future internet

MDPI

Article

A Transfer Reinforcement Learning Approach for Capacity Sharing in Beyond 5G Networks

Irene Vilà *, Jordi Pérez-Romero and Oriol Sallent

Signal Theory and Communications Department (TSC), Universitat Politècnica de Catalunya (UPC), 08034 Barcelona, Spain; jordi.perez-romero@upc.edu (J.P.-R.); sallent@tsc.upc.edu (O.S.)
* Correspondence: irene.vila.munoz@upc.edu

Abstract: The use of Reinforcement Learning (RL) techniques has been widely addressed in the literature to cope with capacity sharing in 5G Radio Access Network (RAN) slicing. These algorithms consider a training process to learn an optimal capacity sharing decision-making policy, which is later applied to the RAN environment during the inference stage. When relevant changes occur in the RAN, such as the deployment of new cells in the network, RL-based capacity sharing solutions require a re-training process to update the optimal decision-making policy, which may require long training times. To accelerate this process, this paper proposes a novel Transfer Learning (TL) approach for RL-based capacity sharing solutions in multi-cell scenarios that is implementable following the Open-RAN (O-RAN) architecture and exploits the availability of computing resources at the edge for conducting the training/inference processes. The proposed approach allows transferring the weights of the previously learned policy to learn the new policy to be used after the addition of new cells. The performance assessment of the TL solution highlights its capability to reduce the training process duration of the policies when adding new cells. Considering that the roll-out of 5G networks will continue for several years, TL can contribute to enhancing the practicality and feasibility of applying RL-based solutions for capacity sharing.

Keywords: RAN slicing; capacity sharing; reinforcement learning; transfer learning; transfer reinforcement learning

Citation: Vilà, I.; Pérez-Romero, J.; Sallent, O. A Transfer Reinforcement Learning Approach for Capacity Sharing in Beyond 5G Networks. *Future Internet* **2024**, *16*, 434. https://doi.org/10.3390/fi16120434

Academic Editor: Djamel Djenouri

Received: 9 October 2024
Revised: 15 November 2024
Accepted: 18 November 2024
Published: 21 November 2024

1. Introduction

Network slicing was introduced in the 5G architecture to support multiple services over a common network infrastructure by associating each of them with an end-to-end logical network optimized to its requirements. This multi-service perspective of 5G is envisaged to be inherited in future 6G systems, enabling applications such as unmanned aerial vehicle networks, remote surgery, or augmented reality, originally identified for 5G but not realized in practice [1]. Therefore, network slicing is also expected to be a fundamental feature of 6G, so enhanced, robust, and automated solutions will be required to face the complexity needed to guarantee the diverse and multiple end-to-end quality of service (QoS) requirements of all the slices.

The realization of network slicing in the Radio Access Network (RAN) requires capacity sharing solutions that efficiently distribute the available radio resources at the different cells among the different RAN slices to fulfill their traffic demands and QoS requirements. The complexity of this task, due to the rapidly varying traffic demands and the multiplicity of QoS requirements, has motivated the use of Artificial Intelligence (AI)/Machine Learning (ML) techniques to address the capacity sharing problem through the combination of Artificial Neural Networks (ANNs) and Reinforcement Learning (RL) [2–12]. This is due to the ability of these techniques to deal with dynamic decision-making problems with large state and action spaces. In addition to the capacity sharing problem, these techniques have also been applied to many other problems, such as integrated unmanned aerial vehicle sensing [13,14] or autonomous driving [15].

In general, the practical application of RL-based solutions involves a training stage used to learn the optimal decision-making policy for a given environment, followed by an inference stage in which the learned policy is applied [16]. However, whenever a mismatch between the training and the inference environments occurs, the learned policy will no longer be optimal and can even lead to significant performance degradation. In such a case, a re-training process is needed to learn a new optimal decision-making policy. This mismatch is relevant in capacity sharing in the RAN because the operator will very often perform changes in the network topology, for example by deploying new cells to improve the coverage footprint or to increase the deployed capacity. Therefore, given the massive deployment of 5G and eventually 6G networks, addressing the AI/ML model re-training deserves careful attention for developing scalable and robust solutions that materialize the embedding of AI/ML into future communication networks.

Carrying out a re-training process in RL can take a lot of time, as it typically involves acquiring a large number of state/action/reward experiences with the new environment. To address this challenge, Transfer Learning (TL) techniques are identified as key tools [17], since they allow accelerating the training process by reusing previous knowledge of a source task to perform a new target task. In this way, the re-training does not have to be done from scratch but can start from the previously learned policy and update it according to the new environment.

With all the above, the main contribution of this paper is the proposal of a new TL approach for re-training an RL-based capacity sharing policy in multi-cell scenarios when expanding the RAN with new cells. To this end, we leverage the multi-cell RL-based capacity sharing solution proposed in our previous work [11,12] that makes use of Multi-Agent Reinforcement Learning (MARL) with the Deep Q-Network (DQN) technique, and it is referred to here as the DQN-MARL algorithm. A key feature of this algorithm is that it associates one DQN agent with each slice, and this agent learns the policy that assigns the capacity to the slice in each of the cells in the scenario. Thus, it is a solution that jointly manages multiple cells. As discussed in [11], this approach brings substantial benefits with respect to single-cell decision-making processes, in which capacity sharing is managed independently at each cell.

The potential of TL to accelerate the re-training in RL-based capacity sharing solutions has been highlighted by some available works in the literature, where TL has shown to be effective in tackling different use cases. The authors in [18,19] address drastic changes in traffic demand that the RL policy is not generic enough to adapt to, so they require a re-training. To support this re-training, the authors propose the use of the policy reuse TL technique. Instead of re-training the policy for the new situation from scratch, the ANN values and architecture of the old policy are reused as the initial policy in the new situation. Then, the transferred policy can be fine-tuned to the new scenario conditions, thus reducing the re-training time. The works in [20,21] exploit TL to take a policy learned for a cell and then transfer it to be applied as the initial policy to another cell. Then, the initial policy is re-trained with specific data from this other cell. In [20], the policy reuse TL approach is used, while policy distillation is employed in [21]. In turn, refs. [22,23] propose to exploit TL to accelerate the simultaneous training of multiple cells by learning a generalist policy that uses the experiences of multiple cells. Then, the knowledge of this generalist policy is transferred to learn local cell-specific policies, using the TL policy reuse technique.

In this paper, TL is proposed to deal with the problem of the deployment of new cells, which, to the authors' best knowledge, has not yet been addressed in the literature. The deployment of new cells presents some challenges concerning RL-based capacity sharing solutions. On the one hand, there is the need to extend the intelligence of the RL-based capacity sharing solutions to manage the newly deployed cells. On the other hand, the deployment of a new cell will lead to changes in traffic demands and radio conditions of the neighboring cells (i.e., traffic demand will be redistributed among cells), leading to significant changes from a multi-cell scenario perspective. The TL approach proposed here addresses these two challenges for RL-based capacity sharing solutions where an RL

agent jointly manages multiple cells, as in our work in [11]. For this kind of solution, the number of inputs and outputs of the ANN that constitutes the decision-making policy in RL depends on the number of cells. Therefore, a change in the number of deployed cells involves a change in the structure of this ANN. Thus, a novel aspect of the proposed TL approach in this paper is that it supports the transfer of weights between ANNs with a different number of inputs and outputs by using a TL technique referred to as inter-task mapping. Note that the fact that the proposed TL approach is designed for RL-based capacity sharing solutions where an agent deals with multiple cells is a distinctive aspect of this paper, since the works in [18–23] have explored the use of TL for RL agents that make capacity sharing decisions for only a single cell. The results obtained in this paper illustrate the potential of the TL solution to accelerate the training process when increasing the number of cells, contributing to the practicality and feasibility of applying RL-based solutions for capacity sharing. In addition, the paper presents a functional framework for implementing the solution in the O-RAN architecture and discusses the deployment of the related O-RAN functionalities at the edge.

The rest of the paper is organized as follows. Section 2 presents the system model and formally defines the problem addressed in the paper. Section 3 describes the proposed TL approach by first defining the states, actions, and rewards in the source and target tasks and then describing the proposed TL inter-task mapping approach. Afterwards, Section 4 provides the results, and Section 5 discusses the implementation considerations for the solution. Finally, Section 6 summarizes the conclusions extracted from the paper.

