. Sun, T. S. Rappaport, S. Rangan, T. A. Thomas, A. Ghosh, I. Z. Kovacs, I. Rodriguez, O. Koymen, A. Partyka, and J. Jarvelainen, "Propagation path loss models for 5g urban micro- and macro-cellular scenarios," in *2016 IEEE 83rd Vehicular Technology Conference (VTC Spring)*, May 2016, pp. 1–6.

[27] S. Chen, J. Zhang, J. Zhang, E. BjÃ¶rnson, and B. Ai, "A survey on user-centric cell-free massive mimo systems," *Digital Communications and Networks*, vol. 8, no. 5, pp. 695–719, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352864821001024

[28] Z. Tang, K. Yu, G. Yang, L. X. Cai, and H. Zhou, "New bridge to cloud: An ultra-dense leo assisted green computation offloading approach," *IEEE Transactions on Green Communications and Networking*, vol. 7, no. 2, pp. 552–564, 2023.

[29] W. Zhu, H. D. Tuan, E. Dutkiewicz, and L. Hanzo, "Collaborative beamforming aided fog radio access networks," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 7, pp. 7805–7820, 2022.

[30] X. Jiang, H.-C. Chao, G.-M. Muntean, G. Ghinea, and C. Xu, "Green communication for mobile and wireless networks," *Mobile Information Systems*, vol. 24, no. 6, pp. 120–127, 2016.

[31] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[32] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lactot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, vol. 48, pp. 1–15, 2016.

[33] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015.

[34] J. Zhao, T. Q. S. Quek, and Z. Lei, "Coordinated Multipoint Transmission with Limited Backhaul Data Transfer," *IEEE Transactions on Wireless Communications*, vol. 12, no. 6, pp. 2762–2775, June 2013.

[35] J. Yu and M. Dong, "Distributed low-complexity multi-cell coordinated multicast beamforming with large-scale antennas," in *2018 IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, 2018, pp. 1–5.

[36] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konecny, S. Mazzocchi, H. B. McMahan *et al.*, "Towards federated learning at scale: System design," *arXiv preprint arXiv:1902.01046*, 2019.

[37] X. Cao, T. BaÅar, S. Diggavi, Y. C. Eldar, K. B. Letaief, H. V. Poor, and J. Zhang, "Communication-efficient distributed learning: An overview," *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 4, pp. 851–873, 2023.

[38] M. Sana, A. De Domenico, W. Yu, Y. Lostanlen, and E. Calvanese Strinati, "Multi-agent reinforcement learning for adaptive user association in dynamic mmwave networks," *IEEE Transactions on Wireless Communications*, vol. 19, no. 10, pp. 6520–6534, 2020.

[39] C. Wang, Y. Sun, and Y. Ren, "Distributed user association for computation offloading in green fog radio access networks," in *2020 Information Communication Technologies Conference (ICTC)*, 2020, pp. 75–80.

[40] R. Bellman, R. Corporation, and K. M. R. Collection, *Dynamic Programming*, ser. Rand Corporation research study. Princeton University Press, 1957. [Online]. Available: https://books.google.co.uk/books?id=wdtoPwAACAAJ

[41] T. P. Le, N. A. Vien, and T. Chung, "A deep hierarchical reinforcement learning algorithm in partially observable markov decision processes," *IEEE Access*, vol. 6, pp. 49 089–49 102, 2018.

[42] B. Chalaki and A. A. Malikopoulos, "A hysteretic q-learning coordination framework for emerging mobility systems in smart cities," in *2021 European Control Conference (ECC)*, 2021, pp. 17–22.

[43] Y. S. Nasir and D. Guo, "Multi-agent deep reinforcement learning for dynamic power allocation in wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 10, pp. 2239–2250, 2019.

[44] P. Liu, T. Gao, and C. Li, "Optimization of federated learning communications with heterogeneous quantization," in *2022 IEEE 22nd International Conference on Communication Technology (ICCT)*, 2022, pp. 292–296.