Accelerating Reinforcement Learning via Predictive Policy Transfer in 6G RAN Slicing

Ahmad M. Nagib, *Graduate Student Member, IEEE*, Hatem Abou-Zeid, *Member, IEEE*, and Hossam S. Hassanein, *Fellow, IEEE*

***Abstract*—Reinforcement Learning (RL) algorithms have recently been proposed to solve dynamic radio resource management (RRM) problems in beyond 5G networks. However, RL-based solutions are still not widely adopted in commercial cellular networks. One of the primary reasons for this is the slow convergence of RL agents when they are deployed in a live network and when the network's context changes significantly. Concurrently, the open radio access network (O-RAN) paradigm promises to give mobile network operators (MNOs) more control over their networks, furthering the need for intelligent and RL-based network management. O-RAN's standardized interfaces will allow MNOs to make real-time custom changes to intelligently control various RRM functionalities. We consider a RAN slicing scenario in which MNOs can modify the weights of the RL reward function. This enables MNOs to change the priorities of fulfilling the service level agreements of the slices. However, this results in a practical challenge since the RL agent needs to adapt promptly to the changes made by the MNO. This challenge is addressed in this paper, where we first present and discuss the results from an exhaustive experiment to examine the efficiency of using transfer learning (TL) to accelerate the convergence of RL-based RAN slicing in the considered scenario. We then propose a novel *predictive* approach to enhance the TL-based acceleration by selecting the best-saved policy for reuse. By adopting the proposed policy transfer approach, RL agents are able to converge up to 14000 learning steps faster than their non-accelerated counterparts. The proposed machine learning (ML)-based *predictive* approach also shows up to a 96.5% accuracy in selecting the best expert policy to reuse for acceleration.**

***Index Terms*—O-RAN, RAN slicing, resource allocation, predictive transfer learning, accelerated reinforcement learning, 6G.**

I. Introduction

NEXT-GENERATION wireless networks will have to deal with growth and heterogeneity on many levels. This includes growth in mobile data traffic and a higher density of mobile users. This also involves a variety of radio access technologies, services, and applications. As a result, various objectives, such as low latency, high reliability, and throughput need to be fulfilled simultaneously based on the service used. Moreover, resource allocation should be dynamically optimized based on the changing network conditions. Nevertheless, given the inherent uncertainty of wireless network environments, conventional approaches for resource management that require perfect knowledge of the network are inefficient [1]. Machine learning (ML), and specifically reinforcement learning (RL)-empowered next-generation wireless networks are vital due to the following reasons [2], [3]:

- *Network Complexity:* Next-generation networks (NGNs) will be more complicated due to the aforementioned reasons. In such complex deployment scenarios, estimating the optimal performance is computationally infeasible given its many-sided heterogeneous nature. ML, however, can address the network complexity while providing competitive performances.
- *Model Deficiency:* Modern cellular networks have been designed with many assumptions to approximate the end-to-end system behaviour using simple modelling approaches. ML-based approaches can be employed to capture the underlying unknown dynamic networks' nonlinearities.
- *Algorithm Deficiency:* The optimal algorithms are too complex to be practically implemented in some network scenarios. This result in system designs that most likely rely on heuristics based on simple rules. ML can strike the right balance between acceptable system performance and complexity in such cases.

RL algorithms have recently gained wide attention in the wireless networks domain [4]. They are considered promising approaches to solving dynamic Radio Resource Management (RRM) problems in NGNs. RL algorithms can deal with the multifaceted complexities of wireless network environments given their capabilities to build an approximate and continuously updated model of such environments. The open radio access network (O-RAN) paradigm will allow the network to be more customizable. It will also enable data-driven network management [5]. With O-RAN, NGNs will include generic modules and interfaces for data collection, distribution, and processing [6]. This way, the mobile network operators (MNOs) will have more control over the network.

Manuscript received 1 May 2022; revised 17 October 2022 and 7 February 2023; accepted 14 February 2023. Date of publication 17 March 2023; date of current version 6 July 2023. This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) under Grant RGPIN-2019-05667 and Grant RGPIN-2021-04050. The associate editor coordinating the review of this article and approving it for publication was N. Zincir-Heywood. *(Corresponding author: Ahmad M. Nagib.)*

Ahmad M. Nagib is with the School of Computing, Queen's University, Kingston, ON K7L 2N8, Canada, and also with the Faculty of Computers and Artificial Intelligence, Cairo University, Giza 12613, Egypt (e-mail: ahmad@cs.queensu.ca).

Hatem Abou-Zeid is with the Department of Electrical and Software Engineering, University of Calgary, Calgary, AB T2N 1N4, Canada (e-mail: hatem.abouzeid@ucalgary.ca).

Hossam S. Hassanein is with the School of Computing, Queen's University, Kingston, ON K7L 2N8, Canada (e-mail: hossam@cs.queensu.ca).

Digital Object Identifier 10.1109/TNSM.2023.3258692

In this paper, we consider a radio access network (RAN) slicing scenario in which MNOs can change the priorities of fulfilling the service level agreements (SLAs) of the available slices [7]. This can be done by tuning the weights of the corresponding KPIs in the RL reward function for each slice. Different deployments and operators may have different preferences that change over time. However, changing the reward function weights can drastically change the system's performance. This raises a practical challenge as the RL agent needs to adapt quickly to such changes. In such cases, MNOs need an efficient way to accelerate the RL agent to quickly converge back to a good policy avoiding extreme instabilities and drops in performance for a long time [8]. This RL-related challenge is rarely tackled in research studies developing RL-based RRM solutions. Even when it is tackled, the proposed approaches are not thoroughly investigated to understand the resulting potential positive convergence behaviour.

Transfer learning (TL) is one of the commonly used approaches to accelerate RL convergence in the wireless networks' domain [9]. One of the main categories of TL is policy transfer. In policy transfer, the policy of a previously trained RL agent, namely an expert agent, is used to guide the exploration phase of a learner agent instead of learning from scratch. The work presented in this paper is related, but quite different from our previous work in [10]. In our previous work, we only investigated the viability of employing TL to address slow convergence in deep reinforcement learning (DRL)-based RAN slicing. In this study, we focus on analyzing and enhancing the RL convergence acceleration behaviour when using the policy transfer approach of TL in a specific O-RAN scenario.

The main contributions of this paper can be summarized as follows:

- We present an O-RAN slicing scenario in which MNOs can change the weights of the RL reward function, and consequently, the priorities of fulfilling the slices' SLAs. Given such a scenario, we present an evaluation methodology to examine the convergence behaviour of the accelerated RL algorithms when policy transfer is applied.
- We perform a thorough analysis of around 3400 simulation runs to study the efficiency of using policy transfer to accelerate RL-based RAN slicing. This includes analyzing the RL convergence speed gains when using a policy reuse approach. This also considers the effect of the distance between the reward function weight vectors of expert and learner agents on the reward convergence error of the accelerated learner agents.
- We propose a novel *predictive* approach to enhance policy transfer acceleration in RL-based RAN slicing. We specifically propose to save the policies of several expert RL agents that are trained using different network slicing reward weights. When a new reward function weight combination is set by the MNO, an ML-based approach is used to select the expert policy with the least expected reward convergence error. This is vital to efficiently accelerate a learner agent when an MNO changes the reward function weight vector that reflects the priority of the various slices' SLAs fulfillment. We train multilayer perception (MLP) and extremely randomized trees (extra-trees) regressor models and compare their performance with a Euclidean distance metric.

To the best of our knowledge, this is the first study to 1) identify the need, and propose TL, to mitigate convergence problems of RL-based O-RAN slicing when MNOs change the reward function weights for different network slices, and 2) propose a mechanism to efficiently accelerate RL for this purpose by using a novel form of *predictive* TL. The rest of the paper is organized as follows. In Section II, we give an overview of the problem. Section III discusses the related work. The system model, the acceleration approach, and the experimental setup are described in Section IV. In Section V, we propose an approach to enhance policy transfer acceleration of RL-based RAN slicing. Section VI provides the reader with a thorough analysis of the results. Lastly, our work is concluded, and some future directions are presented in Section VII.

II. Background

A. Radio Access Network Slicing

Both radio access and core networks are considered parts of the end-to-end network slicing, each with a slightly different optimization goal [11]. Network slicing's objective is to share the physical infrastructure among several services. In this paper, we mainly focus on the RAN part of network slicing. RAN slicing is mainly concerned with two RRM functionalities, slice admission control, and resource allocation. Slice admission control allows an infrastructure provider to accept or deny a service provider's slice request. While slice resource allocation is concerned with assigning the available physical resource block (PRBs) to the slices approved by the admission control function. An overview of RAN slicing and its main functionalities are depicted in Fig. 1.