2. System Model and Problem Definition

The considered system consists of a RAN infrastructure that is owned by an Infrastructure Provider (InP). It is initially composed of N cells that use the 5G New Radio (NR) technology with diverse deployment characteristics (i.e., cell radius, transmission power, frequency of operation). Cell n has a total of W_n Physical Resource Blocks (PRBs), each with bandwidth B_n, providing a total cell capacity c_n (b/s), expressed as:

$$c_n = W_n \cdot B_n \cdot S_n \tag{1}$$

where S_n is the average spectral efficiency of the cell. The total system capacity C is obtained by aggregating c_n for all cells $n = 1...N$. The InP shares its RAN infrastructure among K tenants by providing each tenant with a RAN Slice Instance (RSI). The provisioning of RSIs to each tenant is subject to the definition of certain Service Level Agreements (SLAs) between each tenant and the InP (see [11] for details).

To dynamically adapt the capacity assigned to the different tenants to their spatial and temporal traffic variations among the different cells, minimize SLA breaches (i.e., violations) in the system, and optimize the resource utilization of the different cells in the system, the DQN-MARL capacity sharing solution in [11] is considered. In this solution, a DQN agent learns the policy π that tunes the capacity share of a tenant in the N cells in time steps of duration Δt (in the order of minutes). Specifically, at time step t the DQN agent provides the capacity share $\boldsymbol{\sigma(t)}$, which is defined as:

$$\boldsymbol{\sigma(t)} = [\sigma_1(t),..., \sigma_n(t), ..., \sigma_N(t)] \tag{2}$$

where each component $\sigma_n(t)$ is the proportion of the total PRBs W_n in cell n assigned to the tenant during time step t.

In this context, let us assume that the InP decides to deploy ΔN new cells to improve the coverage and capacity of the RAN infrastructure in the area so that the resulting number of deployed cells in the system is $N' = N + \Delta N$. To be able to distribute the resulting capacity C' in the scenario considering the new cells, the DQN-MARL capacity sharing model needs to be updated to learn the new policy π', since the previous policy π can only distribute the capacity among N cells. This constitutes a practical problem since re-training the policies for each tenant from scratch can be highly consuming both in terms of time and

computing resources. To deal with this problem, this paper proposes the use of a Transfer Reinforcement Learning (TRL) approach for capacity sharing, which is described in further detail in the following section.

3. Transfer Reinforcement Learning Approach

TL techniques leverage the knowledge gained for a source task (i.e., the origin task that has already been previously learned) to accelerate the training process of a target task (i.e., the task to be learned). While TL has been extensively studied for supervised learning, its application to RL is an emerging trend that leads to a subtype of TL techniques known as TRL [24]. TRL deals with the application of TL to RL, in which an RL agent iteratively interacts with an environment to learn an optimal decision-making policy. The knowledge to be transferred in RL can take different forms, depending on the RL component that is transferred, e.g., policies, ANN weights, experiences from the RL agent interaction, etc.

This section proposes using TRL to accelerate the re-training process of the DQN-MARL solution when extending the number of cells in a RAN infrastructure. The proposed TRL approach, which is depicted in Figure 1, considers as a source task the policy π for capacity sharing in a scenario with N cells and as a target task the policy π' for the case of $N' = N + \Delta N$ cells, where both policies π and π' rely on the DQN-MARL capacity sharing solution in [11]. In the source task, a DQN agent iteratively interacts with the RAN environment of N cells to learn the policy π. In each step of this training process, the DQN agent obtains the *state* of the environment, which is provided by the performance monitoring module. The agent then selects the most appropriate *action* according to the current policy π available at the DQN agent, and this action is applied to the environment. In the following step, the DQN agent receives a *reward* from the performance monitoring module. This reward assesses the suitability of the previous action and is used by the DQN agent to update the policy π. This process is repeated until a certain termination condition is reached. In the case of the target task, the training process of policy π' is sped up by leveraging the previous knowledge gained during the training process of policy π. In this case, the knowledge from policy π is first transferred to policy π', and then policy π' is refined through the interaction of the DQN agent with the new environment consisting of N' cells, as described for the source task. This knowledge transfer occurs according to the state, action, and reward definitions of the RL agents in the source and target tasks, which are formulated in Section 3.1, and follows a specific TRL technique referred to as inter-task mapping, which is described in Section 3.2.

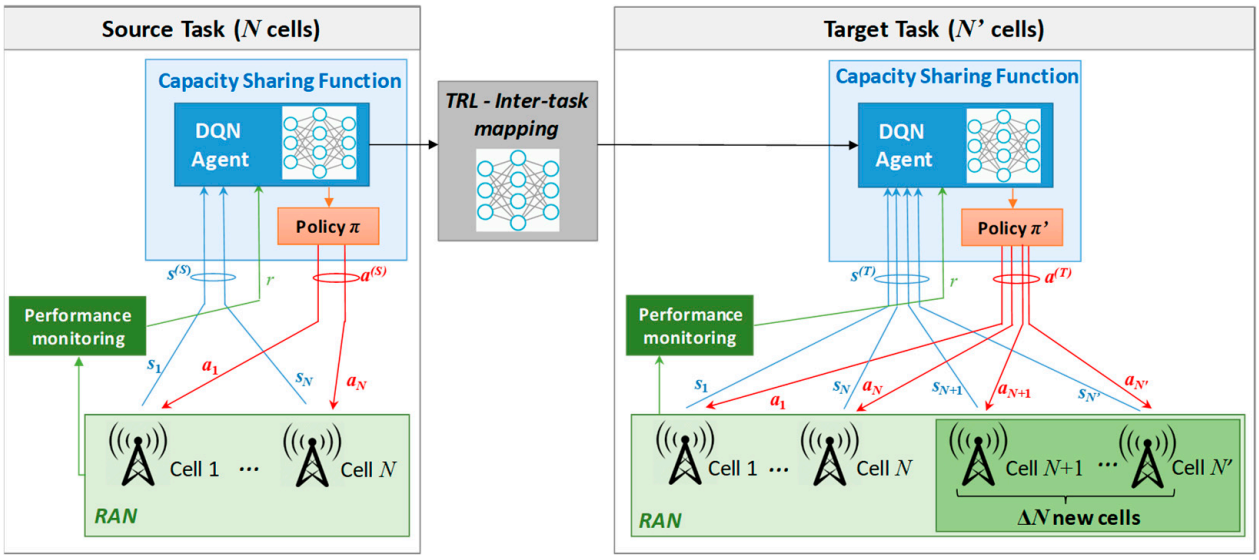

Figure 1. TRL approach for relearning a new policy after increasing the number of cells.

3.1. States, Actions, and Reward in Source and Target Tasks

The particularized definitions of state, action, and reward in the source and target tasks for an agent of the DQN-MARL capacity sharing solution are presented in the following. The presented definitions correspond to the agent associated with tenant k at time step t, but for simplicityin the notation, the dependency on k and t is omitted.

- State: The state in the source task is defined as $s^{(S)} = \{s^N, s^{SLA}\}$, where $s^N = \{s_n \mid n = 1...N\}$ includes the state components of the tenant in the N cells, denoted as vector s_n for the n-th cell, and s^{SLA} includes the state components that reflect the SLA of the tenant. In turn, the state in the target task is defined as $s^{(T)} = \{s^N, s^{\Delta N}, s^{SLA}\}$, where $s^{\Delta N} = \{s_n \mid n = N + 1...N'\}$ includes the state of the newly deployed cells.
- Action: One action in the source task includes N components a_n, $n = 1...N$, each one associated with one cell. a_n tunes the capacity share $\sigma_{k,n}(t)$ to be applied in the following time step in the n-th cell and can take three different values $a_n \in \{\Delta, 0, -\Delta\}$, corresponding to increasing the capacity share by a step of Δ, maintaining it, or decreasing it by a step of Δ, respectively. As a result, the action space $\mathcal{A}^{(S)}$ in the source task has 3^N possible actions, and the i-th action $a^{(S)}(i) \in \mathcal{A}^{(S)}$ is denoted as $a^{(S)}(i) = \{a_n(i) \mid n = 1...N\}$, for $i = 1,..., 3^N$. Correspondingly, the action space of the target task $\mathcal{A}^{(T)}$ has $3^{N'}$ possible actions, and the d-th action is denoted as $a^{(T)}(d) = \{a_n(d) \mid n = 1...N'\}$, for $d = 1,..., 3^{N'}$. An action in the target task can be decomposed into two parts: $a^{(T)}(d) = \{a^N(d), a^{\Delta N}(d)\}$, where $a^N(d) \in \mathcal{A}^{(S)}$ includes the components of the initial N cells (so it is one of the actions of the source task), and $a^{\Delta N}(d) = \{a_n(d) \mid n = N + 1...N'\}$ includes the components for the newly deployed cells.
- Reward: The reward in both the source and target tasks assesses at the system level how good or bad the action applied to the different cells in the RAN infrastructure was, promoting the learning of optimal actions during the training process. These optimal actions are those that allow satisfying the tenant's SLAs with less assigned capacity and penalizing those that lead to not fulfilling the SLA or to assigning more capacity than needed to a tenant in a cell (i.e., overprovisioning). Therefore, a common definition of the reward, denoted as r, is considered for both the source and target tasks. However, in the source task, the reward will be based on the actions made over N cells, while in the target task, it will be based on N' cells.

