On the Training of Reinforcement Learning-based Algorithms in 5G and Beyond Radio Access Networks

I. Vilà, J. Pérez-Romero, O. Sallent
Dept. of Signal Theory and Communications, Universitat Politècnica de Catalunya (UPC)
Barcelona, Spain
[irene.vila.munoz, jordi.perez-romero]@upc.edu, sallent@tsc.upc.edu

Abstract—**Reinforcement Learning (RL)-based algorithmic solutions have been profusely proposed in recent years for addressing multiple problems in the Radio Access Network (RAN). However, how RL algorithms have to be trained for a successful exploitation has not received sufficient attention. To address this limitation, which is particularly relevant given the peculiarities of wireless communications, this paper proposes a functional framework for training RL strategies in the RAN. The framework is aligned with the O-RAN Alliance machine learning workflow and introduces specific functionalities for RL, such as the way of specifying the training datasets, the mechanisms to monitor the performance of the trained policies during inference in the real network, and the capability to conduct a retraining if necessary. The proposed framework is illustrated with a relevant use case in 5G, namely RAN slicing, by considering a Deep Q-Network algorithm for capacity sharing. Finally, insights on other possible applicability examples of the proposed framework are provided.**

Keywords—Radio Access Network, Network slicing, Reinforcement Learning, Training.

I. Introduction

The advent of 5G promises to deliver a wide range of new services (e.g., self-driving car, virtual reality, eHealth, etc.) with heterogeneous requirements (e.g., high data rates, reliability, low latencies, massive connection densities, etc.). To meet the demands of all these services and to provide a high level of flexibility, the 5G New Radio (NR) technology comes with a number of substantially new features in relation to previous Long Term Evolution (LTE) standard. These are included both from an architectural perspective, e.g. through the support of network slicing or the split of the gNB network function, and from a radio access technology perspective, e.g. through the support of higher frequency ranges, flexible numerologies or enhanced multi-antenna transmission. In turn, increasing virtualization and embedding data-driven intelligence into the Radio Access Network (RAN) are two of the major paradigm changes associated with 5G and beyond systems. In this way, the RAN industry is re-shaping towards more intelligent, autonomous, open, virtualised and fully interoperable mobile networks.

A key driving force in this transformation towards more autonomous mobile networks comes from the exploitation of Artificial Intelligence (AI) and, more specifically, Machine Learning (ML) techniques. These are permeating almost all layers of the next generation RAN architecture and the associated operations and support systems (OSS) for network management. There is a vast literature in the application of AI/ML in the RAN, which is profusely captured in survey papers (e.g., [1]-[4]) or tutorial-like papers discussing the challenges, opportunities and applicability areas (e.g., [5]-[7]). In turn, the interest of the industry in this area is also reflected by a number of standardisation initiatives started in the last years for introducing AI/ML techniques in wireless networks through the specification of supporting architectures or AI/ML workflows. This is the case of the Open RAN (O-RAN) Alliance [8], the ITU-T Y.3172 recommendation [9] or the ETSI Industry Specification Groups (ISG) on Zero-touch network and Service Management (ZSM) [10] and on Experiential Networked Intelligence (ENI) [11]. In turn, Third Generation Partnership Project (3GPP) has also approved very recently a new Rel. 18 study item on AI/ML for the New Radio (NR) air interface [12] that will explore the benefits of augmenting the air interface with features enabling improved support of AI/ML-based algorithms.

Reinforcement Learning (RL) techniques [13], a subset within the ML domain, are of special interest in the RAN because they are conceived for solving decision-making problems in an optimal way. As such, they have been proposed in the literature for multiple functionalities in the RAN, as presented in [1]-[7] and references therein. Most of these works focus on the formulation of RL-based algorithmic solutions and the assessment of their performance. Nevertheless, the current literature lacks from a consolidated view on how an RL algorithm has to be trained in order that it can be successfully and safely deployed in a RAN. Not sufficient attention has been paid in making explicit distinction between the training environment and the execution of the algorithm in the real network or in studying in detail practical aspects such as the time needed to achieve convergence, the impact of the trial and error process or the capability of the algorithm to react in front of changes in the environment. To the authors' best knowledge, only a few works have considered to some extent such practical issues in the evaluation of RL, such as [14], which proposes deep RL for RAN slicing and distinguishes between an initial training of the algorithm conducted on a training scenario and a subsequent evaluation of the trained algorithm in an evaluation scenario, [15], which analyses the robustness of a Q-learning algorithm for channel selection to relearn proper solutions when changes in the environment occur, or [16], which trains a deep RL-based mobility load balancing (MLB) algorithm in an offline network evaluation system based on historical performance records. The research interest in the area of model training for the RAN is also sustained by the abovementioned new 3GPP Rel. 18 study item [12], which will pay special attention to considering adequate model training strategies and to defining common assumptions for training dataset construction.

The abovementioned aspects constitute the research gap addressed by this paper, which contributes to fill it by proposing a functional model for training RL algorithms operating in the RAN. The proposed model is aligned with the vision on ML functionality of O-RAN [8]. The model distinguishes between the training stage, executed on a specific training environment, and the inference stage, executed on the real network, and proposes specific functions for controlling the training process of RL models. These are in charge of specifying the configurations of the training environment through datasets, monitoring the training process and the operation in the real environment to detect situations when the model needs to be retrained. The functional model is illustrated with an applicability example of a deep RL algorithm for capacity sharing in RAN slicing.

The rest of the paper is organized as follows. Section II presents some initial background on RL for better understanding of their capabilities. This is followed by a discussion on the applicability domains of RL in the RAN. Then, Section III highlights the main challenges for training an RL algorithm in the RAN and presents the proposed functional model. Section IV presents a specific applicability example of the proposed model illustrated with some results. Finally, conclusions and future work are summarized in Section V.

II. Preliminaries on Reinforcement Learning and its Applicability in the RAN

A. *Background on RL*

Machine learning techniques are usually subdivided into three main categories. The first one is supervised learning that consists in learning from training examples provided by an external supervisor in the form of pairs of inputs and desired outputs, so it is not valid for interactive problems where the desired behaviour is not known. The second category is the unsupervised learning, consisting in learning to represent particular input patterns (e.g. independent samples of an underlying unknown probability distribution) in a way that reflects their statistical structure. The third category is reinforcement learning, which consists in learning a behavioral model through the dynamic interaction with an environment. This interaction is based on the execution of actions that provide as a result a reinforcement signal in the form of a reward that encodes the success of the action outcome. In this way, the learner, referred to as the *agent*, seeks to learn the actions that maximize the reward. RL provides a mathematical formalism for learning-based control that allows acquiring near-optimal behavioral skills and, for this reason, RL methods have applicability in many decision making problems.