The available resources at a given time are significantly affected by the stochastic channel quality. Moreover, they are affected by the time-varying user demands for the provided services. The traffic demand for each type of service is dynamic and cannot be easily predicted, particularly in the short term. At the beginning of a slicing window, the available limited resources are assigned among the admitted slices. These allocated resources are expected to enable the services provided by the admitted slices to comply with their different QoS requirements given the dynamic network conditions. The exact requirements are defined by the SLAs and should not be violated by the infrastructure provider, otherwise monetary penalties may be enforced. Hence, RAN slicing cannot tolerate the long RL exploration phase and this poses many challenges for RL-based RAN slicing solutions.

B. Reinforcement Learning-Based Slicing

6G networks are expected to have ubiquitous intelligence. They are also expected to adopt an open architecture. This allows a customizable AI-native RAN slicing [12]. Accordingly, various ML-based approaches have been recently proposed to solve RAN slicing-related problems [13]. The most important feature that distinguishes RL from the other

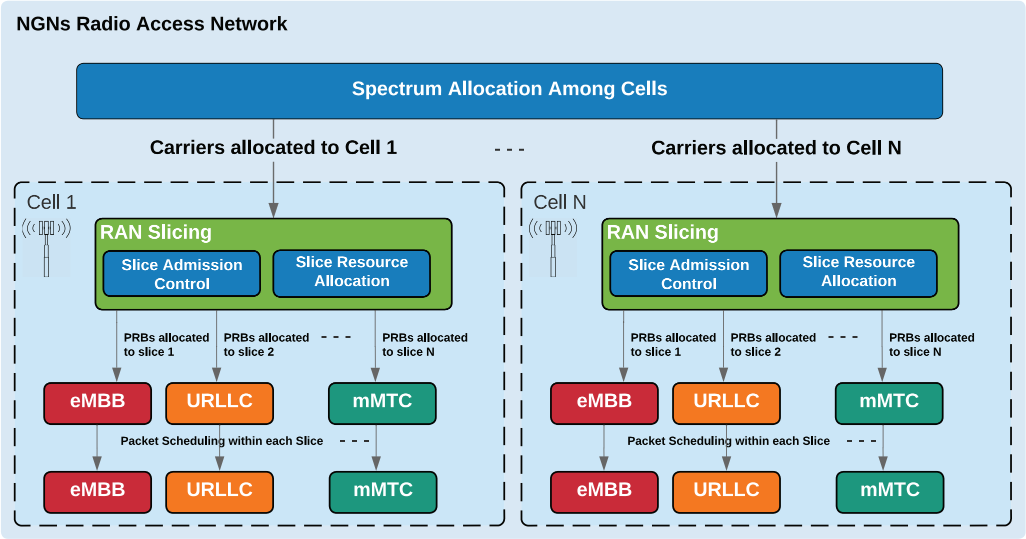

NGNs Radio Access Network
Spectrum Allocation Among Cells
Carriers allocated to Cell 1
- - -
Carriers allocated to Cell N
Cell 1
RAN Slicing
Slice Admission Control
Slice Resource Allocation
PRBs allocated to slice 1
PRBs allocated to slice 2
PRBs allocated to slice N
eMBB
URLLC
mMTC
Packet Scheduling within each Slice
eMBB
URLLC
mMTC
Cell N
RAN Slicing
Slice Admission Control
Slice Resource Allocation
PRBs allocated to slice 1
PRBs allocated to slice 2
PRBs allocated to slice N
eMBB
URLLC
mMTC
Packet Scheduling within each Slice
eMBB
URLLC
mMTC

Fig. 1. Overview of RAN slicing.

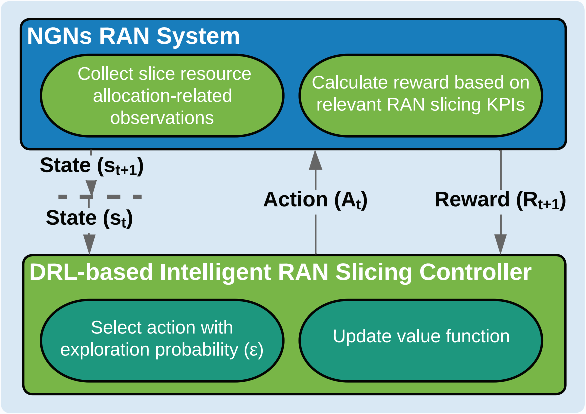

NGNs RAN System
Collect slice resource allocation-related observations
Calculate reward based on relevant RAN slicing KPIs
State (s_{t+1})
State (s_t)
Action (A_t)
Reward (R_{t+1})
DRL-based Intelligent RAN Slicing Controller
Select action with exploration probability (ε)
Update value function

Fig. 2. RL-based slicing controller-environment interaction.

types of ML is that it evaluates the actions taken rather than specifying correct actions [14]. RL does not require complete knowledge of the RAN system or prior knowledge of the network. Both requirements are inefficient and infeasible for stochastic environments such as RANs in NGNs. Thus, RL is an attractive approach to solve the resource allocation problem in RAN slicing [15], [16].

An RL-based RAN slicing controller typically interacts with the RAN environment bidirectionally as seen in Fig. 2. At any given slicing time step, an RL agent observes the RAN system state and chooses an action to take, i.e., resource allocation for each slice. The action taken changes the RAN environment in a way and the RL agent receives feedback in terms of a reward value that represents the system performance.

The RL agent aims at maximizing the reward feedback that it gets from its interaction with the NGNs RAN system. The reward function is designed by network experts to guide the RL agent's search for the optimal policy, π^*. It is often represented in terms of a weighted sum of the relevant network's key performance indicators (KPIs). This way, the RL-based RAN slicing controller indicates how good the action taken was. This is estimated based on the agent's sampled experience from interacting with the RAN environment in a real-time and dynamic-open control fashion. In this study, we design a sigmoid-based reward function to control the effect of getting closer to the minimum acceptable threshold of each slice's SLAs.

C. Transfer Learning-Accelerated RL-Based RAN Slicing

Slow convergence of RL algorithms is a challenge that relates to the number of learning steps needed to find a good set of radio resource allocation configurations given the various system states. This can happen while training, when the agents are newly deployed in a live network, and when the network's context changes significantly [8]. The RL agent needs to observe a representative variety of the RAN system's possible states several times. The learning happens by iteratively updating a value function until convergence. This process is referred to as the exploration phase. The value function gives an estimate of the expected return if the agent starts in a given state or state-action pair, and then acts according to a particular policy.

TL expedites the learning of new target tasks by exploiting knowledge from related source tasks [17]. This can shorten the learning time of ML algorithms and enhance their robustness to changes in wireless environments. TL is widely used in image object classification, where pre-trained top-performing models are used as the basis for image recognition and related computer vision tasks. This includes but is not limited to, initializing an artificial neural network (ANN) with the architecture and weights from such pre-trained models. This is done to accelerate the training of an object classifier using a local dataset that might include a different set of objects. TL techniques have recently emerged as potential solutions to RL practical challenges such as the long exploration phase in the constantly changing wireless environments [9].

TL in RL is further categorized based on the knowledge being transferred, and when and how to transfer such knowledge.

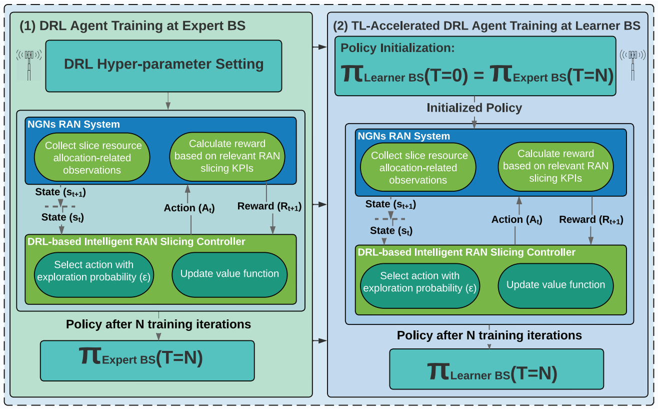

(1) DRL Agent Training at Expert BS
DRL Hyper-parameter Setting
NGNs RAN System
Collect slice resource allocation-related observations
Calculate reward based on relevant RAN slicing KPIs
State (s_{t+1})
State (s_t)
Action (A_t)
Reward (R_{t+1})
DRL-based Intelligent RAN Slicing Controller
Select action with exploration probability (ε)
Update value function
Policy after N training iterations
$\pi_{\text{Expert BS}}(T=N)$
(2) TL-Accelerated DRL Agent Training at Learner BS
Policy Initialization:
$\pi_{\text{Learner BS}}(T=0) = \pi_{\text{Expert BS}}(T=N)$
Initialized Policy
NGNs RAN System
Collect slice resource allocation-related observations
Calculate reward based on relevant RAN slicing KPIs
State (s_{t+1})
State (s_t)
Action (A_t)
Reward (R_{t+1})
DRL-based Intelligent RAN Slicing Controller
Select action with exploration probability (ε)
Update value function
Policy after N training iterations
$\pi_{\text{Learner BS}}(T=N)$

Fig. 3. Example of policy reuse to accelerate RL-based RAN slicing.