3.2. Inter-Task Mapping Transfer Approach

The state and action definitions for the source and target tasks differ in the number of components. Specifically, the state and action definitions in the target task incorporate the components $\boldsymbol{s}^{\boldsymbol{\Delta N}}$ and $\boldsymbol{a}^{\boldsymbol{\Delta N}}(d)$, respectively, with respect to the source task definitions, while the rest of the components are common. Based on this, only a partial mapping can be defined between the states/actions in the source and target tasks, because $\boldsymbol{s}^{\boldsymbol{\Delta N}}$ and $\boldsymbol{a}^{\boldsymbol{\Delta N}}(d)$ in the target task cannot be mapped to any component in the source task. This introduces a special requirement when selecting the TRL approach, as the majority of TRL solutions require a full mapping between the state, action, and reward components in both tasks. Considering this requirement, the TRL approach selected here is based on the *transfer* via *inter-task mapping for policy search methods* (TVITM-PS) in [25], which proposes a methodology that supports partial mappings between states and actions when transferring the weights of the ANN that represents the policy in the source task to build the ANN for the policy in the target task.

The RL-based capacity sharing solution considered here in the source and target tasks relies on DQN-based agents [26] to learn the policy associated with each tenant. In particular, assuming that the state is $\boldsymbol{s}$, the policy that selects the action $\boldsymbol{a}$ for a tenant is given by:

$$\pi = \underset{\boldsymbol{a}}{argmax} Q(\boldsymbol{s}, \boldsymbol{a}, \theta) \tag{3}$$

where $Q(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\theta})$ is an ANN with weights $\boldsymbol{\theta}$. The ANN architecture consists of an input layer composed of a neuron for each component of the state, L hidden layers (where layer l

contains H_l neurons), and an output layer with as many neurons as the number of possible actions in the action space. Each output neuron provides the value of $Q(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\theta})$ for each possible action $\boldsymbol{a}$ in the action space. Considering that the neurons in the ANN are fully connected, $\boldsymbol{\theta}$ is defined as $\boldsymbol{\theta} = \{\theta_{l,o,d} \mid l = 1 \ldots L+2, \forall o, \forall d\}$, where $\theta_{l,o,d}$ corresponds to the weight between an origin neuron o of layer l and a destination neuron d of layer $l+1$.

Let us define $\boldsymbol{\theta^{(S)}}$ and $\boldsymbol{\theta^{(T)}}$ as the set of weights of the ANNs that represent the policy in the source and target tasks, and $\theta^{(S)}_{l,o,d}$ and $\theta^{(T)}_{l,o,d}$ as the individual weights in $\boldsymbol{\theta^{(S)}}$ and $\boldsymbol{\theta^{(T)}}$, respectively. Note that the ANNs in the source and target tasks present a different number of neurons in the input and output layers due to differences in the state and action definitions, but they have the same number of hidden layers and neurons in the hidden layers. This is illustrated in Figure 2, which shows the transfer operation between a source task with $N = 2$ cells and a target task with $N' = 3$ cells, indicating the ANN associated with both tasks. As cell $n = 3$ is introduced in the target task, the input layer incorporates new neurons corresponding to the $\boldsymbol{s^{\Delta N}}$ component in $\boldsymbol{s^{(T)}}$ (i.e., state $\boldsymbol{s_3}$ for cell $n = 3$ in this case). Regarding the output layer, new neurons are incorporated in the ANN for all possible actions $3^{N'}$.

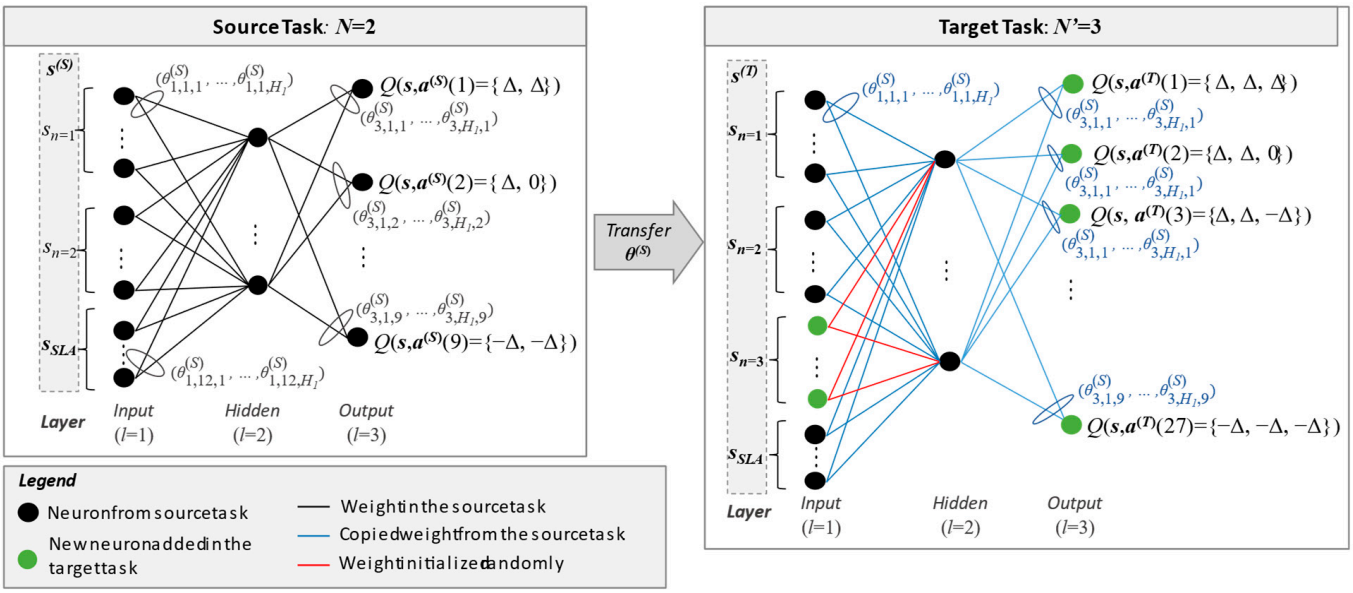

Figure 2. Illustrative example of the transfer of weights $\boldsymbol{\theta^{(S)}}$ when the transfer is performed for a source task with $N = 2$ cells to a target task with $N' = 3$.

Once the ANN in the target task is created, the weights $\boldsymbol{\theta^{(T)}}$ are initialized based on those in $\boldsymbol{\theta^{(S)}}$ of the source task as follows:

$$\theta^{(T)}_{l,o,d} = \begin{cases} \theta^{(S)}_{l,o,d} & \left(l = 1,\ o \in s^{(s)}\right) \text{ or } (2 <= l <= L) \\ \theta^{(S)}_{l,o,\varphi(d)} & l = L+1 \\ U(x_1, x_2) & \text{otherwise} \end{cases} \tag{4}$$

The first condition in (4) applies in cases where the weights $\theta^{(S)}_{l,o,d}$ are copied between the same pair of neurons in the source and target ANNs. This occurs in two cases. The first case is between the nodes in the hidden layers (i.e., $2 <= l <= L$), where all the weights are copied. The second case is when transferring the weights between the input neurons corresponding to the components in $\boldsymbol{s^{(T)}}$ that are common with $\boldsymbol{s^{(S)}}$ (i.e., $\boldsymbol{s^N}$ and $\boldsymbol{s^{SLA}}$) and the neurons of the first hidden layer. This is depicted in the example in Figure 2, where the $\theta^{(T)}_{l,o,d}$ between the state components for $\{\boldsymbol{s_n}\}_{n=1,2}$ are copied from $\boldsymbol{\theta^{(S)}}$. Regarding the rest of the neurons in the input layer, since they correspond to the new cells (i.e., corresponding

to $\boldsymbol{s^{\Delta N}}$), all weights are set randomly as $U(x_1, x_2)$, which denotes a uniformly distributed random variable in the range $[x_1, x_2]$. This is considered in the third condition in (4) and illustrated by the red lines associated with the state component $\mathbf{s_3}$ for the new cell $n = 3$ in Figure 2.