The operation of an RL system relies on the following elements [13]: (i) *State*: It conveys to the agent some sense of how the environment is at a certain time. (ii) *Policy*: It defines the agent behavior by mapping states of the environment to actions to be taken when being in those states. (iii) *Reward signal*: It is provided by the environment to the agent defining what are the good and bad actions, constituting the primary basis for modifying the policy with the objective of maximizing the total reward accumulated by the agent in the future, referred to as *return*. (iv) *Value functions (state-value function and action-value function)*: The state-value function of a state under a given policy is the return that the agent can expect to obtain over the future, starting from that state and following the policy. Similarly, the action-value function for a state-action pair and under a policy is defined as the expected return starting from the state, taking the action and thereafter following the policy. In this way, these functions capture what is good in the long run, in contrast to the reward, which defines what is good in an immediate sense.

With these elements, the target of RL methods is to find an optimal policy, i.e. a policy whose value functions are the highest ones among any other policy for all possible states. This search is done by means of a *policy iteration* process in which a policy is evaluated to estimate the value functions. Then, this estimation is used to improve the policy, and the process is repeated until reaching a termination condition. This policy iteration is the basis for the *training* of RL methods, understood as the process for acquiring the optimal policy of the RL agent.

RL methods can be either model-based or model-free depending on whether they use or not a model of the environment for finding the optimal policy. In model-based methods, such as Dynamic Programming (DP), a perfect (mathematical) model of the environment is used. Instead, model-free methods do not assume complete knowledge of the environment, but instead, they learn from experiences, i.e., sample sequences of states, actions, and rewards obtained from actual or simulated interaction with an environment. Depending on how the policy iteration is performed, model-free methods can in turn be categorized as Monte Carlo methods or Temporal-Difference (TD) methods. Monte Carlo methods are defined for episodic tasks (i.e., in which the agent-environment interaction ends in a termination state), so that policies are only updated at the end of an episode based on the obtained return. Instead, in TD methods the policy updates are done on a time-step basis by using the observed reward, so they are valid for both episodic and continuing tasks (i.e., in which the agent-environment interaction does not break naturally into episodes, but goes on continually without limit). Examples of TD methods are Q-learning or SARSA. TD methods are the most widely used RL methods, mainly due to the fact that they can be applied online and to their simplicity, as they can be expressed almost entirely by single equations that can be implemented with small computer programs.

Another relevant aspect of RL is how to determine, represent and store the value functions and policies. When the state and action spaces are small, they can be exhaustively represented as arrays or lookup tables and in this case methods can often find exact solutions for the optimal value functions and policies. However, in many practical problems the state space can be extremely large, so it is unpractical to find an optimal policy or value function. Therefore, in these cases, the goal is to find a good approximate solution using limited computational resources. Moreover, the problem with large state spaces is not just the memory needed for representing them with large tables, but the time and data needed to fill them accurately because there will be many states that will never have seen before. Thus, to make sensible decisions in one of these states it is necessary to generalize from previous encounters with different states that are in some sense similar to the current one. Such generalization can be achieved by means of function approximation methods that take examples from a desired function and attempt to generalize from them to construct an approximation of the entire function. Deep neural networks can be used as effective function approximation methods, and its combination with RL methods has been successfully applied along a wide range of applications. One example is the Deep Q-Network (DQN), that combines Q-learning with a deep convolutional neural network [17].

B. *Applicability domains of RL in the RAN*

Machine learning techniques have applicability for dealing with multiple problems in different areas of the RAN such as physical layer processing, Medium Access Control

(MAC), Radio Resource Management (RRM), Radio Network Management (RNM) or Self-Organized Networks (SON). However, given the different nature of the involved problems and the different principles behind the operation of each category of ML techniques, some of them become more appropriate than others in front of a given problem. Then, in contrast to supervised and unsupervised techniques, which are trained based on labeled and unlabeled data, respectively, and are typically used for tasks involving classification, regression, prediction or feature extraction, RL algorithms are conceived for decision making by learning from the interactions with the environment on the basis of trial and error. Therefore, they are suitable for problems involving some sort of decisions that can be translated into actions. As noted by [2], they have been used in the literature for problems such as cell selection, channel selection, resource allocation (scheduling), power allocation, small cell activation/deactivation for energy efficiency, adaptive modulation and coding, etc. Overall, the problems where RL techniques are applicable are mainly associated to the RRM functions, which dynamically manage the provisioned resources operating at different time scales from the milliseconds to a few seconds, and to the RNM and SON functions, which support the network planning, configuration, optimization and fault management of the RAN operating at longer term time scales.

III. Training RL Models in the RAN

In relation to the training, the most important feature distinguishing RL from other types of learning is that the training uses evaluative feedback (i.e. indicates how good the action taken was), rather than instructive feedback (i.e. indicates the correct action to take), which is the basis for supervised learning. This characteristic creates the need for active exploration, searching explicitly for good behavior [13]. Indeed, the need to balance exploitation and exploration is a distinctive challenge that arises in RL. Exploitation refers to selecting the greedy actions, i.e. the actions whose estimated value is the highest one with the current knowledge that has been acquired at a certain time. In turn, exploration refers to selecting non-greedy actions, e.g., actions selected randomly. Exploration allows identifying actions that may produce greater total reward in the long run.

Indeed, one of the key claimed advantages of RL techniques, which is the ability to progressively learn from the interactions with an environment without any a-priori knowledge of what are the good and bad decisions, becomes paradoxically one of the key challenges that needs to be addressed to facilitate the practical adoption of RL in real RANs. This is because the fundamentally online learning behaviour based on trial and error of RL in the real RAN environment may lead to unacceptable degradations in network performance during the learning or even to dangerous situations (e.g., in autonomous driving or healthcare).

In order to deal with this problem, the possibility of training RL algorithms in simulated environments has been considered in different areas, such as robotics [18], cooperative driving [19] or healthcare [20]. Similarly, another approach is the so-called offline RL [21]. It is a data-driven RL in which the algorithm is trained using only previously collected offline data without any further additional online interaction with the environment. This offline data will be a static dataset of transitions between states, the actions taken and the associated rewards. For example, in a healthcare problem, this offline data could be obtained from previous treatment histories of real patients, including the actions that were selected by their doctors.