TABLE I
LIST OF SYMBOLS

Symbol	Definition
π	RL agent's policy
N	Number of RL agent's learning iterations until convergence
S	Number of slices sharing the available bandwidth
B	Available bandwidth shared among slices
$x(a)$	An instance of slicing PRB allocation configuration
X	Possible slicing PRB allocation configuration
L	Inverse form of latency of the available slices
α	Importance of the latency for each slice
$\theta(t)$	System state at time t
b_i	Bandwidth allocated to slice i
d	Contribution to traffic load within a time window for each slice
$\mathbb{E}(\cdot)$	Expectation of the argument
w_i	Reward function weight for slice i
r^2	Coefficient of determination
$SS_{\textbf{residual}}$	Residual sum of squares
$SS_{\textbf{total}}$	Total sum of squares associated with the outcome variable
$\bar{y}_i$	Predicted values
y_i	Ground truth values
n	Number of observations in supervised learning training

Policy transfer is a class of TL in which a source policy is transferred to an agent with a similar target task [9]. In policy reuse, one of the policy transfer sub-categories, a source policy that is learned by an expert agent is directly reused to guide the target policy of a learner agent [9]. The expert RL agent learns until converging to a good policy, while the learner RL agent reuses the expert policy, π_{Expert} to tackle the practical challenge of RL slow convergence. This can be configured in different ways. As shown in Fig. 3, this could be done by initializing the target policy at a learner base station (BS) with the learned source policy from an expert BS as follows [10]:

$$\Pi_{\text{Learner BS}}(t = 0) = \Pi_{\text{Expert BS}}(t = N) \tag{1}$$

where N is the number of learning iterations carried out by the RL agent at an expert BS until convergence as defined in Table I.

III. RELATED WORK

TL has been gradually used in the wireless networks domain. This includes but is not limited to, BSs switching, indoor localization, and intrusion detection [17]. However, most of these studies mainly use the supervised setting of TL. More recently, some wireless network researchers started to employ TL to accelerate the learning process in their RL-based solutions [8], [9]. The authors of [18] proposed two Q-learning-based techniques to address interference mitigation in an mm-Wave network with beamforming and NOMA. One of these techniques employs TL to speed up RL convergence. The expert agent is trained to solve a user-cell association problem. Then, a learner agent uses such knowledge to solve joint user-cell association and selection of number of beams problem to cover the associated users. Furthermore, the work in [19] applies TL to make the system more prone to variation in network status and topology and to improve the training process efficiency. The authors employ generative adversarial networks (GANs) to capture unchanging features in different network environments and utilize them to accelerate the training process.

Moreover, the work presented in [20] proposed deep transfer RL-based joint radio and cache resource allocation. The authors reported that the proposed approach resulted in better network performance and faster convergence speeds. Furthermore, the authors of [21] developed a TL mechanism to enable aerial vehicles (AVs) to exploit valuable experiences. This helps in accelerating the training process when the AVs move to a previously unseen environment. Additionally, the authors of [22] discuss the idea of changing the parameters of a reward function to balance the QoS requirements of users and system energy consumption. The authors combine relational DRL with TL to address the insufficient generalization ability and the slow recovery when exposed to new conditions. Finally, it is worth noting that other researchers address the issue of RL slow convergence using approaches such as heuristics and meta-learning [23], [24], [25], [26]. However, the focus of this paper is the efficient use of TL as it is one of the commonly used approaches in the wireless networks' domain [9].

The reviewed studies mainly use TL to accelerate the learning process of RL-based RRM-related solutions without paying attention to the efficiency of the acceleration process. Unlike these studies, we focus on analyzing and enhancing the efficiency of the TL-based acceleration of RL convergence in RAN slicing. We propose a novel *predictive* approach that selects the best-saved policy out of several stored policies for more efficient TL-based acceleration. We also consider a vital deployment scenario of the O-RAN paradigm where MNOs can change the priorities of fulfilling the available slices' SLAs. These priorities are mostly assumed as constants when considered in other studies [27]. To the best of our knowledge, this is the first research study to address the abovementioned aspects in the context of RAN slicing in next-generation wireless networks. The authors of the reviewed studies can, for instance, revisit their work using our proposed modules to account for efficient TL-based acceleration and to enable reward function weight change by MNOs.

IV. POLICY TRANSFER FOR ACCELERATING RL-BASED 6G RAN SLICING

A. System Model

As mentioned in Section II-A, resource management for network slicing can be considered from several perspectives [11]. In this paper, we focus on the downlink case of the radio access part, and more specifically the RAN slicing

TABLE II
SIMULATION PARAMETERS AND RL AGENT DESIGN DETAILS

(a) RAN Slicing Simulation Parameter Settings			
	Video	**VoLTE**	**URLLC**
Scheduling Algorithm	Round Robin		
Bandwidth Allocation Window Size	40 scheduling time slots		
Packet Interarrival Time Distribution (Expert and Learner Agents)	Truncated Pareto (Mean = 6 ms, Max = 12.5 ms)	Uniform (Min = 0, Max = 160 ms)	Exponential (Mean = 180 ms)
Packet Size Distribution (Expert and Learner Agents)	Truncated Pareto (Mean = 100 Byte, Max = 250 Byte)	Constant (40 Byte)	Truncated Lognormal (Mean = 2 MB, Standard Deviation = 0.722 MB, Max = 5 MB)

(b) RAN Slicing RL Design		
State		The contribution to traffic load within a specific time window for each slice $(d_{\text{VoLTE}}, d_{\text{URLLC}}, d_{\text{Video}})$
Action		Bandwidth allocated to each slice (15 allocation configurations) $(b_{\text{VoLTE}}, b_{\text{URLLC}}, b_{\text{Video}})$, s.t. $b_{\text{VoLTE}} + b_{\text{URLLC}} + b_{\text{Video}} = B$
Reward		A weighted sum of an inverse form of latency experienced in a slicing window by the various slices
RL Parameters	**RL Algorithms**	Q-Learning and SARSA
	Total Number of Time Steps	20,000 per simulation run
	Number of Expert Agents	16 reward function weight combinations
	Number of Learner Agents	3392 (106 weight combinations, each accelerated using all expert policies)
	Epsilon	Expert Agent: 1, Learner Agent: 0.1
	Epsilon Decay	Expert Agent: 0.9, Learner Agent: 0.5 (every 100 steps)
	Learning Rate (alpha)	Expert Agent: 0.1, Learner Agent: 0.5

resource allocation problem. The main goal is to allocate the limited PRBs to the available slices, maintaining an acceptable spectral efficiency (SE) while keeping an acceptable delay, and generally, quality of experience (QoE) satisfaction. Given the list of symbols in Table I, the slice resource allocation problem can be mathematically formulated as follows [10], [28].

There exists a set of S slices that share the available bandwidth B. A parameter that controls the number of PRBs allocated to each slice needs to be optimized for each slice. This can be described by the vector $x \in \mathbb{R}^{\text{S}}$. At a given instance, a RAN slicing controller decides to choose a specific slicing PRB allocation configuration, i.e., $x(a)$, out of the X possible configurations, where $a = 1, 2, 3, \ldots, X$. Based on such a decision, the system performance is affected. For the purpose of this paper, the system performance is represented in terms of the latency of the admitted slices and can be represented by a single value as follows:

$$f(x(a), \theta(t)) = \alpha L \in \mathbb{R}, \tag{2}$$

where L is a function that represents an inverse form of the latency of the available slices, while α represents the importance of the latency for each slice. Moreover, $\theta(t)$ is the system state at time t. This function is unknown to the controller, therefore it can not explicitly relate input to output and can only observe the function's outcome. The system state can be represented by the traffic load, the channel quality, or other external factors that might affect the RAN system performance. The majority of these variables evolve in a way that is hard to infer theoretically, especially in time scales of seconds or shorter. The RAN slicing controller explores different slice allocation configurations and observes the corresponding system performance in search of the optimal configuration that maximizes the performance, i.e.,

$$\hat{x} = \underset{x}{\operatorname{argmax}}\ f(x) \tag{3}$$

B. Mapping to Reinforcement Learning

Based on the model defined in Section IV-A, an RL agent would take an action at the beginning of each slicing window to decide the PRB allocation for each slice; $b = (b_1, \ldots, b_S)$, subject to $b_1 + \cdots + b_S = B$. Such action is taken based on the observed system state, defined in this paper as the contribution to traffic load within a specific time window for each slice, $d = (d_1, \ldots, d_S)$. We define the reward function as the weighted sum of an inverse form of latency as detailed in Section IV-E1b. The goal is to maximize the long-term reward expectation,

$$\mathbb{E}\{f(x(a), \theta(t))\}, \tag{4}$$

where the notation $\mathbb{E}(\cdot)$ is the expectation of the argument, that is,

$$\begin{aligned} \underset{x}{\operatorname{argmax}}\ \mathbb{E}\{f(x(a), \theta(t))\} &= \underset{x}{\operatorname{argmax}}\ \mathbb{E}\{\alpha L(x(a), \theta(t))\} \\ &= \underset{b}{\operatorname{argmax}}\ \mathbb{E}\{R(b, d)\} \end{aligned} \tag{5}$$

This allows the agent to learn a policy, π, that takes a state d as input and outputs an action, $b = \pi(d) \in A$, to maximize reward, R. The key challenge to solve (5) lies in the time-varying demand in terms of traffic models and the number of users for each service type. The optimal solution for the problem can be precisely calculated by carrying out an exhaustive search. In such a case, all the possible allocations should be considered at the beginning of every slicing window and the corresponding system performance should be noted. This approach, however, is computationally expensive and practically infeasible. Hence, RL is a good alternative to solve the problem. The RAN slicing RL design parameters are highlighted in Table II-(b).