The second condition in (4) corresponds to the transfer of weights between the last hidden layer ($l = L + 1$) and the output layer ($l = L + 2$). In this case, considering that the d-th output neuron in the target task corresponds to the d-th action, i.e., $\boldsymbol{a^{(T)}}(d)$, and that the i-th output neuron in the source task corresponds to the i-th action, $\boldsymbol{a^{(S)}}(i)$, the transfer of weights between the last hidden layer and the output layer is held according to the action mapping function $\varphi(d)$ that maps each output neuron d in the target task to the output neuron i in the source task as follows:

$$\varphi(d) = i \mid \boldsymbol{a^{(S)}}(i) = \boldsymbol{a^{N}}(d) \tag{5}$$

which considers that the values of $\boldsymbol{a^{N}}(d)$ components in action $\boldsymbol{a^{(T)}}(d)$ of the target task are the same as those of $\boldsymbol{a^{(S)}}(i)$ in the source task. An example of this mapping function is shown in Figure 2, where the weights between the last hidden layer and the output neuron in the target ANN corresponding to action $\boldsymbol{a^{(T)}}(1) = \{\Delta, \Delta, \Delta\}$ are copied from those of source action $\boldsymbol{a^{(S)}}(1) = \{\Delta, \Delta\}$, since the values of $\boldsymbol{a^{N}}(1) = \{\Delta, \Delta\}$ of the target task are the same as $\boldsymbol{a^{(S)}}(1)$. Note that this mapping implies that all actions in the target task with common values of $\boldsymbol{a^{N}}(d)$ will be mapped to the same source task action, e.g., $\boldsymbol{a^{(T)}}(1) = \{\Delta, \Delta, \Delta\}$, $\boldsymbol{a^{(T)}}(1) = \{\Delta, \Delta, 0\}$ and $\boldsymbol{a^{(T)}}(1) = \{\Delta, \Delta, -\Delta\}$ in Figure 2 are mapped to $\boldsymbol{a^{(S)}}(1) = \{\Delta, \Delta\}$.

The fact that random weights are set for the input neurons related to new cells (third condition in (4)) and that different output neurons are mapped to the same output neuron in the source ANN (second condition in (4)) implies that once the weights in the ANN of the source task are transferred to the ANN in the target task, they cannot be directly used for the target task. Therefore, a training stage of the transferred ANN is required after transferring the weights to tune the weights of the ANN in the target task.

Figure 3 summarizes the overall process for obtaining the policy of a DQN agent in the target task following the proposed TRL approach. After obtaining the ANN from the source task, the ANN for the target task is generated considering the number of input neurons as the number of components in $\boldsymbol{s^{(T)}}$, the same number of hidden layers and hidden nodes as in the source ANN, and $3^{N'}$ neurons in the output layer. Then, the weights $\boldsymbol{\theta^{(S)}}$ are transferred to $\boldsymbol{\theta^{(T)}}$ according to (4). Next, an iterative process begins where the ANN in the target task is trained using the DQN algorithm to update and tune the weights $\boldsymbol{\theta^{(T)}}$. In each training step t, the DQN algorithm starts by collecting the state/action/reward experiences from the new RAN environment with N' cells, which are stored in the experience dataset of the DQN agent. Then, the algorithm updates the weights according to the mini-batch gradient descent of the loss. This is repeated until a training termination condition is met, obtaining policy π' as a result. The termination condition can be expressed in different ways, such as reaching a maximum number of training steps or achieving the convergence of a given metric. For further details on the training operation of the considered RL-based capacity sharing solution, the reader is referred to [11].

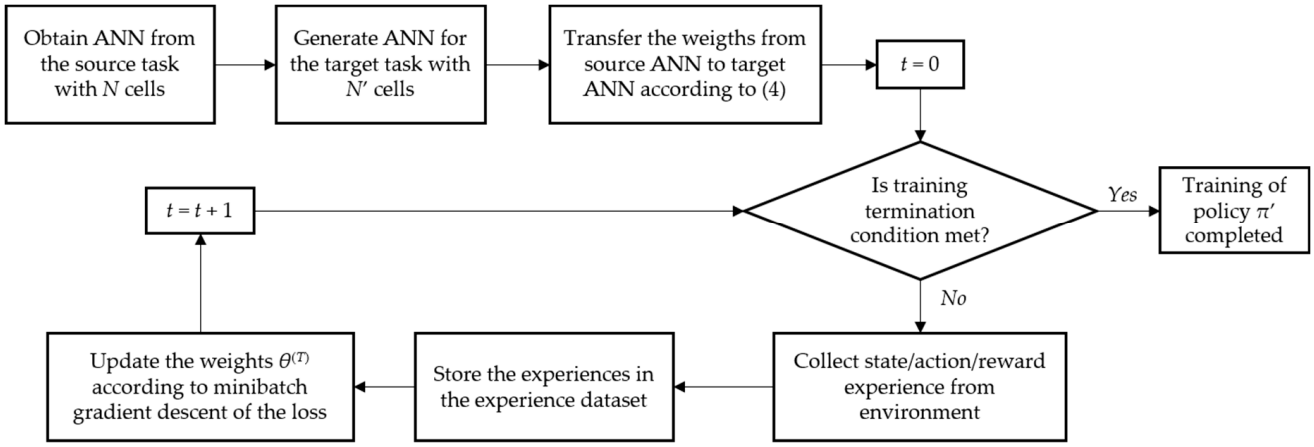

Figure 3. Learning process of the target policy with inter-task mapping.

4. Performance Evaluation

This section evaluates the proposed TRL approach by assessing its performance in a multi-cell and multi-tenant scenario, where a new cell is deployed to increase the scenario capacity. First, the considered scenario is presented in Section 4.1, and the training assessment methodology and Key Performance Indicators (KPIs) are defined in Section 4.2. Then, in Section 4.3, the assessment of the training performance of the TRL is conducted by first focusing on the considered scenario and then extending the study to a wider range of RAN deployments with different sizes. Finally, an analysis of the performance achieved by the trained policies according to the TRL approach during their inference is included in Section 4.4.

4.1. Considered Scenario

The assumed scenario comprises an NG-RAN infrastructure deployment that initially contains $N = 4$ cells, which serve the users of two different tenants, denoted as *Tenant* 1 and *Tenant* 2. The cells' configuration considered in the deployment is included in Table 1, along with the SLA parameters established for each tenant in the scenario. The SLA of the k-th tenant is defined in terms of the Scenario Aggregated Guaranteed Bit Rate, $SAGBR_k$, which is the bit rate to be provided across all cells to the tenant if requested, and the Maximum Cell Bit Rate, $MCBR_k$, which is the maximum bit rate that can be provided to the tenant in each cell. The DQN-MARL capacity sharing solution is used to distribute the capacity of the cells in the scenario among the two tenants, considering that the state of the k-th tenant in the n-th cell is given by $\boldsymbol{s_n} =< \rho_n(t), \rho_n^A(t), \sigma_n(t-1), \sigma_n^A(t-1), MCBR_{k,n}/c_n >$, where $\rho_n(t)$ is the fraction of resources occupied by the tenant, $\rho_n^A(t)$ is the fraction of available resources not used by any tenant, and $\sigma_n^A(t)$ is the available capacity share not assigned to any tenant, all referring to the n-th cell. Moreover, $\boldsymbol{s^{SLA}} = \{SAGBR_k/C, \sum_{k'=1,k'\neq k}^{K} SAGBR_{k'}/C\}$.

The traffic generated by a tenant in one cell is given in terms of the offered load, defined as the ratio between the bit rate required by the tenant and the cell capacity. The offered load of each tenant varies with time to capture the fluctuation of the traffic demands in different hours/days. Moreover, the offered load of a tenant is homogeneous in the scenario, so that all the cells experience the same offered load of that tenant. In the scenario under consideration, a situation of high offered load is experienced. To overcome this situation, the capacity of the scenario is increased by adding a new cell, which absorbs 25% of the offered load of each of the previous cells and is configured according to the cell parameters in Table 1. As a result, the new scenario with $N' = 5$ cells can provide more capacity to satisfy the offered loads of the tenants.

After the addition of the new cell, a re-training process of the DQN-MARL capacity sharing solution is required to enable the distribution of the capacity among tenants. This re-training is conducted in two modes. The first mode, denoted as *TRL mode*, considers that

the re-training is conducted by following the inter-task mapping TRL approach proposed in Section 3.2, considering $\Delta N = 1$ cell and using the trained policies for $N = 4$ cells. The second mode, denoted as *Non-TRL mode*, considers that no knowledge transfer is used, so the re-training is performed from scratch for the case of $N' = 5$ cells.