These approaches have already been considered when dealing with solutions for some functionalities in the RAN. The use of simulated environments for training has been considered in the coverage and capacity optimization (CCO) solution in [22], which uses a pre-trained policy in a simulator to adjust the user scheduling parameters in each antenna sector in the real network, and in the capacity sharing solution in [23], where training is also performed in a network simulator based on synthetically generated data. Moreover, the offline RL approach has been considered in the Deep Reinforcement Learning (DRL)-based mobility load balancing (MLB) solution in [16], which performs the training on a network evaluation system based on historical performance records gathered from the real environment. We note that the decisions made by these MLB, CCO and capacity sharing functionalities impact on a large number of users, so bad actions can highly impact on network performance (e.g., increase call blocking or call dropping rate).

A. *Functional model of the RL training process*

Fig. 1 depicts the functional model for RL training proposed in this paper. It starts from the main components involved in the O-RAN's Alliance ML workflow [8], colored in blue in Fig. 1. The figure also depicts in orange color different components that are proposed in this work to deal with the peculiarities of the training of RL-based models. The proposed RL-specific components are split between the training host, where the RL model is trained using a training environment, and the inference host, where the trained model is executed directly on the subject of actions, which is the entity of the real RAN environment that is configured by the actions of the model. Examples of subjects of actions in the O-RAN architecture are the O-RAN Central Unit (O-CU), which hosts the upper layers of the radio interface protocol stack in a gNB, the O-RAN Distributed Unit (O-DU), which hosts the lower layers of the protocol stack, and the O-RAN Radio Unit (O-RU), which hosts radio-frequency and low physical layer functions. Other subjects of actions can be the RAN Intelligent Controllers (RICs), namely, the near-real time RIC (near-RT RIC), which enables the optimization of the RAN through, e.g., RRM functions operating at short term time scales, or the non-real time RIC (non-RT RIC), which allows the optimization of the RAN from the service management and orchestration layer.

The RL model training is carried out by the RL agent at the training host under the *model training & testing* functionality through the dynamic interaction with the training environment, which could be a simulated model of the real RAN or a trial RAN (i.e., a RAN deployed in a real environment though carrying test traffic instead of real customers). The model training consists in a policy iteration process in which the RL agent iteratively observes the state of the environment, triggers an action, obtains the reward and uses this information to improve the policy. When selecting the actions, the RL agent makes use of the both exploration and exploitation in order to select the actions. For example, this can be done by means of an ε-greedy policy that, with probability 1-ε selects the best action based on the currently available policy at that time (i.e. exploitation), and with

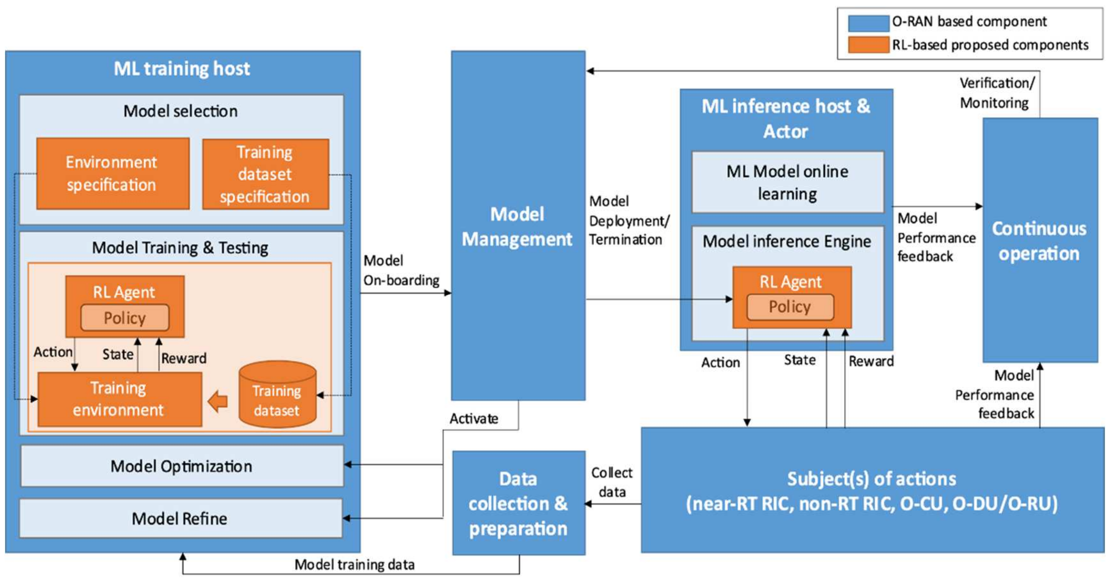

Fig. 1.- Functional model of the RL model training process.

probability ε selects a random action (i.e. exploration). The amount of exploration can be progressively reduced as the training evolves, e.g., by reducing the value of ε. The training is monitored to control the model convergence and gather Key Performance Indicators (KPIs) from the agent such as memory used, loss, accuracy, etc. Based on this, the decision on whether the training process is completed or not can be taken in accordance with a given termination criterion (e.g., a convergence condition is met, a maximum number of iterations is reached, etc.). Once this occurs, the model testing can be executed, consisting in validating the behavior of the resulting model using the training environment and gathering different RAN-related KPIs (e.g., throughput, error rates, etc.).

Within the training host, the *model selection* function selects the configurations for the inference and the training of the RL model based on the analysis of the requirements of the RL solution e.g., training accuracy, training time, hardware resources in training and inference, inference speed, etc. To control and configure the RL training process, the *environment specification* function and the *training dataset* specification function are proposed to be included in the *model selection* functionality. The *environment specification* function defines the relevant features of the real environment that need to be captured by the training environment. In a RAN, this includes the characterization of aspects such as the supported services, the propagation conditions, the mobility, as well as the relevant RAN functionalities to be considered, such as the scheduling, handover or admission control functions. Further details are given in Section III.B.

Complex environments are typically characterized by a multiplicity of (possibly dynamic) configuration parameters (e.g. different load distributions in time/space in a RAN, different cell parameters, etc.) that will eventually determine the visited states by the RL agent. All these different configurations need to be captured in a properly designed training dataset by the *training dataset specification* function, which determines the set of configurations of the training environment to be used when training the RL model. This includes configuration parameters of the RAN nodes (e.g. transmitted power levels, handover parameters, etc.), service requirements (e.g. required throughput, required delay, etc.) and dynamic distributions of the load in time and space for the different cells of the RAN. The selection of the training dataset configurations should be done to cover the relevant situations that the RL agent will have to face when deployed on the real RAN environment. Further details are discussed in Section III.C.