TABLE III
REWARD FUNCTION WEIGHT COMBINATIONS OF EXPERT AND LEARNER AGENTS

	Format: $[w_{\text{VoLTE}}, w_{\text{URLLC}}, w_{\text{Video}}]$
Expert Agents Reward Function Weights	[0.1, 0.8, 0.1], [0.1, 0.1, 0.8], [0.1, 0.45, 0.45], [0.1, 0.7, 0.2], [0.1, 0.2, 0.7], [0.1, 0.6, 0.3], [0.1, 0.3, 0.6], [0.1, 0.5, 0.4], [0.1, 0.4, 0.5], [0.3333, 0.3333, 0.3333], [0.8, 0.1, 0.1], [0.4, 0.5, 0.1], [0.4, 0.4, 0.2], [0.4, 0.1, 0.5], [0.4, 0.2, 0.4], [0.4, 0.3, 0.3]
Learner Agents Reward Function Weights	[0.1, 0.1, 0.8], [0.1, 0.8, 0.1], [0.8, 0.1, 0.1], [0.6, 0.2, 0.2], [0.2, 0.6, 0.2], [0.2, 0.2, 0.6], [0.4, 0.3, 0.3], [0.3, 0.4, 0.3], [0.3, 0.3, 0.4], [0.2, 0.4, 0.4], [0.4, 0.2, 0.4], [0.4, 0.4, 0.2], [0.1, 0.2, 0.7], [0.1, 0.7, 0.2], [0.2, 0.1, 0.7], [0.2, 0.7, 0.1], [0.7, 0.2, 0.1], [0.7, 0.1, 0.2], [0.1, 0.3, 0.6], [0.1, 0.6, 0.3], [0.3, 0.1, 0.6], [0.3, 0.6, 0.1], [0.6, 0.3, 0.1], [0.6, 0.1, 0.3], [0.2, 0.3, 0.5], [0.2, 0.5, 0.3], [0.3, 0.2, 0.5], [0.3, 0.5, 0.2], [0.5, 0.3, 0.2], [0.5, 0.2, 0.3], [0.1, 0.4, 0.5], [0.1, 0.5, 0.4], [0.4, 0.1, 0.5], [0.4, 0.5, 0.1], [0.5, 0.4, 0.1], [0.5, 0.1, 0.4], [0.1, 0.15, 0.75], [0.1, 0.85, 0.05], [0.8, 0.15, 0.05], [0.6, 0.25, 0.15], [0.2, 0.55, 0.25], [0.2, 0.25, 0.55], [0.4, 0.35, 0.25], [0.3, 0.45, 0.25], [0.3, 0.35, 0.35], [0.2, 0.45, 0.35], [0.4, 0.25, 0.35], [0.4, 0.45, 0.15], [0.1, 0.25, 0.65], [0.1, 0.75, 0.15], [0.2, 0.15, 0.65], [0.2, 0.75, 0.05], [0.7, 0.25, 0.05], [0.7, 0.15, 0.15], [0.1, 0.35, 0.55], [0.1, 0.65, 0.25], [0.3, 0.15, 0.55], [0.3, 0.65, 0.05], [0.6, 0.35, 0.05], [0.6, 0.15, 0.25], [0.2, 0.35, 0.45], [0.2, 0.55, 0.25], [0.3, 0.25, 0.45], [0.3, 0.55, 0.15], [0.5, 0.25, 0.15], [0.5, 0.25, 0.25], [0.1, 0.45, 0.45], [0.1, 0.55, 0.35], [0.4, 0.15, 0.45], [0.4, 0.55, 0.05], [0.5, 0.45, 0.05], [0.5, 0.15, 0.35], [0.15, 0.1, 0.75], [0.15, 0.8, 0.05], [0.85, 0.1, 0.05], [0.65, 0.2, 0.15], [0.25, 0.5, 0.25], [0.25, 0.2, 0.55], [0.45, 0.3, 0.25], [0.35, 0.4, 0.25], [0.35, 0.3, 0.35], [0.25, 0.4, 0.35], [0.45, 0.2, 0.35], [0.45, 0.4, 0.15], [0.15, 0.2, 0.65], [0.15, 0.7, 0.15], [0.25, 0.1, 0.65], [0.25, 0.7, 0.05], [0.75, 0.2, 0.05], [0.75, 0.1, 0.15], [0.15, 0.3, 0.55], [0.15, 0.6, 0.25], [0.35, 0.1, 0.55], [0.35, 0.6, 0.05], [0.65, 0.3, 0.05], [0.65, 0.1, 0.25], [0.25, 0.3, 0.45], [0.25, 0.5, 0.25], [0.35, 0.2, 0.45], [0.35, 0.5, 0.15], [0.55, 0.2, 0.15], [0.55, 0.2, 0.25], [0.15, 0.4, 0.45], [0.15, 0.5, 0.35], [0.45, 0.1, 0.45], [0.45, 0.5, 0.05], [0.55, 0.4, 0.05], [0.55, 0.1, 0.35]

C. Simulation Environment

Reproducing an existing RL-based RAN slicing solution is not straightforward due to the lack of RL-based RRM benchmark environments that can be easily integrated and reused out of the box. Hence, the algorithms and environment implementations will vary. We improved the OpenAI GYM-compatible RL environment that was initially implemented in our previous research study, [10]. The improved implementation of the environment is available on GitHub.[1] This allows further analysis and comparison of the various RL convergence acceleration approaches in RAN slicing.

The simulation environment was changed to support interfaces that enable the MNOs to change the SLA fulfillment priorities of the available slices. This was done by allowing the change of reward function weights. Such interfaces also enable the consultation of supervised models to select the best policy to load given a certain reward function weight vector. With the rise of the O-RAN paradigm, this is a vital scenario that will be more feasible in 6G networks. We also made the following changes:

- We changed the environment state representation to reflect the slices' contribution to the overall traffic load within a previous time window instead of the number of packets.
- We updated the scheduling algorithm to support scheduling multiple transmissions per transmission time interval (TTI) if resources were available (i.e., PRBs).
- A user priority mode was added as a scheduling setting so that if on, the transmissions belonging to one user are given priority within the same TTI if resources were available.
- We updated the environment so that the unsatisfied users who have multiple unfulfilled transmission requests leave the system.
- The reward function was updated to reflect more control over the effect of getting closer to the minimum acceptable threshold of each slice's SLAs.

D. Transfer Learning Evaluation Setup

We conducted an exhaustive experiment to investigate the transferability of various expert policies when accelerating RL-based RAN slicing via policy reuse. We mainly focused on the scenario when an MNO needs to change the priorities of the various slices, and hence, the weights of the reward function. To do so, we followed a similar acceleration approach to the one mentioned in Section II-C but to extensively study the reward convergence behaviour of the accelerated learner agents. This constitutes a step towards a more efficient way for acceleration when the context changes. It provides insights into how to choose an expert policy to use for acceleration when an MNO decides to change the slices' priorities. We started with training and saving 16 basic models, namely the expert models, using a limited number of reward function weight-combinations as seen in Table III. The policies of each of these trained models are then reused to initialize the policies of 106 learner agents reflecting 106 different reward function weight combinations as defined in the table. The evaluation process includes accelerating 3392 RL learner agents in total via policy reuse. We used two RL algorithms, and hence, we employed policy transfer to accelerate 1696 agents per RL algorithm.

We use settings that are known to be used in slicing-related studies for better interpretability of TL efficiency results. Moreover, we run a large number of simulations to be confident about the generality of our analysis and approach as described in the next sections. The round-robin algorithm is one of the common scheduling algorithms that ensure fairness [16]. We used it as the scheduling algorithm per slice similar to the case in [29]. We simulated a scenario with three types of services; voice over LTE (VoLTE), video, and ultra-reliable low-latency communications (URLLC). The prevalent 4G networks mainly classify services into voice and best effort, hence it is hard to have access to live network traces of the services addressed in this paper. Therefore, we decided to use traffic that follows known mathematical models similar to those in [29].