Table 1. Cells and SLA configuration.

Parameter		Value
Cell configuration		
PRB bandwidth (B_n)		360 kHz
Number of available PRBs in a cell (W_n)		65 PRBs
Average spectral efficiency (S_n)		5 b/s/Hz
Total cell capacity (c_n)		117 Mb/s
SLA configuration		
$SAGBR_k$	Tenant $k = 1$	60% of system capacity C
	Tenant $k = 2$	40% of system capacity C
$MCBR_{k,n}$	Tenant $k = 1$	80% of cell capacity c_n
	Tenant $k = 2$	

A Python implementation of the DQN-MARL capacity sharing solution and the TRL approach proposed in Section 3.2 has been carried out to obtain the results. Table 2 presents the DQN configuration parameters used for training the developed DQN agents for $N = 4$ cells and $N' = 5$ cells. These parameters apply to both *TRL* and *Non-TRL* re-training modes. In the *TRL* mode, ANN weights are initialized according to (4), while in the *Non-TRL* mode, weights are initialized randomly.

Table 2. DQN configuration parameters.

Parameter		Values	
Number of cells		$N = 4$	$N' = 5$
Initial training steps		1000	3000
ANN config.	Input layer (nodes)	22	27
	Fully connected layer	1 Layer (100 nodes)	
	Output layer(nodes)	81	243
Experience replay buffer maximum length (l)		10^7	
Mini-batch size (J)		256	
Learning rate (τ)		10^{-4}	
Discount factor(γ)		0.9	
ε value (ε-Greedy)		0.1	
Reward weights		$\varphi_1 = 0.5$, $\varphi_2 = 0.4$	
Time step duration (Δt)		3 min	
Action step (Δ)		0.03	

The training is performed through the interaction of the developed DQN agents with a simulated network environment, also developed in Python. This environment is fed with data from a training dataset composed of synthetically generated offered loads with different patterns for the two tenants, each with a one-day duration. The considered patterns cover a wide range of offered load values for both tenants.

4.2. Training Assessment Methodology and KPIs

The training performance for both *TRL* and *Non-TRL* modes is assessed by evaluating the policies available every 10^3 steps during the training. Each evaluation consists of applying the policy during a simulation of 40 days. For each day, there is a different offered load pattern for each tenant, thus embracing a rich set of offered load conditions for both tenants. This allows evaluating the trained policies across a large range of states.

The following KPIs are defined to quantify the performance of the model:

- Average aggregated reward per evaluation (R): It is computed as the average of the aggregated reward of the two tenants throughout one evaluation.
- Standard deviation ($std(m, W)$): The standard deviation of R at the m-th training step measured over the window of the last W training steps.
- Training duration: Number of training steps until the convergence criterion is achieved. This criterion considers that convergence is achieved in the training step m that fulfills $std(m, W) < std_{th}$, where std_{th} is a threshold.

4.3. Training Performance

Figure 4 shows the evolution of the average aggregated reward per evaluation, R, during the training of the policies for $N' = 5$ cells with the *TRL* and *Non-TRL* modes. For both cases, the value of R increases abruptly at the beginning of the training and then stabilizes to a value of around 1.78. However, the value of R increases faster with the *TRL* mode than with the *Non-TRL* mode. Moreover, while R exhibits high fluctuations from the beginning until approximately 200×10^3 training steps with the *Non-TRL* mode, these fluctuations are much smaller with the *TRL* mode throughout the training. These results indicate that the proposed TRL approach allows for achieving a higher and more stable reward sooner during the training, thus reducing the training duration. To further assess the convergence of the policies obtained for *TRL* and *Non-TRL* modes, Figure 5 shows the standard deviation of R, $std(m,W)$, when considering a window of $W = 30 \times 10^3$ training steps. In contrast to the evolution of R, the value of $std(m,W)$ decreases abruptly at the beginning of the training for both modes. After this initial decrease, the value of $std(m,W)$ decreases slowly until stabilizing. Higher standard deviation values are obtained with the *Non-TRL* mode due to the larger reward fluctuations that were observed in Figure 5. Accordingly, the standard deviation values for the *TRL* mode decrease and stabilize much sooner, evidencing that the proposed TRL approach accelerates the training process. In fact, considering that convergence is achieved when the standard deviation $std(m,W)$ is lower than $std_{th} = 0.01$, the training duration for the *Non-TRL* mode is 265×10^3 training steps, while for the *TRL* mode it is 123×10^3 training steps. This means that the use of the proposed TRL approach reduces the training duration by 54%.

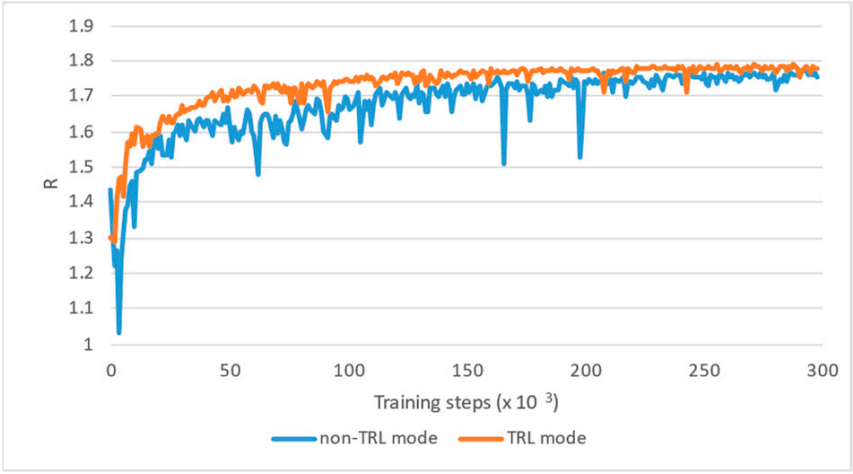

Figure 4. Average aggregated reward (R) during training.

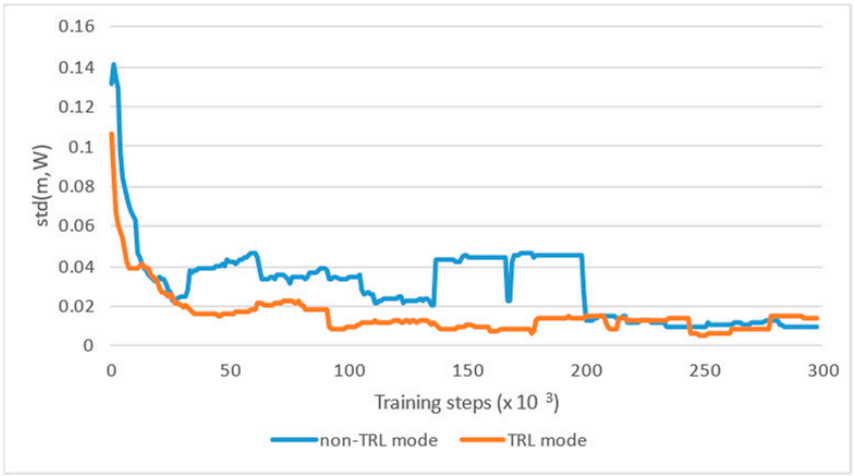

Figure 5. Standard deviation of R.

To evaluate the proposed TRL approach in a wider range of RAN deployments, the analysis of the training behavior has been extended to different RAN deployments that initially have $N = 1...7$ cells and are extended to $N' = 2...8$ cells, respectively. The cells' configuration and SLA parameters in the different deployments are configured according to Table 1. For each deployment, results have been obtained when training the DQN-MARL capacity sharing solution according to the *TRL* and *Non-TRL* modes. The DQN configuration parameters are summarized in Table 3 for each number of cells, while the rest of the parameters not mentioned in this table are the same as those in Table 2. Note that the number of input and output layers for each number of cells in Table 3 equals the number of components of the state and the number of possible actions, respectively, which depend on the number of cells. Also, the number of initial training steps is increased with the number of cells to better explore the larger number of possible actions and states.

Table 3. DQN configuration for different deployments.