Moreover, the training host includes the *model optimization* and *model refine* functions. The former allows the optimization of the hyperparameters of the RL model based on certain hardware or performance metrics requirements such as model accuracy, model size, inference speed, memory used, etc. In turn, the *model refine* functionality allows, if required, the upgrade of the model through re-training after the model has been already deployed in the inference host.

The management of the deployed RL model in the inference host is performed by the *model management* function. Once the training is completed, this function onboards the resulting trained policy to the RL agent in the *model inference engine* of the inference host. In this way, the RL model is deployed on the real RAN environment. Depending on the RL algorithm the transferred policy can take different forms. For example, in the case of a Q-learning algorithm the model will be a look-up table with the action value functions for each action/state pair. Similarly, in the case of DQN the model will consist in the set of weights of the deep neural network. Moreover, the *model management* function is responsible for the termination of a RL model deployment or its replacement by a different one.

The RL agent at the inference host applies the trained policy on the subject of actions. The policy is applied continuously, repeating the process of observing the state and following the trained policy to decide the best action for that state. No exploration (or at maximum a limited and controlled exploration) is considered at this stage when selecting the actions, to prevent the potential performance degradation associated to random action selections.

The *continuous operations* function provides a series of online functionalities to support the continuous improvement of the RL model within its lifecycle (i.e., deployment, modification, retraining and termination of a RL model). This includes the functions for the verification, monitoring, analysis, model improvement recommendation or the continuous optimization of the RL model. Therefore, while the RL model is executed, the *continuous operations* function continuously analyzes measurements and KPIs from the real RAN environment and the RL agent. This allows the detection of changes in the real environment conditions with respect to those that were considered in the training (e.g., different channel conditions, load distributions that substantially differ from those considered in the training dataset, changes in the deployed network such as the addition of a new base station, etc.).

In some cases, the capability of the RL agent to generalize from the situations observed in the training dataset may still allow a satisfactory performance. However, in some other cases, these new conditions may lead to performance degradations because the trained policy will no longer be optimal. Hence, a retraining of the policy is needed to update the policy so that an optimal behavior is achieved under the new conditions. To detect the need of retraining, the *model management function* should include a retraining condition, dependent of the specific RL problem. For example, this can be based on the definition of target KPIs thresholds, so that the retraining is triggered when certain KPIs are above or below these thresholds.

Depending on the nature of the detected changes, the *model management* function may decide to activate the retraining through the *ML online learning* function to update the RL model based on the real interaction with the subject of actions in the real RAN environment. This can be done by configuring a certain amount of exploration of the RL agent (e.g., by increasing the value of ε in case of an ε-greedy policy). In this case, the reward signal obtained from the subject of actions can be used to progressively update the policy. In turn, in case of substantial changes that may severely impact on the performance, the *model management function* may activate the *model refine* function in the *training host* to retrain the RL model to derive a new policy able to deal with the new conditions. For this purpose, it will send updated information about the real RAN environment to the *environment specification* and *training dataset* specification functions so that they take it into consideration when defining the new features of the *training environment* and an upgraded *training dataset*. This information can be gathered from the *data collection & preparation* function. The way of performing this retraining in terms of required data and retraining duration highly depends on the specific RL problem. In addition to the retraining capability, the *model management* function should also include protection mechanisms to terminate the deployment of a RL model in case a malfunctioning of the RL model is detected (e.g. extreme KPIs degradation, anomalies, etc.). Thus, a termination criterion needs to be established and a backup solution needs to be deployed in the *ML inference host*. The backup solution can be either another well-trained RL model or, a non-AI-based algorithm. For example, this can be relevant in case that the retraining duration is long and the performance of the actual model further degrades before the retraining is completed.

B. *RL training environment*

The RL training environment enables the execution of the trial and error process inherent to RL under a safe operation environment that does not suffer the consequences of bad decisions. Whether the RL training environment is a simulated model of the RAN or a trial RAN, it needs to include the elements that are relevant for the RL model under test and should be configured to properly capture similar operation conditions than those that the RL agent will experience when deployed on the real RAN environment, for example, in terms of propagation environment, types of traffic, number of users, etc.

In the case that the RL training environment is a simulated model, the different RAN functionalities can be modeled with different level of detail in the simulator depending on their relevance for the problem at hand, thus arising a trade-off between simulator complexity, agility in the training process and accuracy of the trained model. For example, the introduction of a detailed model for the physical signal transmission and reception at the symbol level will significantly slow down the simulation process but it might not have any relevant impact on the accuracy of the results if the RL model only cares about average transmitted/received signal levels (e.g. when RL is used by the handover function).

Then, it is envisaged that, for RL models supporting MAC level functionalities (e.g. scheduling) operating in the millisecond time scale, the RL training environment should accurately model aspects such as the instantaneous channel conditions (e.g. Channel Quality Indicator) of each user in a cell, the packet queues impacted by the per-user traffic generation process, the retransmission schemes (i.e. Automatic Repeat reQuest (ARQ) and Hybrid ARQ at Radio Link Control (RLC) and MAC layers, respectively) or the beamforming and spatial multiplexing approaches. In turn, when training RL models for RRM or RNM functions operating at longer term time scales, some of the abovementioned effects may be modelled in a more averaged and/or aggregated way (e.g. by considering only average channel conditions or an aggregate model of the traffic in a cell). Instead, other elements such as user mobility may need to be considered and properly modelled.

C. *RL training dataset specification*

The training dataset should be specified in a way that the RL training environment is driven to produce those situations (i.e., states) that the RL agent may face when it will run in the real RAN environment, including not only those that are expected to occur frequently but also those that will be less usual. For this purpose, the relevant parameters of the training environment impacting on the RL model need to be identified and adequately configured to ensure that a sufficiently representative number of states are encountered by the RL agent, so that the agent can properly generalize from them.

The relevant parameters to configure the environment will be typically related with how the state is defined and their values in the dataset should be set in accordance with a dynamic range that reflects those situations in which the agent is supposed to learn how to behave. Moreover, it should be ensured that the different situations are observed a sufficient number of times so that the RL agent is able to consolidate the actions to be selected when being in these situations. All these aspects will determine the size of the training dataset that will eventually impact on the training duration. In practice, it may happen that the specification of a dataset that covers a big range of possible situations leads to an extremely large dataset that makes impractical the training process. In this case, a trade-off can be found by having a more reduced training dataset and then relying on model retraining if needed to deal with new situations.