User requests are generated based on the distributions shown in Table II-(a). In such cases, URLLC users generate the largest, but the least frequent packets compared with users of the other services. VoLTE users generate the smallest packets, while video packets are the most frequently generated ones. Users belonging to the same slice share bandwidth equally.

[1] Available at http://www.github.com/ahmadnagib/TL4RL.

More specifically, the round-robin scheduler is used within each slice at the granularity of 0.5 ms. Moreover, the slicing window size is 40 scheduling time slots. In other words, the RL agent takes an action to adjust the PRB allocation to each slice every 20 ms. We summarize the parameters used to create the environment and train various RL agents in Table II-(b).

We used two RL algorithms to train and accelerate the RL-based RAN slicing agents, namely Q-Learning and state-action-reward-state-action (SARSA). The implementation of both is available on GitHub.[2] The implementation was adopted and modified to accommodate the acceleration process using the policy reuse approach of TL. The trained expert policy is loaded to initialize the learner agent's policy. Moreover, a decaying epsilon greedy is incorporated to better control the exploration-exploitation behaviour of the RL agent. Finally, the code was modified to log the various learning steps' relevant information for better tracing and debugging. The two aforementioned RL algorithms are used to decide the percentage of PRBs to be allocated to each slice. Afterward, round-robin scheduling is followed within each slice. The non-accelerated agents were compared against their accelerated counterparts having the same reward function weights. The accelerated agents were guided by policies from the 16 saved basic models. Each one of these basic expert models has a different reward function weight vector.

E. Reinforcement Learning for Network Slicing

We use the RL mapping defined in Section IV-B for both the expert and learner agents. The objectives of both the expert and the learner agents are the same. As seen in Table II-(b), we use 106 reward function weight combinations to simulate a wide range of possible MNO configurations of the slices' priorities. This enables us to observe the reward function weight vectors of the saved expert policies and study their effect on the learner agents' reward convergence, and hence, acceleration. The 3392 simulation runs executed allowed us to assess our proposed approach's capacity for generalization.

It is expected that the reward distribution of the learner agents will vary from the expert agents as the reward function weights are changed intentionally by the MNO. This can happen when a new RL agent is deployed from scratch at a BS. This can also happen when the MNO decides to change the slices' priorities by changing the reward function weights after following another set of weights for a given time. In this paper, we present a RAN slicing scenario in which the MNO reconfigures the weights of the RL reward function. This scenario happens after deploying the RL agent in a live network. After reconfiguring the reward function weights, the policy that has been followed by the RL agent can lead to non-optimal resource allocation actions. Hence, if the RL agent does not recover quickly to an optimal or a near-optimal policy, this will lead to taking non-optimal actions for a long duration.

In this study, we give more weight to latency in deciding the system's performance, and hence the reward function is purely represented in terms of latency for better TL efficiency results' interpretability. As a result, such non-optimal actions taken by the agent will lead to non-optimal latency performance. Therefore, the RL agent's recovery to an optimal policy after the changes made by an MNO needs to be as fast as possible. It is worth noting that the issue of slow convergence of RL agents is not directly related to the latency defined in the reward function. However, the agent's convergence to the optimal solution will lead to the best performance in terms of slices' latency. Accordingly, the faster the agent's convergence, the better the slices' latency performance. We also assume that the policies learned by the expert agents are saved and can be loaded by a learner agent before the latter starts the exploration process.

1) Expert and Learner Agents Settings:

a) Reinforcement learning agents: We decided to employ the policy learned by the expert agents to be later transferred to initialize all the learner agents' policies. In the case of Q-Learning and SARSA, this means initializing the Q-tables of the learner agents. Q-learning and its variants are among the most popular RL algorithms that have shown impressive results in RL-based RRM-related studies [4]. It gained popularity as it allowed the development of an off-policy temporal difference (TD) algorithm. It is sample efficient, and any policy can be used to generate experience. However, Q-learning variants still lack convergence guarantees for non-linear function approximators. Hence there is still room for improving the convergence performance using approaches such as policy transfer. Our framework supports any other RL algorithms via OpenAI Gym standardized interfaces [30]. For instance, we also used SARSA as it has a different value function update procedure. This allowed us to examine the effect of using a different algorithm on the acceleration process.

We mainly focus on analyzing and enhancing the acceleration process when using different reward function weights. Hence, when we configured the RL settings, we intended to have a configuration that converges to the optimal solution for all the non-accelerated agents. The expert and learner agents hyperparameters are shown in Table II. We used smaller values for the exploration rate (epsilon), and larger values for the learning rate when accelerating the learner agents. This setting is used to accelerate the deployed learner agents' adaptation to the new context taking advantage of the knowledge captured by the expert policies.

b) Reward function: We used the reward function stated in Table II. Based on the system performance function defined in Section IV-A, the reward function was improved to reflect more control over the effect of getting closer to the minimum acceptable threshold of each slice's SLAs as follows:

$$
\begin{aligned}
R &= \sum_{s=1}^{\|S\|} w_s * \frac{1}{1 + e^{c1_s * (l_s - c2_s)}}, \text{that is,} \\
R &= w_{\text{VoLTE}} * \frac{1}{1 + e^{c1_{\text{VoLTE}} * (l - c2_{\text{VoLTE}})}} \\
&\quad + w_{\text{URLLC}} * \frac{1}{1 + e^{c1_{\text{URLLC}} * (l - c2_{\text{URLLC}})}} \\
&\quad + w_{\text{Video}} * \frac{1}{1 + e^{c1_{\text{Video}} * (l - c2_{\text{Video}})}} \qquad (6)
\end{aligned}
$$

[2] Available at https://github.com/dennybritz/reinforcement-learning.

We designed sigmoid function-based [31] rewards to fulfill that purpose. In this study, we focus on the delay requirements, so we used latency as a variable. Such a reward function penalizes actions that come close to violating slices' latency requirements. Two parameters, $c1$ and $c2$, are configured to tune the shape of the function. $c1$ reflects the point from where the slope of the sigmoid function begins to change for the first time. This defines when to start penalizing the agent's actions. While $c2$ represents the inflection point, i.e., the lowest acceptable delay performance for each slice based on the slice's SLAs. We use different, but constant, $c1$ and $c2$ values for each slice type.

The weights of the reward function are adjustable to allow the MNOs to prioritize some slices over others. This will lead to a change in context and the actions taken based on the policy at hand may lead to extreme performance drops. We explore the effect of accelerating the agent with knowledge from already trained agents, namely expert agents, having different slice delay weight combinations in their reward function when trained. Table III lists the base weights used in the expert agents' reward functions and those of the learner agents to be accelerated by the expert policies.

c) Traffic load model: We generated one traffic model per service for the expert and the learner agents as seen in Table II-(a). It is represented in terms of inter-arrival times, and packet sizes.

V. Predictive Policy Transfer for Accelerating RL-Based RAN Slicing

A. Network Slicing SLAs and Weights of RL Reward Function

It is practically vital for an MNO to have the ability to tune the weights of an RL agent's reward function. This enables the MNO to change the priority of fulfilling the SLAs of the admitted network slices. This is important as different services might have similar traffic patterns but significantly different network requirements. For instance, two massive machine-type communications (mMTC) deployments may have the same traffic pattern. However, each of them can have a significantly different latency tolerance depending on the exact application. If both slices are treated equally as generic mMTC slices in the RL reward function, this will lead to unnecessary over-provisioning of resources to the more delay-tolerant slice. This would also lead to SLA violations in the less delay-tolerant slice when resources are scarce. Enabling MNOs to modify the slices' weights in the reward function allows for more efficient use of the available spectrum and results in fewer SLA violations and consequently fewer monetary penalties. However, changing such weights can drastically change the system's performance. Hence, MNOs need an efficient way to accelerate the RL agent so that it quickly recovers to a good policy.

B. Proposed Approach

Based on the analysis results, we propose a data-driven novel approach to select the expert policy with the least expected reward convergence error to be used to accelerate a learner agent. We specifically build ML models based on the expert and learner agents' weight vectors and compare their performance to that of a distance metric. These trained models predict the expected convergence error of accelerating a given learner agent using a certain expert policy when the context changes. Having such a model will allow the MNOs to estimate the expected convergence error when reusing each of the saved expert policies. Hence, they can choose the one with the minimum expected error to load for guiding the learner agent of interest, allowing more efficient policy reuse acceleration. We specifically propose the following procedure:

1) First, an MNO chooses a group of expert policies that have the minimum relative convergence error and store them.
2) Before the MNO decides to change the slices' weights, in other words, slices' delay SLAs fulfillment priorities, the switch to the new reward function weights should be scheduled.
3) Scheduling such a change should trigger an automated process. Such a process should employ an ML-based model previously trained using data collected via O-RAN interfaces. The model predicts the expected convergence error of using all the stored expert models to accelerate a learner agent that includes the provided scheduled weights in its reward function.
4) Based on the predicted errors, the stored expert policy with the least error will be chosen.
5) At the scheduled weight change time, the chosen expert policy should be loaded to initialize the policy of the learner agent and guide the exploration phase as soon as the context changes.
6) Concept drift [32] can happen if the network conditions are significantly different from those experienced when training the prediction model. Hence, it is important to update the model whenever any drifts are detected. The performance of the various expert policies concerning guiding the learner agents should be regularly logged. This info should serve as feedback to be used to update the ML-based models for up-to-date reward convergence error predictions.