Parameter		**Values**							
Number of cells		1	2	3	4	5	6	7	8
Initial training steps		50	100	300	1×10^3	3×10^3	8×10^3	2×10^4	6.5×10^3
ANN config.	Input layer (nodes)	7	12	17	22	27	32	37	42
	Fully connected layer	1 Layer (100 nodes)							
	Output layer (nodes)	3	9	27	81	243	729	2187	6561

The training performance for obtaining the policies for $N' = 2...8$ cells has been assessed according to the methodology described in Section 4.2 with a window $W = 30 \times 10^3$ training steps and a standard deviation threshold $std_{th} = 0.01$, as previously considered. Table 4 includes the training duration in time steps for the different deployments in both *TRL* and *Non-TRL* modes. For a given mode, the results show that the training durations for deployments with a low number of cells are similar (i.e., for $N' = 2$ and $N' = 3$, the duration is around 90×10^3 training steps with the TRL mode and around 170×10^3 training steps with the *Non-TRL* mode). However, for a larger number of cells in the RAN deployment, the training duration increases, since the number of state components and possible actions increases substantially, as does the size of the ANN. Table 4 shows that the increase in the training duration is generally more pronounced when the TRL approach is not used. For instance, the training duration between $N' = 7$ cells and $N' = 8$ cells increases by approximately 300% when TRL is not used, while this increase is 160% when TRL is used.

Table 4. Training duration for the different deployments.

Number of Cells After the New Cell Deployment (N')	Training Duration (Training Steps)		Training Duration Reduction
	Non-TRL Mode	*TRL Mode*	
2	178×10^3	88×10^3	51%
3	169×10^3	92×10^3	46%
4	251×10^3	124×10^3	51%
5	265×10^3	123×10^3	54%
6	950×10^3	195×10^3	79%
7	2377×10^3	788×10^3	67%
8	9700×10^3	2058×10^3	79%

Furthermore, the results in Table 4 indicate that shorter training durations are required in the case of using the TRL approach. To provide further insight, the percentage of training duration reduction when using the *TRL* mode with respect to the *Non-TRL* mode has also been included in the last column of Table 4. For $N' \leq 5$ cells, reductions of around 50% are obtained, while larger reductions are achieved for $N' \geq 6$. The reason is that increasing the number of cells leads to an increase in the number of states and actions, and thus, the ANN size and the complexity of the training. These results highlight that the gains of the proposed TRL approach are higher when it is applied in RAN deployments with a larger number of cells.

4.4. Performance of Trained Policies

To assess the performance of the trained policies, the offered load evolution during one day depicted in Figure 6 has been considered in the scenario in Section 4.1 with $N = 4$ initial cells, which is then extended to $N' = 5$ cells. For both scenarios, the figure includes the evolution of the offered load per tenant in one cell, denoted as O_1 and O_2 for Tenant 1 and Tenant 2, respectively, as well as the aggregated offered load of both tenants in the cell, denoted as O. The scenarios with $N = 4$ cells and $N' = 5$ cells are represented by straight and dotted lines, respectively. The considered offered loads per tenant show a complementary behavior, i.e., while the offered load values of Tenant 1 are larger during the middle of the day, larger values of Tenant 2 are experienced during the night. It is worth highlighting that for $N = 4$ cells, the aggregated offered load of both tenants is close to or even greater than 1 reflecting that there is not enough cell capacity to serve the required bit rate of the two tenants. Instead, for the new scenario with $N' = 5$ cells, the aggregate offered load is at most 0.9, meaning that there is no overload.

For the case of the $N' = 5$ cells, the policies obtained after convergence as a result of the *TRL* and *Non-TRL* modes have been applied to the offered loads of both tenants. Figure 7 shows the assigned capacity to each tenant in the cell, denoted as A_1 and A_2 for tenants 1 and 2, respectively, by the policies obtained with each re-training mode against the offered loads of each of the tenants in the cell. The results show that the policies obtained with both re-training modes execute a similar capacity assignment in which the offered loads are satisfied during most of the day without overprovisioning. In fact, the SLA satisfaction ratio, computed as the average ratio between the throughput of a tenant and the minimum of $SAGBR_k$ and the tenant's offered load, is 96% for the policy trained with the *Non-TRL* mode and 95% with the *TRL* mode, reflecting very similar performance. These results reveal that the behavior of the policies trained with the proposed TRL approach is similar to the one obtained without TRL but with the benefit of a much smaller training duration.

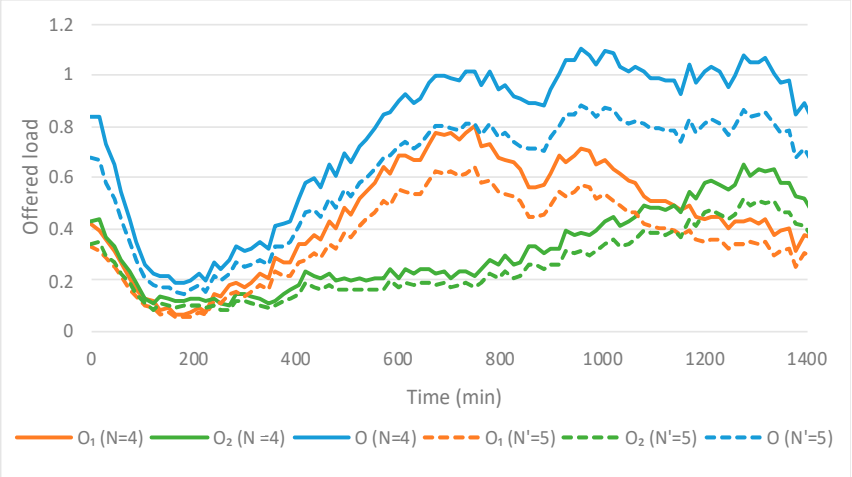

Figure 6. Aggregated offered load per tenant in one cell for $N = 4$ and $N' = 5$.

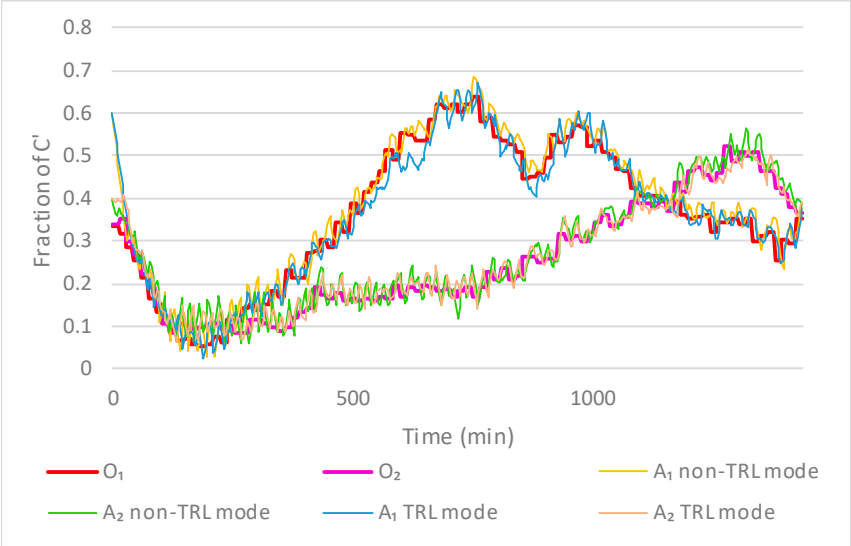

Figure 7. Offered load vs. assigned capacity to tenants 1 and 2 for both re-training modes.

5. Implementation Considerations

The DQN-MARL capacity sharing solution can be implemented into the O-RAN architecture, which is expected to be widely adopted in 5G and beyond deployments. The O-RAN architecture is composed of a set of open interfaces for the realization of a virtualized RAN with disaggregated functionalities and embedded AI [27]. Figure 8 shows the integration of the DQN-MARL capacity sharing solution into the O-RAN architecture, illustrating the involved components of the O-RAN architecture in blue and those related to capacity sharing in orange. The solution is devised as an *rApp* running on the non-Real Time RAN Intelligent Controller (non-RT RIC) functionality of the Service Management and Orchestration (SMO). The SMO is responsible for managing the different O-RAN functionalities; the non-RT RIC functionality enables closed-loop control of RAN and SMO operations with timescales longer than 1 s; and *rApps* are third-party applications that are executed on the non-RT RIC to provide value-added services related to intelligent RAN optimization and operation. Note that approaching capacity sharing solutions as *rApps* has been identified as a use case by the O-RAN use case specification for the non-RT RIC in [28].