The specification of the training dataset will be highly dependent of the RL problem at hand. For example, when training an RL model for scheduling, the training dataset can be configured to produce different channel model statistics (e.g. doppler frequencies, line-of-sight and non-line-of-sight situations), different numbers of users per cell and per service or different service requirements.

The training dataset can make use of data collected from the real network (e.g. actual channel measurements, measurements of the number of users connected to a cell at different periods of time, etc.). This data can be extended with other synthetic data in order to generate those situations that are more difficult to encounter in the real network.

IV. Applicability Example

The O-RAN-based functional framework for the training of RL in Fig. 1 can be adopted for multiple RL problems in the RAN. In this section, an applicability example of the functional model of the RL training process is provided for the use case of capacity sharing in RAN slicing. The considered use case and the scenario for evaluation are described in Section IV.A and Section IV.B, respectively. The evaluation presented in Section IV.C focuses on the retraining aspects of the proposed functional model. Finally, Section IV.D provides some insight on other possible applicability examples.

A. Use case description

This section illustrates the operation of the proposed functional framework and the role of its components when applied to the problem of capacity sharing in RAN slicing. The DQN-Multi-Agent Reinforcement Learning (MARL) capacity sharing solution for multi-tenant and multi-cell scenarios defined in our recent publication [23] is considered. The solution dynamically distributes the available capacity in a RAN infrastructure composed of N cells, where each cell n has a total cell capacity c_n (b/s), among K tenants, each of them provided with a RAN slice. The solution targets the efficient use of the available capacity in the cells and, at the same time, the satisfaction of the Service Level Agreement (SLA) of the tenants. The SLA established for the k-th tenant is defined in terms of: (a) the Scenario Aggregated Guaranteed Bit Rate, $SAGBR_k$, which is the aggregated capacity to be provided across all cells to tenant k if requested, and (b) the Maximum Cell Bit Rate, $MCBR_{k,n}$, which is the maximum bit rate that can be provided to tenant k in cell n.

The DQN-MARL capacity sharing solution considers that each tenant is associated to a different RL agent as depicted in Fig. 2, which shows the deployment of the DQN-MARL capacity sharing solution in the *model inference engine* of the inference host. An RL agent tunes the resource quota (i.e. the fraction of capacity) assigned to the tenant's slice in the different cells in time steps of duration Δt. For this purpose, the RL agent obtains the state of the tenant in the different cells of the environment. The state is defined as a tuple with different metrics. These include, for each cell, the resource usage and resource quota of the tenant, the resource quota not assigned to any tenant and the resources not used in the cell. In addition, the state includes the SLA parameters of the tenant. According to the obtained state, the RL agent decides the actions to perform in each cell, which can be to increase the resource quota in Δ, to decrease it in Δ or to keep it unaltered. The action selection is performed dynamically in time steps of duration Δt. Based on the selected actions by the

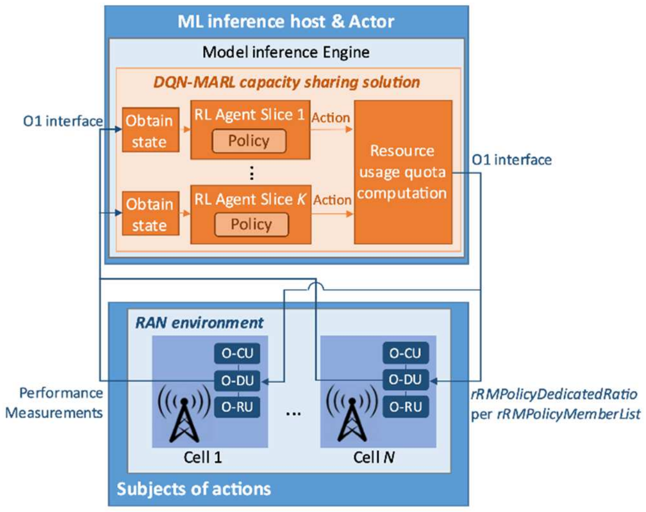

Fig. 2. Deployment of the DQN-MARL solution in the ML inference host.

different RL agents, and given that the decisions are taken separately by each agent, the resource usage quota computation function of Fig. 2 determines the actual resource quota to apply in each cell, ensuring e.g., that the resulting resource quota of all slices within a cell do not exceed 1 (i.e., the total cell capacity). For further details on definitions of the state and action, the reader is referred to [23].

From an implementation perspective, the resource quota of a slice is configured through the so-called *rRMPolicyDedicatedRatio* attribute of each cell, as detailed in [24]. As seen in Fig. 2, this attribute is configured in the O-DU unit that handles each cell and that is responsible of the high-physical layer processing of 5G New Radio, MAC and RLC functionalities. Then, the O-DU acts as the subject of actions and its MAC layer performs the allocation of Physical Resource Blocks (PRBs) to the users of the RAN slice based on the value of *rRMPolicyDedicatedRatio*. The configuration of this attribute is conducted through the O1 interface, defined in the O-RAN architecture for the management provisioning services [26]. Moreover, this interface is also used to obtain the performance measurements from the O-DU of each cell. These measurements are used to determine the states of the RL agents used to select new actions.

The training of the DQN-MARL solution is performed in the *training host,* where the *training environment specification* considers the same values of K, N, c_n, $SAGBR_k$, $MCBR_{k,n}$ and cells distribution as in the real RAN environment. The training environment is fed with the data of the *training dataset*, which is composed of multiple temporal patterns of the offered load of the K tenants (i.e., slices) and different combinations of their SLA values. The offered load is defined as the requested capacity in bits/s of a tenant normalised to the total cell capacity and averaged during a time step. An offered load pattern includes the time evolution of the offered load of a tenant during T time steps. The *training dataset specification function* specifies the number of offered load patterns included in the dataset, the range of values of offered loads and the SLA parameters of the different tenants, as discussed in our previous work [27]. The *optimization* function selects the hyperparameter values of the DQN agents in the DQN-MARL solution after following a try and error procedure and choosing those values that achieve a better training performance.