C. Models and Evaluation Metrics

As part of this paper, we propose an approach to choose the policy to load when an MNO changes the reward function weight vector. To do so, we explore three different types of models. The first uses a simple metric, the second is based on a traditional ML approach, and the third is based on a deep learning approach. This allows us to analyze how a representative sample from each of these three categories performs. This also enables us to observe whether a simple metric can get a performance that is comparable to the more sophisticated ML approaches.

In the first model, we use a simple Euclidean distance measure as a baseline. In such a case, the distances between the weight vectors of the learner agent of interest and all the stored expert policies are calculated, and hence, the one with the least distance is chosen. We then investigate whether we can get a relatively higher accuracy for choosing the best-saved policy. Thus, we proposed to leverage the embedded intelligence of the O-RAN and make use of ML approaches to predict the expert policies' convergence errors and choose the best policy.

For that purpose, we propose the extremely randomized trees (extra-trees) regressor as the traditional ML method. Extra-trees regressor is based on the commonly used, and flexible, decision trees. It is an ensemble learning method that is faster to train than methods such as random forest while maintaining comparable accuracy [33]. We also employ automated machine learning (AutoML) [34] to search for a good-performing traditional ML approach. We used an AutoML tool called H2O [35]. Such a tool trains several model types, and several hyperparameter settings for each model type. Extra-trees regressor was among the well-performing models trained by H2O. Thus, H2O allowed us to further optimize the extra-trees regressor model with the goal of minimizing the regression error.

Finally, we propose to train a multilayer perceptron (MLP) model to reflect one of the potential deep learning architectures. MLP is one of the common data-driven approaches used in supervised learning tasks given multi-attribute network data [36]. Our implemented framework supports the adoption of and comparison against other approaches. This can be other deep learning approaches such as convolutional neural networks (CNNs), other traditional ML approaches, or even non-ML approaches. Using other ML approaches and other hyperparameter settings will lead to a different prediction performance in terms of regression error for instance. Since we propose to load the policy with the least expected error, this will affect the accuracy of our proposed ML-based *predictive* policy transfer approach. Hence, a model that combines a low root mean squared error (RMSE), and a high coefficient of determination (r^2 score), as defined later in this section, is expected to have the highest expert policy prediction accuracy. Such a model should always be selected to be used for reward convergence error prediction. Exhaustively comparing the various ML-based methods for predicting the convergence error is an interesting area to explore. However, it is not the focus of this paper. We specified multiple values for the different parameters of the trained MLP and applied a grid search through them. The best hyperparameter settings among the ones tested are shown in Table IV.

The following metrics are used to evaluate the performance of the built ML models [37]. The test split of the dataset is used to compare the model's predictions against the ground truth to calculate both the r^2 scores and the *RMSE*. Additionally, the accuracy of choosing the expert policy with the least reward convergence error is also evaluated.

1) Coefficient of Determination: r^2 score indicates the proportion of the variance in convergence error that is explained by the model. It is normally a number between zero and one. The closer the value to one, the better the performance of the regression model. r^2 score can be calculated as follows:

$$r^2 = 1 - \frac{SS_{\text{residual}}}{SS_{\text{total}}} \quad (7)$$

TABLE IV
MULTILAYER PERCEPTRON HYPERPARAMETERS

	Multilayer Perceptron (MLP)
Number of Layers	2 dense hidden layers
Number of Units	First hidden layer: 6, Second hidden layer: 3
Loss Function	Mean Squared Error (MSE)
Optimizer	Adam
Activation Function	Hidden layers: ReLU, Output layer: Linear
Learning Rate	0.001
Batch Size	25
Number of Epochs	300 for each run
Pre-processing	Standardization
Validation Split	20% of the training data
Test Split	10% of the dataset, 10-fold cross validation

where SS_{residual} is the residual sum of squares, and SS_{total} is the total sum of squares associated with the outcome variable.

2) Root Mean Square Error (RMSE): RMSE is a measure of accuracy used to compare prediction errors of different regression models for a particular dataset. RMSE represents the quadratic mean of the differences between predicted values and observed values. In general, a lower RMSE is better, and it can be calculated as follows:

$$RMSE = \sqrt{\left(\frac{1}{n}\right)\sum_{i=1}^{n}(\bar{y}_i - y_i)^2} \quad (8)$$

where $\bar{y}_i$ represents the predicted values, while y_i represents the ground truth values, and *n* is the number of observations.

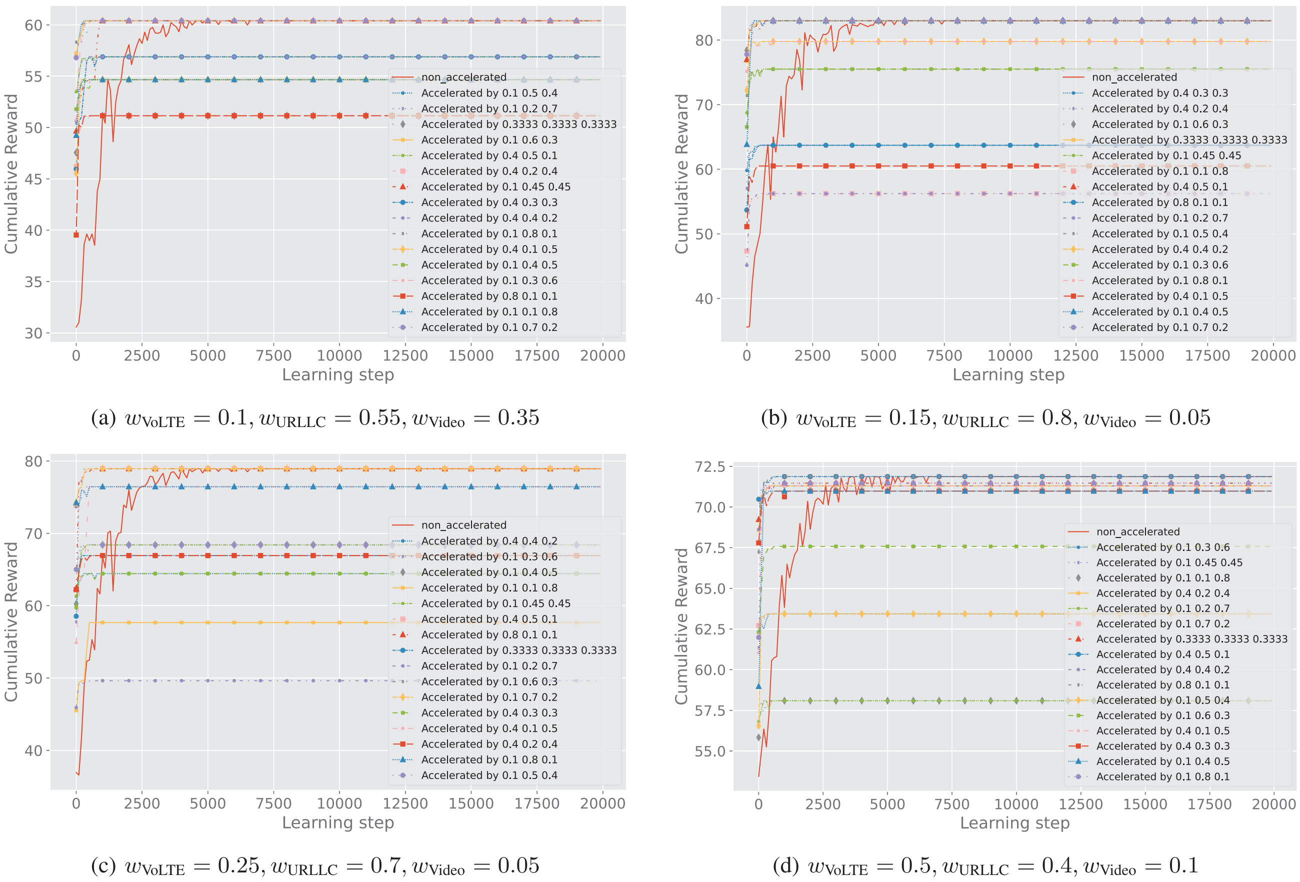
3) Expert Policy Prediction Accuracy: The built models are then used to predict the reward convergence error for the stored expert policies given a certain learner agent's reward function weight vector. Hence, the expert policy with the least predicted convergence error is selected for acceleration via policy reuse. The accuracy of such selection is the main concern of our work. It constitutes a significant step towards having more efficient acceleration. We also compare the accuracy of the ML-based models to that of the simpler Euclidean distance baseline.