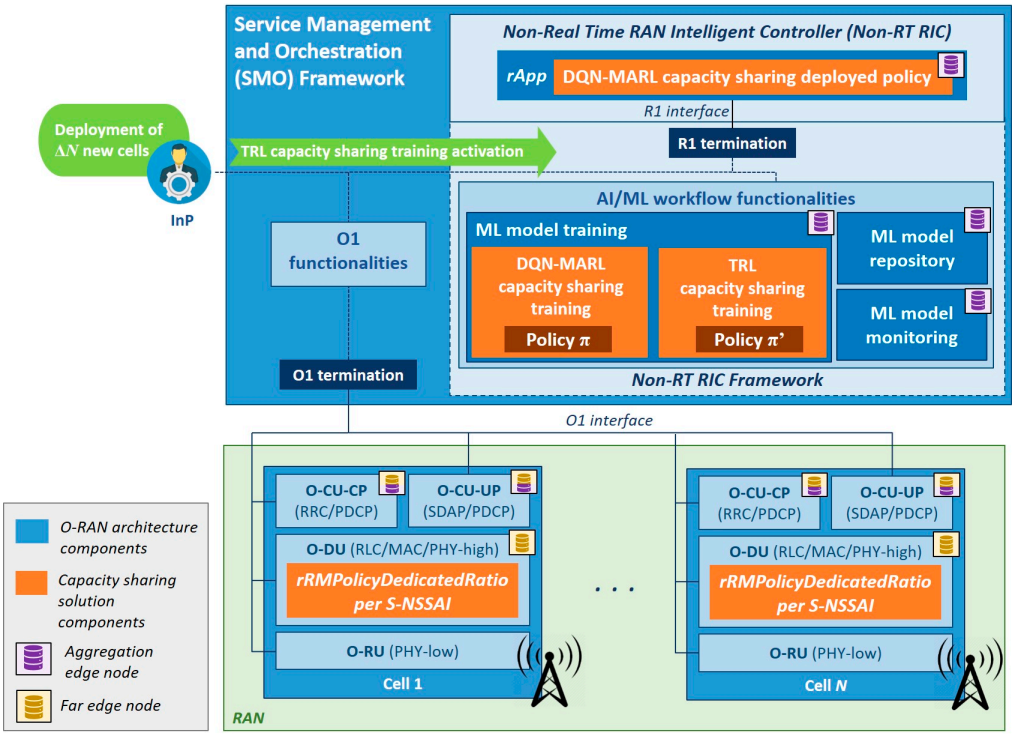

Figure 8. O-RAN-based system model for capacity sharing and deployment at the edge.

The *rApp* includes the learned DQN-MARL capacity sharing policy for tuning the capacity shares $\sigma_n(t)$ in the different cells of the RAN. Following the O-RAN architecture, the different cell functionalities are split among the O-RAN Radio Unit (O-RU), which is in charge of the lower part of the PHY layer functionality; the O-RAN Distributed Unit (O-DU), responsible for high PHY layer processing, Medium Access Control (MAC), and Radio Link Control (RLC); the O-RAN Central Unit–Control Plane (O-CU-CP), hosting the upper layers of the control plane (i.e., Radio Resource Control (RRC), and control plane of Packet Data Convergence Protocol (PDCP)); and the O-RAN Central Unit –User Plane (O-CU-UP), hosting the upper layers of the user plane (i.e., Service Data Adaptation Protocol (SDAP) and user plane of PDCP). All these cell functionalities are managed by the SMO through the O1 interface, providing diverse management services (e.g., provisioning management services, performance assurance services, etc.) [29]. Considering this functional split, the capacity share $\sigma_n(t)$ of a tenant in a cell is configured in the O-DU function through the O1 interface using the parameter *rRMPolicyDedicatedRatio*, which belongs to the 3GPP Network Resource Model (NRM) for characterizing RRM Policies in [30]. This parameter is provided per slice, hence for each tenant, identified by means of the Single Network Slice Selection Assistance Information (S-NSSAI). To determine the capacity share $\sigma_n(t)$, the policy of the DQN-MARL *rApp* makes use of performance measurements gathered by the O-DU and O-CU functions and transferred to the SMO through the O1 interface. For more details on the *rApp* implementation, the reader is referred to our work in [31].

Moreover, the *rApp* can interact with different services in the non-RT RIC and the SMO frameworks through the R1 interface [32]. These services include interactions of the *rApp* through the O1 interface and also with functionalities related to the O-RAN AI/ML workflow services [33]: the *ML model training* functionality, which allows the training of the DQN-MARL capacity sharing solution; the *ML model repository* functionality, which allows storing trained ML model versions as a result of re-training; and the *ML model monitoring* functionality, enabling runtime monitoring of the performance of the deployed policy in the *rApp*.

The proposed TRL approach for capacity sharing in this paper is included in the *ML model training* in Figure 8. Once the InP decides to deploy new cells in the RAN, it can

activate the TRL capacity sharing training functionality, specifying the value of ΔN. This functionality will use the knowledge gained from a previously learned policy π, which can be obtained from the *ML model repository*, to learn the new policy π'. When the training of π' is completed, this policy can be deployed in the *rApp* through the R1 interface, replacing the old policy π.

The disaggregated nature of the O-RAN architecture offers the flexibility to deploy the O-RAN functionalities at different network locations, including far edge nodes (i.e., edge nodes co-located with cells and closer to users), aggregation edge nodes (i.e., edge nodes at an aggregation point of multiple far edge nodes), or the cloud, as described in [34]. The choice of deployment location depends on the requirements of the functionalities in terms of their operating times and geographical scope (i.e., the area where they exert influence or need to gather data). It may also consider the minimization of network data transfer among the transport network, core network, or the cloud to reduce bandwidth requirements and associated costs. Figure 8 also indicates a proposal for the placement of the different components among far edge nodes and aggregation edge nodes, colored in yellow and purple, respectively. Regarding cell-related O-RAN functionalities, the O-DU is deployed in a far edge node, while the O-CU-UP and O-CU-CP can be deployed both at far edge and aggregation edge nodes. Considering that the inference of the *rApp* containing the DQN-MARL capacity sharing policy does not have stringent real-time requirements, it can be deployed at an aggregation edge node that is close to the different cells managed by the solution and has access to them. The *ML model training*, *ML model repository*, and *ML model monitoring* functionalities can be located at the same node if it has enough computing power to support them.

6. Conclusions

This paper has presented a Transfer Reinforcement Learning (TRL) approach for accelerating the re-training process of Reinforcement Learning (RL)-based capacity sharing solutions for multi-cell scenarios when new cells are deployed in the Radio Access Network (RAN). The proposed approach allows transferring the weights of the existing capacity sharing policy for a given number of cells to build the initial artificial neural network of the policy to be used with a higher number of cells. This allows accelerating the re-training process to determine this policy.

The behavior of the proposed TRL approach has been assessed in a multi-cell scenario where a new cell is deployed by comparing the training behavior and the performance of the trained policies with and without TRL. Results have shown that: (i) The training duration is significantly reduced when using the TRL approach for re-training, achieving convergence in around 50–80% less time in the considered scenarios. (ii) The performance of the trained policies with the TRL approach is similar to the performance when training the policies from scratch, reaching SLA satisfaction ratios of 96% and 95%, respectively. Overall, the results presented here reflect the capability of the proposed TRL solution to accelerate the re-training process of the policies when adding new cells in the scenario, obtaining good performance with reduced training durations. Finally, some considerations on the implementation of the solution have been discussed, including the integration of the solution into the O-RAN architecture and the placement of the different functionalities at the edge nodes.

Author Contributions: Conceptualization, I.V., J.P.-R. and O.S.; methodology, I.V., J.P.-R. and O.S.; software, I.V.; validation, I.V., J.P.-R. and O.S.; formal analysis, I.V., J.P.-R. and O.S.; investigation, I.V., J.P.-R. and O.S.; data curation, I.V.; writing—original draft preparation, I.V.; writing—review and editing, I.V., J.P.-R. and O.S.; visualization, I.V., J.P.-R. and O.S.; supervision, J.P.-R. and O.S.; project administration, J.P.-R. and O.S.; funding acquisition, J.P.-R. and O.S. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been partially funded by the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation programme under

Grant Agreement No. 101096034 (VERGE project). Views and opinions expressed are, however, those of the authors only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them. This research has also been partly funded by the Spanish Ministry of Economic Affairs and Digital Transformation and the European Union—NextGenerationEU under projects OPTIMAIX_OaaS (Ref. TSI-063000-2021-34) and OPTIMAIX_NDT (Ref. TSI-063000-2021-35). The work of Irene Vilà has been funded by the European Union-NextGenerationEU, the Spanish Ministry of Universities, and the Plan for Recovery, Transformation, and Resilience, through the call for Margarita Salas Grants of the Universitat Politècnica de Catalunya (ref. 2022UPC-MSC-94079).