Using the selected hyperparameters configuration, the training is performed by consecutively applying the temporal patterns in the *training dataset* to the *training environment* and by letting the DQN algorithm update the policy based on the interactions with this environment. During the training, the RL agent of each slice obtains the state from the training environment and, accordingly selects an action to update the resource quota using an ε-greedy strategy. At the next time step, a reward is obtained. The reward formulation (see [23] for details) promotes the satisfaction of the SLA parameters and the minimization of overprovisioning situations. Then, the RL agent stores in a dataset the experience composed by the last state and action and the resulting state and obtained reward. The policy update is performed in every time step during the training process using the information of the dataset of experiences and following the procedure of [25]. The training process is monitored in terms of the loss function until reaching a convergence criterion [27].

Once the training is completed, the *model management* function on-boards the trained policies for the different tenants in the RL agents of the *model inference engine*. During the inference, the *continuous operations* monitors the model performance feedback in terms of the following KPIs:

- Average SLA satisfaction per tenant: computed as the average ratio between the aggregate throughput of the tenant in the scenario and the minimum between the aggregate of the offered load in the cell, bounded to the $MCBR_{k,n}$, over all cells and the $SAGBR_k$. This average is computed during a window of T time steps.
- Average utilization ratio: average ratio between the system utilization and the aggregated assigned capacity to all tenants in the system during T time steps. The system utilization is computed by dividing the aggregated throughput of all tenants among all cells by the system capacity.
- Average distance between the training and the inference data: This quantifies the similarity between the training and inference data and is obtained by computing the Euclidean distance between the offered load values in each cell at each time step during inference and the closest offered load of the training dataset and then by performing the average of this distance over the last T time steps.

Using the above KPIs obtained by the *continuous operation* function, the *model management* function can detect if a re-training of the solution is required and consequently it can activate the *model refine* function to upgrade the trained policies.

B. Considered scenario

The DQN-MARL capacity sharing solution is applied in a RAN scenario with K=2 tenants, denoted as Tenant 1 and Tenant 2, and N=5 cells, which are distributed in an area of 3km x 3km. Each cell has a total cell capacity c_n= 140 Mb/s, so the total system capacity is C=700 Mb/s. The established SLAs are $SAGBR_1$=420 Mb/s and $SAGBR_2$=280 Mb/s, corresponding to the 60% and 40% of the system capacity, respectively, and the $MCBR_{1,n}$=$MCBR_{2,n}$=112 Mb/s, corresponding to the 80% of c_n.

For evaluation purposes, both the *training environment* and the real RAN environment are implemented by a RAN simulator configured according to the considered scenario. The *optimization* function considers the hyperparameters shown in Table I. The selected values in Table I correspond to the those providing the best trade-off between the training duration and loss after testing different possibilities.

The DQN-MARL capacity sharing solution has been initially trained with an action step Δ=0.03 and applying the offered loads patterns of the *training dataset*, each one with duration T=288 time steps. These patterns have been defined considering a homogenous distribution of the offered load among the different cells, which means that, for a given tenant, very similar offered loads are assumed in all the cells. The training is completed after $2\cdot10^6$ time steps. The performance of the learnt policy, denoted as *Initial Policy*, has been monitored in the RAN environment during T=288 time steps.

C. Performance results

This section analyses the performance achieved during the inference stage. The achieved performance of a policy during inference depends on the generalization capability of the trained policy to adapt to situations not included in the training dataset but also on the similarity between the training dataset and the experienced data during inference, as discussed in our previous work [27]. Deepening into these aspects, the presented results intend to, on the one hand, quantify the performance degradations that arise if the initial training does not match the actual conditions experienced during inference and, on the other hand, to illustrate the capability to retrain the RL model, as considered in the proposed functional model of Fig. 1.

To this end, we start by considering that the abovementioned *Initial Policy* is generated and deployed by the *model management* in the *model inference engine*, which starts applying it at time t_o. Fig. 3 shows the evolution of the average SLA satisfaction for Tenant 1 and Tenant 2 and the average utilization ratio. Starting at t_o and for the first 600 time steps the traffic distribution across cells is homogeneous. Correspondingly, since the traffic considered for the training of the *Initial Policy* exhibited this feature, high KPI values are observed in Fig. 3 during this period, reflecting the good performance of this policy. Specifically, the average SLA satisfaction is around 96% for both tenants and the average utilization ratio around 95%.

TABLE I. DQN-MARL MODEL HYPERPARAMETERS

Parameter	Value
Initial collect steps	5000
Maximum number of time steps for training	$2\cdot10^6$
Experience Replay buffer maximum length (l)	10^7
Mini-batch size (J)	256
Learning rate (τ)	0.0001
Discount factor(γ)	0.9
ε value (ε-Greedy)	0.1
DNN configuration	Input layer: 17 nodes 1 full connected layer: 100 nodes Output layer: 243 nodes

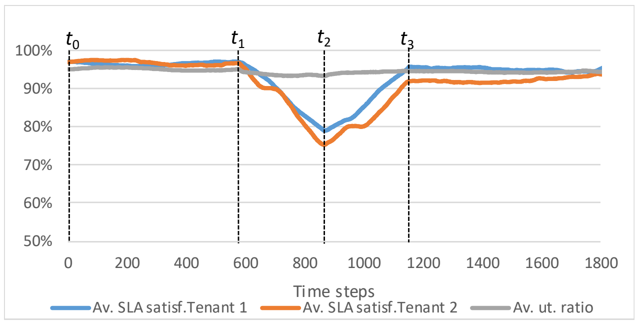

Fig. 3. Evolution of the average SLA satisfaction and utilization ratio.

Then, at t_1, we assume that a change of the offered load distribution of both tenants occurs due to the appearance of hotspots, leading to a heterogeneous load distribution across cells. This is represented in Fig. 4, which depicts the offered load density in Mb/s/km^2 of Tenant 1 and Tenant 2 at some illustrative time instants relative to t_1. This pattern is repeated every 288 time steps from t_1 until the end of the analyzed period. While the average utilization ratio remains high after t_1, as seen in Fig. 3, the average SLA satisfaction values of both tenants decrease significantly. Then, the *continuous operations* function assesses the average distance between the training and inference data. Fig. 5 shows the evolution of this average distance during the same period of Fig. 3. From t_0 to t_1 the average distance remains at a low value of around 0.08, indicating that the experienced data during inference and the training dataset are similar. However, when the offered load distribution among cells becomes strongly heterogeneous at t_1, the average distance starts increasing abruptly and a decrease of the SLA satisfaction of both tenants follows. Based on this, the *model management* function determines that a retraining of the RL model is required and activates the *model refine* function in the training host. Indeed, the observations made in Fig. 3 and Fig. 5 reflect the trade-off between the achieved performance and the complexity of the training process. While the initial training has been sufficient to achieve a satisfactory performance until t_1 without the need of a complex training dataset, this has not been enough to generalize the policy to the offered load situations that arise after t_1 and thus the average SLA satisfaction values of both tenants degrade in around 25%. Therefore, a richer and more complex training dataset with a wider range of offered load values for the different cells is needed for the retraining stage. This new dataset is built considering the offered load conditions observed after t_1 during inference, gathered by *the data collection & preparation* function, and applying data augmentation to increase the size of the training dataset.