All the experiments carried out in this study including the ones for building the ML models were carried out on a Linux machine having 8 CPUs, 64 GB of RAM, and NVIDIA GeForce RTX 2080Ti GPU. Keras, with TensorFlow as the backend, and sklearn are the Python packages used to implement the deep learning models and the extra-tree regressors respectively.

VI. RESULTS

A. Reward Convergence Behaviour

We first explored the reward convergence behaviour of the learner agents when policy transfer is employed to initialize the policy of a learner agent using the saved expert policies. Fig. 4 shows such behaviour for 4 samples out of the 106 learner agent contexts when accelerated using all the expert policies. All the sub-figures also show convergence behaviour when the learner agent is left to learn from scratch without any guidance from the expert policies. It is evident from the figures and the statistics compiled in Table V that the non-accelerated versions of the learner agents need more learning steps to



(a) $w_{\text{VoLTE}} = 0.1, w_{\text{URLLC}} = 0.55, w_{\text{Video}} = 0.35$

(b) $w_{\text{VoLTE}} = 0.15, w_{\text{URLLC}} = 0.8, w_{\text{Video}} = 0.05$

(c) $w_{\text{VoLTE}} = 0.25, w_{\text{URLLC}} = 0.7, w_{\text{Video}} = 0.05$

(d) $w_{\text{VoLTE}} = 0.5, w_{\text{URLLC}} = 0.4, w_{\text{Video}} = 0.1$

Fig. 4. Cumulative reward over 100 learning steps for 4 different contexts, i.e., different slice weight combinations.

TABLE V
TRANSFER LEARNING ACCELERATION STATISTICS

	Max	Min	Mean	Median
Number of expert models that resulted in better convergence per leaner agent (out of 16 expert models)	16	1	7	7
Number of learner agents improved by a certain expert model (out of 106 learner agents)	76	24	47	49
Number of learning steps reduced	14000	4500	7377	6900

converge compared with their accelerated counterparts. The non-accelerated version, visualized in solid lines in all four figures, takes an average of 7377 learning steps more than its accelerated counterparts to converge to the optimal reward. Some of the accelerated versions can take up to 14000 steps less than their non-accelerated counterparts to converge.

Moreover, the statistics suggest that at least one of the 16 expert models improved convergence when used to accelerate the 106 learner agents compared with the non-accelerated versions. This is mainly due to two reasons, the first is that the accelerated versions make use of the existing knowledge of the expert agents. The second is that the accelerated versions are configured to minimize exploration. This is necessary, as the context of the network changes, to avoid any rapid drops and instabilities in the system performance. However, as shown in Fig. 4, and mainly due to such restriction on exploration, several accelerated learner agents could not converge to the optimal reward. The error in the reward convergence value can reach 30 as seen in Fig. 4c where the agent accelerated by the expert policy of weights: $w_{\text{VoLTE}} = 0.1, w_{\text{URLLC}} = 0.2, w_{\text{Video}} = 0.7$ converged to a local maximum of almost 50 instead of 80.

We meant to use the same initial RL configurations for all the trained agents to highlight the importance of the RL hyperparameter tuning. This includes the learning rate, the exploration rate, the exploration decay factor, and others. The results indicate the sensitivity and importance of hyperparameter optimization in accelerating and stabilizing the agents' learning process. The process of hyperparameter setting is time-consuming. Even the automation of such a process is not straightforward and is computationally expensive. This requires an efficient way to choose the best policy to guide any given learner agent when the context, such as the slices' priorities, changes without having to deal with the costly online hyperparameter tuning process. More interestingly, Table V suggests that every expert policy out of the 16 saved ones failed to improve the acceleration of, at least, 30 of the learner agents. This is also evident in Fig. 4 when compared with their non-accelerated counterparts. The results reemphasize the importance of having an accurate expert model selection approach to guarantee a better RL exploration performance.

It can be noticed from Fig. 4a, 4b, 4c, and 4d that the difference between the weight vectors of the learner and the expert agents may have an impact on the accelerated agent's reward convergence behaviour. For instance, in Fig. 4c, the learner agent accelerated by the expert policy having weight vector of $w_{\text{VoLTE}} = 0.1, w_{\text{URLLC}} = 0.2, w_{\text{Video}} = 0.7$ has a poor convergence performance. Nonetheless, the learner agent accelerated using expert policy having a weight vector of $w_{\text{VoLTE}} = 0.1, w_{\text{URLLC}} = 0.7, w_{\text{Video}} = 0.2$ has a very good convergence performance. The two expert policies were used to guide the same learner agent having a weight vector of $w_{\text{VoLTE}} = 0.25, w_{\text{URLLC}} = 0.7, w_{\text{Video}} = 0.05$. The main difference, that can be noticed here, is the distance between the weight vectors. The second expert agent's weight vector is much closer to the weight vector of the learner agent. We continue to analyze the correlation between such distance and convergence performance in the next section.

B. Error and Distance Correlation

We also compiled the simulation data and calculated the distance between the weight vectors of the learner and expert agent for each run. We measured the Euclidean distance between two weight vectors as follows:

$$\delta(w1, w2) = \sqrt{\sum_i (w1_i - w2_i)^2} \tag{9}$$

where i is VoLTE, URLLC, and video,

$$\text{and } \sum_i (w1_i - w2_i)^2 = (w1_{\text{VoLTE}} - w2_{\text{VoLTE}})^2 + (w1_{\text{URLLC}} - w2_{\text{URLLC}})^2 + (w1_{\text{Video}} - w2_{\text{Video}})^2$$

We also compared the reward convergence of the accelerated and the non-accelerated agents of each run to calculate the error in reward convergence. This allows us to explore TL's potential to guide the learner agents when the context changes. The error is measured as follows:

$$\xi = max(r_{\text{learner}}) - max(r_{\text{expert}}) \tag{10}$$

where ξ is the error in the convergence reward value, $max(r_{\text{learner}})$ is the optimal convergence reward value of the non-accelerated learner agent, while $max(r_{v expert})$ is the optimal convergence reward value of the agent accelerated using an expert policy. Both values are calculated over 100 learning steps.

The error is correlated to the distance as seen in Fig. 5. The figure shows a scatter plot of the reward convergence error plotted against the distance between the reward function weight vectors of the learner and the expert agents. Each point represents one simulation run where one of the base expert policies is used to initialize the policy of the leaner agent instead of learning from scratch. This is done using both Q-learning and SARSA algorithms. However, the correlation is not linear. Regression estimates are also plotted in the figure for both the Q-learning and SARSA data points. Such an estimate will still lead to large prediction errors if it is purely used to predict the expected error, and hence, deciding which expert policy to reuse, i.e., to maximize the learner agent's reward and minimize the convergence error.