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Yzar, A.; Dogan-Tusha, S.; Arslan, H. 6G Vision: An Ultra-flexible Perspective. *ITU J. Future Evol. Technol.* **2020**, *1*, 121–140.
2. Li, R.; Zhao, Z.; Sun, Q.; Chih-Lin, I.; Yang, C.; Chen, X.; Zhao, M.; Zhang, H. Deep Reinforcement Learning for Resource Management in Network Slicing. *IEEE Access* **2018**, *6*, 74429–74441. [CrossRef]
3. Qi, C.; Hua, Y.; Li, R.; Zhao, Z.; Zhang, H. Deep Reinforcement Learning with Discrete Normalized Advantage Functions for Resource Management in Network Slicing. *IEEE Commun. Lett.* **2019**, *23*, 1337–1341. [CrossRef]
4. Hua, Y.; Li, R.; Zhao, Z.; Chen, X.; Zhang, H. GAN-Powered Deep Distributional Reinforcement Learning for Resource Management in Network Slicing. *IEEE J. Sel. Areas Commun.* **2020**, *38*, 334–349. [CrossRef]
5. Sun, G.; Gebrekidan, Z.T.; Boateng, G.O.; Ayepah-Mensah, D.; Jiang, W. Dynamic Reservation and Deep Reinforcement Learning Based Autonomous Resource Slicing for Virtualized Radio Access Networks. *IEEE Access* **2019**, *7*, 45758–45772. [CrossRef]
6. Sun, G.; Xiong, K.; Boateng, G.O.; Ayepah-Mensah, D.; Liu, G.; Jiang, W. Autonomous Resource Provisioning and Resource Customization for Mixed Traffics in Virtualized Radio Access Network. *IEEE Syst. J.* **2019**, *13*, 2454–2465. [CrossRef]
7. Li, T.; Zhu, X.; Liu, X. An End-to-End Network Slicing Algorithm Based on Deep Q-Learning for 5G Network. *IEEE Access* **2020**, *8*, 122229–122240. [CrossRef]
8. Sun, G.; Boateng, G.O.; Ayepah-Mensah, D.; Liu, G.; Wei, J. Autonomous Resource Slicing for Virtualized Vehicular Networks with D2D Communications Based on Deep Reinforcement Learning. *IEEE Syst. J.* **2020**, *14*, 4694–4705. [CrossRef]
9. Mei, J.; Wang, X.; Zheng, K.; Bondreau, G.; Bin, A.; Abou-Zeid, H. Intelligent Radio Access Network Slicing for Service Provisioning in 6G: A Hierarchical Deep Reinforcement Learning Approach. *IEEE Trans. Commun.* **2021**, *69*, 6063–6078. [CrossRef]
10. Abiko, Y.; Saito, T.; Ikeda, D.; Ohta, K.; Mizuno, T.; Mineno, H. Flexible Resource Block Allocation to Multiple Slices for Radio Access Network Slicing Using Deep Reinforcement Learning. *IEEE Access* **2020**, *8*, 68183–68198. [CrossRef]
11. Vilà, I.; Pérez-Romero, J.; Sallent, O.; Umbert, A. A Multi-Agent Reinforcement Learning Approach for Capacity Sharing in Multi-tenant Scenarios. *IEEE Trans. Veh. Technol.* **2021**, *70*, 9450–9465. [CrossRef]
12. Vilà, I. Contribution to the Modelling and Evaluation of Radio Network Slicing Solutions in 5G. Ph.D. Thesis, Universitat Politècnica de Catalunya, Departament de Teoria del Senyal i Comunicacions, Barcelona, Spain, 6 April 2022.
13. Zhou, L.; Leng, S.; Wang, Q.; Liu, Q. Integrated Sensing and Communication in UAV Swarms for Cooperative Multiple Targets Tracking. *IEEE Trans. Mob. Comput.* **2023**, *22*, 6526–6542. [CrossRef]
14. Zhou, L.; Leng, S.; Wang, Q. A Federated Digital Twin Framework for UAVs-Based Mobile Scenarios. *IEEE Trans. Mob. Comput.* **2024**, *23*, 7377–7393. [CrossRef]
15. Zhao, R.; Li, Y.; Fan, Y.; Gao, F.; Tsukada, M.; Gao, Z. A Survey on Recent Advancements in Autonomous Driving Using Deep Reinforcement Learning: Applications, Challenges, and Solutions. *IEEE Trans. Intell. Transp. Syst.* **2024**. *early access*. [CrossRef]
16. O-RAN-WG2. *AI/ML Workflow Description and Requirements v01.03*; Technical Report; O-RAN Alliance: Alfter, Germany, October 2021.
17. Pan, S.J.; Yang, Q. A Survey on Transfer Learning. *IEEE Trans. Knowl. Data Eng.* **2010**, *22*, 1345–1359. [CrossRef]
18. Mei, J.; Wang, X.; Zheng, K. An intelligent self-sustained RAN slicing framework for diverse service provisioning in 5G-beyond and 6G networks. *Intell. Converg. Netw.* **2020**, *1*, 281–294. [CrossRef]
19. Gautam, N.; Lieto, A.; Malanchini, I.; Liao, Q. Leveraging Transfer Learning for Production-Aware Slicing in Industrial Networks. In Proceedings of the 2023 IEEE 97th Vehicular Technology Conference (VTC2023-Spring), Florence, Italy, 20–23 June 2023.
20. Nagib, A.M.; Abou-Zeid, H.; Hassanein, H.S. Transfer Learning-Based Accelerated Deep Reinforcement Learning for 5G RAN Slicing. In Proceedings of the 2021 IEEE 46th Conference on Local Computer Networks (LCN), Edmonton, AB, Canada, 4–7 October 2021.
21. Nagib, A.M.; Abou-Zeid, H.; Hassanein, H.S. Safe and Accelerated Deep Reinforcement Learning-Based O-RAN Slicing: A Hybrid Transfer Learning Approach. *IEEE J. Sel. Areas Commun.* **2024**, *42*, 310–325. [CrossRef]
22. Hu, T.; Liao, Q.; Liu, Q.; Carle, G. Network Slicing via Transfer Learning aided Distributed Deep Reinforcement Learning. In Proceedings of the 2022 IEEE Global Communications Conference, Rio de Janeiro, Brazil, 4–8 December 2022.

23. Hu, T.; Liao, Q.; Liu, Q.; Carle, G. Inter-Cell Network Slicing with Transfer Learning Empowered Multi-Agent Deep Reinforcement Learning. *IEEE Open J. Commun. Soc.* **2023**, *4*, 1141–1155. [CrossRef]
24. Zhu, Z.; Lin, K.; Jain, A.K.; Zhou, J. Transfer Learning in Deep Reinforcement Learning: A Survey. *IEEE Trans. Pattern Anal. Mach. Intell.* **2023**, *45*, 13344–13362. [CrossRef]
25. Taylor, M.E.; Whiteson, S.; Stone, P. Transfer via Inter-Task Mappings in Policy Search Reinforcement Learning. In Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems, Honolulu, HI, USA, 14–18 May 2007.
26. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef]
27. O-RAN.WG1. *O-RAN Architecture Description Version 6.00*; O-RAN Alliance, Working Group 1, Technical Specification; O-RAN Alliance: Alfter, Germany, November 2021.
28. O-RAN.WG2. *O-RAN Non-RT RIC & A1 Interface: Use Cases and Requirements Version 5.00*; O-RAN Alliance, Working Group 2, Technical Specification; O-RAN Alliance: Alfter, Germany, November 2021.
29. O-RAN.WG10. *O-RAN Operations and Maintenance Interface Specification v05.00*; O-RAN Alliance, Working Group 10, Technical Specification; O-RAN Alliance: Alfter, Germany, August 2020.
30. 3GPP. *Management and Orchestration; 5G Network Resource Model (NRM) (Release 16)*; 3GPP TS 28.541 v16.0.0; 3GPP: Sophia Antipolis, France, March 2019.
31. Vilà, I.; Sallent, O.; Pérez-Romero, J. On the Implementation of a Reinforcement Learning-based Capacity Sharing Algorithm in O-RAN. In Proceedings of the 2022 IEEE Globecom Workshops (GC Wkshps), Rio de Janeiro, Brazil, 4–8 December 2022.
32. O-RAN.WG2. *Non-RT RIC Architecture v02.01*; O-RAN Alliance, Working Group 2, Techical Specification; O-RAN Alliance: Alfter, Germany, October 2022.
33. O-RAN.WG2. *Non-RT RIC Functional Architecture v01.01*; O-RAN Alliance, Working Group 2, Technical Report; O-RAN Alliance: Alfter, Germany, June 2021.
34. Polese, M.; Bonati, L.; D'Oro, S.; Basagni, S.; Melodia, T. Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges. *IEEE Commun. Surv. Tutor.* **2023**, *25*, 1376–1411. [CrossRef]