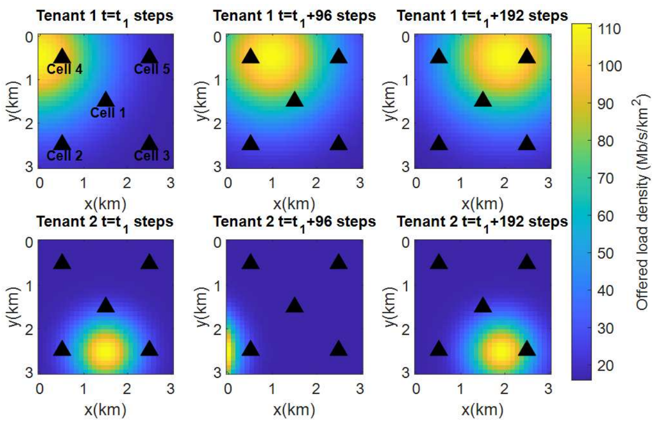
Tenant 1 t=t$_1$ steps
Tenant 1 t=t$_1$+96 steps
Tenant 1 t=t$_1$+192 steps
Tenant 2 t=t$_1$ steps
Tenant 2 t=t$_1$+96 steps
Tenant 2 t=t$_1$+192 steps
Cell 1
Cell 2
Cell 3
Cell 4
Cell 5
x(km)
y(km)
Offered load density (Mb/s/km^2)

Fig. 4.- Offered load density maps of Tenant 1 and 2 at some illustrative time instants.

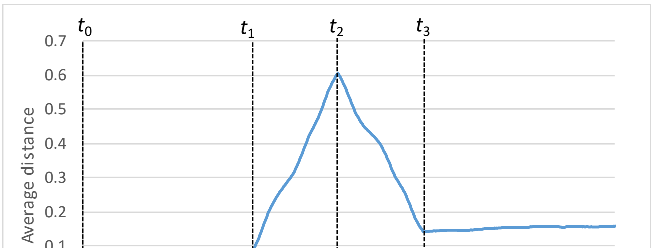

Fig. 5. Average distance between the training and inference data.

After retraining, the *model management* replaces the *Initial Policy* by the upgraded policy, denoted as *Re-trained Policy*, in the inference host at time t_2. As a result, an increase of the average SLA satisfaction of both tenants is experienced in Fig. 3, achieving again values above 0.9 from t_3 until the end of the analyzed period. Indeed, improvements of 22% with respect to the ones in t_2 are achieved for both tenants. This improvement is also justified by the smaller average distance between the training and inference data shown in Fig. 5. It is observed that this average distance starts to decrease when the *Re-trained Policy* is applied at t_2 and achieves again low values of around 0.15 after t_3.

The above results suggest that the practical exploitation of the proposed framework could embrace an initial simulation-based training keeping the simulator complexity at a relatively low extent and a subsequent retraining exploiting data extracted from the real RAN environment with proper data augmentation techniques to speed up the process if needed. In this way, the learnt policy can be upgraded to capture the new situations that may arise during inference.

D. Other applicability examples

In order to provide a wider view on the applicability of the proposed framework, two other use cases are discussed in the following.

The first example is the MLB problem, in which cells suffering congestion can transfer load to other cells with less traffic. The control of this load transfer is typically done using the handover function executed at the O-CU, and can make use of RL algorithms. An example is given in [16], which proposes a DRL solution to tune a parameter named cell individual offset considered in the handover decision for a user. The proposed framework can be used to train an RL-based MLB solution using a *training environment* composed of multiple cells in a certain area and considering multiple heterogeneous spatial distributions of the load in the different cells. Then, the *training dataset* will be defined by these spatial distributions, together with the cell configuration parameters and the user mobility patterns. This data could be based on real data gathered and processed by *data collection & preparation* function but also on synthetic models for generating user trajectories. The trained policies will be deployed in the *ML inference host* by considering as subject of actions the O-CU. To monitor the performance of the deployed MLB policies, relevant KPI parameters to check are the occupation among the cells and the quality metrics of the UE, such as the Signal to Noise and Interference Ratio (SINR). These quality metrics are particularly important because wrong MLB decisions may lead to significant interference increases if a UE is not served by an adequate cell, so they can be used to detect if a retraining is needed.

Another applicability example is the CCO problem, which consists in adjusting certain cell parameters (e.g. antenna tilts, reference signal powers, scheduling parameters, etc.) to optimize the resulting capacity and coverage. An example of the use of DRL for CCO is given in [22], which adjusts two specific parameters of their scheduling algorithm. DRL-based solution to configure user scheduling parameters. For this type of solutions, the *training environment* has to consider multiple cells serving multiple users and needs to carefully model the short term channel variations of each user and the scheduling process, thus having to operate at time scales in the order of 1 ms. The *training dataset* should then include different numbers of users and requirements and can be enriched with Channel Quality Indicators of real cells to characterize a certain network environment. The ML inference host and the subject of actions would depend on the specific parameter adjusted by the CCO. For example, in the solution of [22], both of them would be the O-DU function.

V. Conclusions and Future work

Motivated by the high interest that the applicability of RL techniques has raised for different problems in the RAN and by the little attention that has been paid so far to the training of these algorithms, this paper has presented a framework for training RL algorithms in the RAN that is aligned with the ongoing work by O-RAN Alliance on the application of ML for the RAN. The proposed framework complements the O-RAN approach through the incorporation of specific functions for the training of RL solutions. These include different functionalities for specifying training datasets to capture the situations that will be faced by the model in the real network and for continuously monitoring the performance of the inference stage to trigger a retraining if needed. The operation of the proposed framework has been illustrated with a DQN-based algorithm for capacity sharing. It has been shown that, when there are significant differences between inference and training data, performance degradations of around 25% are observed, which can be overcome by means of a proper retraining process. In addition, the applicability of the framework for RL-based solutions to the problems of mobility load balancing and coverage and capacity optimizations are discussed.

The considered DQN-MARL capacity sharing solution has allowed illustrating the operation of the O-RAN-based functional framework proposed in this paper for the specific problem of capacity sharing in RAN slicing scenarios. Two future research directions are identified. The first one is the validation of the proposed framework using a testbed in order to study its feasibility and implementation complexity when deploying it in a real RAN, treating aspects such as the software and virtualization tools required to deploy the different components of the proposed framework as well as the interfaces that enable the interaction between them. The second research direction is the assessment of the proposed framework for a wider range of problems other than the capacity sharing, in order to validate the generality of the proposed framework and to identify potential enhancements.

Acknowledgment

This paper is part of ARTIST project (ref. PID2020-115104RB-I00) funded by MCIN/AEI/10.13039/501100011033 and PORTRAIT project (ref. PDC2021-120797-I00) funded by MCIN/AEI/10.13039/501100011033 and by European Union Next GenerationEU/PRTR.

References

[1] Y. Sun, M. Peng, Y. Zhou, Y. Huang and S. Mao, "Application of Machine Learning in Wireless Networks: Key Techniques and Open Issues," *IEEE Comms. Surveys & Tutorials*, vol. 21, no. 4, pp. 3072-3108, 4th Quarter 2019.

[2] J. Wang, et.al, "Thirty Years of Machine Learning: The Road to Pareto-Optimal Wireless Networks" *IEEE Comms. Surveys & Tutorials,* Vol. 22, No. 3, 3rd Quarter, 2020.

[3] C. Zhang, P. Patras, H. Haddadi, "Deep Learning in Mobile and Wireless Networking: A Survey", *IEEE Comms. Surveys & Tutorials,* Vol. 21, No. 3, 3rd Quarter, 2019.

[4] Q. Mao, F. Hu, Q. Hao, "Deep Learning for Intelligent Wireless Networks: A Comprehensive Survey", *IEEE Comms. Surveys & Tutorials,* Vol. 20, No. 4, 4th Quarter, 2018.

[5] M. Elsayed, M. Erol-Kantarci, "AI-enabled Future Wireless Networks. Challenges, Opportunities and Open Issues", *IEEE Vehicular Technology Magazine*, Sept. 2019.

[6] F. D. Calabrese, et. al, "Learning radio resource management in RANs: Framework, opportunities, and challenges," *IEEE Commun. Mag.*, vol.56, No. 9, Sept. 2018

[7] R. Ferrús, O. Sallent, J. Pérez-Romero, R. Agustí, "Applicability Domains of Machine Learning in Next Generation Radio Access Networks", 6th *Annual Conference on Computational Science & Computational Intelligence* (CSCI-2019), Las Vegas, USA, Dec. 2019

[8] O-RAN Alliance O-RAN.WG2.AIML-v01.03, "AI/ML Workflow Description and Requirements 01.01", July, 2021.

[9] ITU-T Y.3172, "Architectural framework for machine learning in future networks including IMT-2020", June 2019.

[10] ETSI GR ZSM 005 V1.1.1, "Zero-touch network and Service Management (ZSM); Means of Automation", May, 2020.

[11] ETSI GS ENI 005 V1.1.1, "Experiential Networked Intelligence (ENI); System Architecture", September, 2019.

[12] RP-213599, "Study on Artificial Intelligence (AI)/Machine Learning (ML) for NR Air Interface", 3GPP TSG RAN Meeting #94e, December, 2021.

[13] R. S. Sutton, A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd edition, The MIT Press, 2018.

[14] Y. Abiko, et. al "Flexible Resource Block Allocation to Multiple Slices for Radio Access Network Slicing Using Deep Reinforcement Learning," in *IEEE Access*, vol. 8, pp. 68183-68198, 2020.

[15] J. Pérez-Romero, O. Sallent, R. Ferrús, R. Agustí, "A Robustness Analysis of Learning-based Coexistence Mechanisms for LTE-U Operation in Non-Stationary Conditions", *IEEE Vehicular Technology Conference Fall* (VTC Fall 2015), Boston, USA, Sept., 2015.

[16] Y. Xu, W. Xu, Z. Wang, J. Lin and S. Cui, "Load Balancing for Ultradense Networks: A Deep Reinforcement Learning-Based Approach," in *IEEE Int. of Things Journal*, vol. 6, no. 6, Dec. 2019.

[17] V. Mnih, et al. "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[18] W. Zhao, J. Peña Queralta, T. Westerlund, "Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey", IEEE Symp. Series on Computational Intelligence (SSCI), 2020.

[19] A. Chandramohan, et al. "Machine Learning for Cooperative Driving in Multi-Lane Highway Environments", 2019 Wireless Days (WD).

[20] E. Tagliabue et al. "UnityFlexML: Training Reinforcement Learning Agents in a Simulated Surgical Environment", I-RIM Conf., 2020.

[21] S. Levine, A. Kumar, G. Tucker, J. Fu, "Offline Reinforcement Learning: Tutorial, Review and Perspectives on Open Problems", Nov. 2020, https://arxiv.org/abs/2005.01643v3

[22] Y. Yang et al., "DECCO: Deep-Learning Enabled Coverage and Capacity Optimization for Massive MIMO Systems," in *IEEE Access*, vol. 6, pp. 23361-23371, 2018.

[23] I. Vilà, J. Pérez-Romero, O. Sallent, A. Umbert, "A Multi-Agent Reinforcement Learning Approach for Capacity Sharing in Multi-tenant Scenarios," in *IEEE Trans. Veh. Tech.*, vol. 70 no. 9, July 2021.

[24] 3GPP TS 28.541 V16.5.0, "Management and orchestration; 5G Network Resource Model (NRM) (Release 16)", June, 2020.

[25] V. Mnih, et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[26] O-RAN.WG1.O-RAN-Architecture-Description-v05.00, "O-RAN Architecture Description version 5.00," O-RAN Alliance, Working Group 1, Technical specification, July 2021.

[27] I. Vilà, J. Pérez-Romero, O. Sallent, A. Umbert, "Impact Analysis of Training in Deep Reinforcement Learning-based Radio Access Network Slicing," Accepted in *IEEE Cons. Comms. & Net. Conf.* 2022.