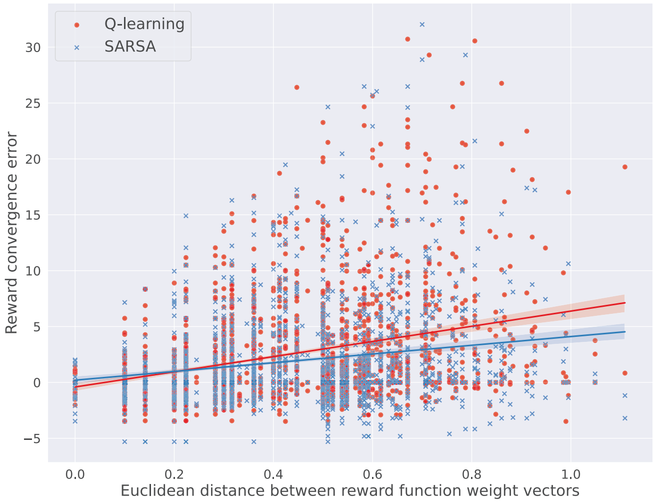


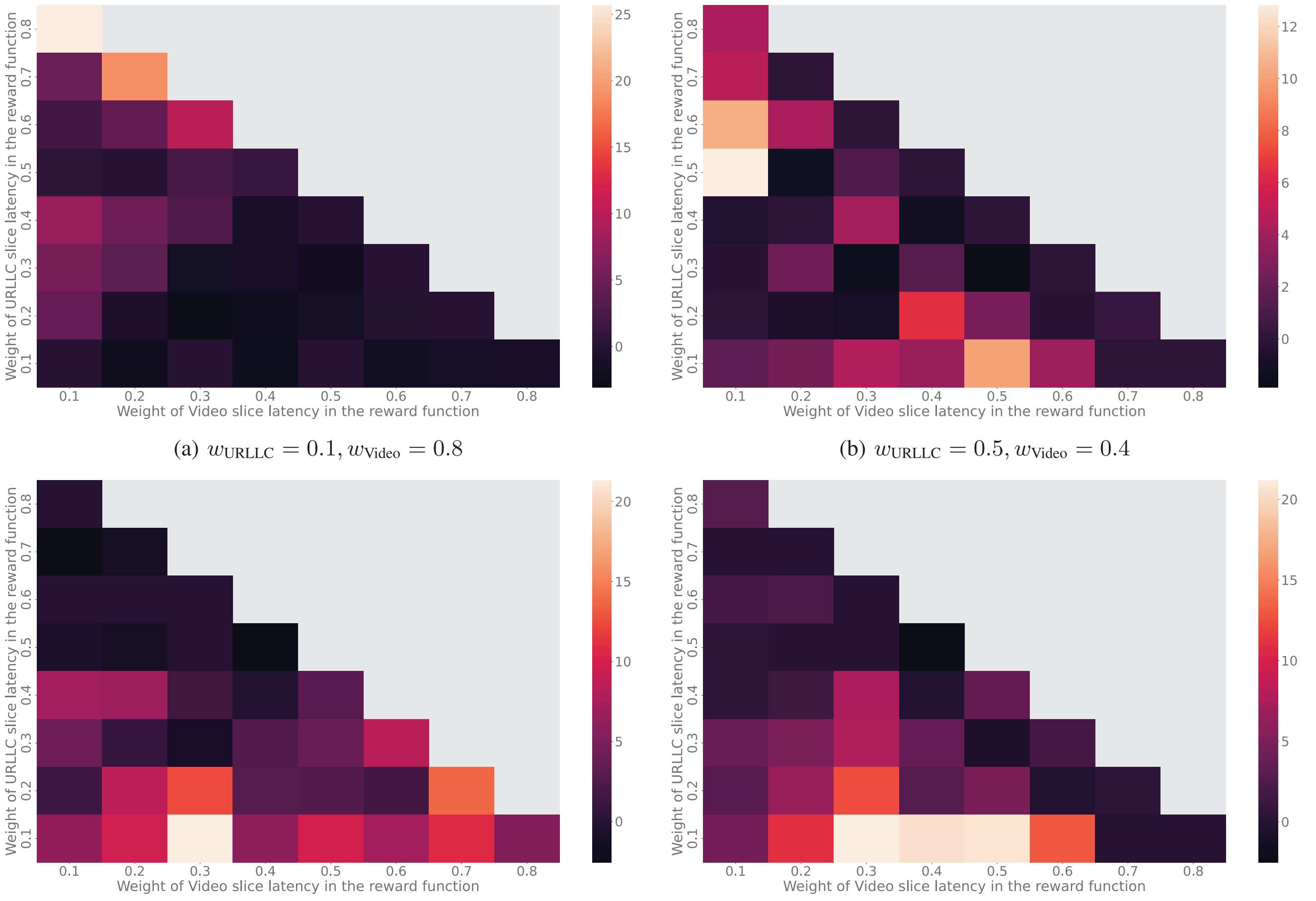
Fig. 5. Scatter plot of reward convergence error and the distance between the weight vectors of expert and learner agents' reward functions.

In Fig. 6, we show two-dimensional heat maps of reward convergence error of 4 different saved expert models when used to accelerate a subset of the learner agents. We focus on the last two weights in the weight vectors, in other words, w_{URLLC} and w_{Video}. It is still evident that the distance between the learner and expert agents' weight vectors affects the convergence error of the learner agents. For instance, Fig. 6a shows the error when the expert model having $w_{\text{URLLC}} = 0.1, w_{\text{Video}} = 0.8$ is used. The learner agent trained using the same reward weight vector is represented by the bottom right square of the heat map. Its colour reflects a small error. Similarly, all the heat map squares close to it seem to have small error values. On the other hand, the square on the top left, i.e. $w_{\text{URLLC}} = 0.8, w_{\text{Video}} = 0.1$, furthest from the $w_{\text{URLLC}} = 0.1, w_{\text{Video}} = 0.8$ square has the highest error. The other figures show similar behaviour concerning the expert and learner agents' location on the heat map grid.

This distance-error correlation is mainly due to the influence of the reward function weights on the performance of the network system and hence the rewards that will be received by the agents. Therefore, with a slight difference in the weight vector, the reward distribution will not be very different. This will allow the learner agent to converge in a few learning steps without having to explore much. However, when the weight difference is big, the Q-tables will be different and exploration is much needed for the learner agent. Given that the learner agents' exploration is restricted to avoid sudden drops in the system performance, they will probably converge to a local maximum in this case.

C. ML-Based Predictive Policy Transfer Performance Discussion

We trained the models mentioned in Section V-C to solve the regression problem of predicting the reward convergence error. We then applied the *predictive* policy transfer procedure



(a) $w_{\text{URLLC}} = 0.1, w_{\text{Video}} = 0.8$

(b) $w_{\text{URLLC}} = 0.5, w_{\text{Video}} = 0.4$

(c) $w_{\text{URLLC}} = 0.7, w_{\text{Video}} = 0.2$

(d) $w_{\text{URLLC}} = 0.8, w_{\text{Video}} = 0.1$

Fig. 6. Heat maps of reward convergence error of different saved expert models used to accelerate learner agents when the context changes, i.e., the MNO changes the slice priority configurations. Each sub-figure reflects a specific expert weight vector.

TABLE VI
ML-BASED MODELS PERFORMANCE EVALUATION

	MLP	Extra-trees regressor	Distance
Average RMSE	3.87	3.62	N/A
Average r^2 score	0.35	0.48	N/A
Expert policy average prediction accuracy	82.80%	90.14%	78.59%
Expert policy minimum prediction accuracy	70.58%	84.52%	64.70%
Expert policy maximum prediction accuracy	89.88%	96.55%	83.90%

proposed in Section V-B to efficiently accelerate RL-based RAN slicing by picking the least error expert policy. Although they are related, we are more concerned with the expert policy prediction accuracy out of the metrics mentioned in Section V-C. This is the main metric that will decide whether a good policy will be reused to guide a learner agent when an MNO changes the weight vector of the reward function.

We followed a 10-fold cross-validation approach and calculated the performance metrics listed in Table VI using both the MLP and the extra-trees regressor models. We also show the accuracy when we purely use the Euclidean distance to decide the best expert policy to load. The distance metric, conforming to our analysis, can still get a very close performance to the other two more complex models. However, we can get a very good accuracy using the extra-trees regressor that mainly fits several randomized decision trees. The MLP had an intermediate performance and needs more expensive hyperparameter tuning to get close to the performance of the extra-tree regressors. In Fig 7, we show the performance of one of the built deep learning models. The plot affirms the relatively lower r^2 values of the trained MLP regression models compared with the extra-tree regressor models as Table VI depicts. Although the high convergence error data points are scarce, the trained models still managed to have a very promising end-to-end expert policy prediction performance.

VII. CONCLUSION AND FUTURE WORK

The work in this paper addresses one of the key practical challenges that are faced when deploying RL-based RRM solutions in dynamic wireless network environments. We conduct an exhaustive experiment to analyze the performance of accelerating RL agents using policy transfer. We mainly consider an O-RAN scenario in which MNOs change the SLAs' fulfillment priorities of the available network slices. It is evident that the distance between the reward function weight vectors of the learner and the expert agents has a significant effect on the reward convergence behaviour. Our analysis

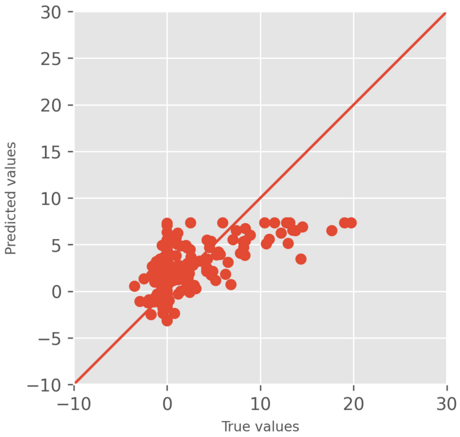


Fig. 7. MLP: actual values vs. predicted values.

shows a high, though non-linear, correlation between the reward convergence error and the Euclidean distance between the weight vectors. We then propose an ML-based mechanism to enhance the acceleration performance of RL-based RAN slicing agents using a novel form of *predictive* TL. This constitutes a key step toward robust intelligent resource management in O-RAN. We argue that it is inevitable to enhance the TL-based acceleration approaches for RL to find its way to RRM commercial solutions.

We plan to acquire and use data traces, and models, to reflect real 6G services traffic such as VR gaming and the metaverse. This will reflect more complex scenarios and more dynamic behaviour. Hence, more sophisticated DRL algorithms may be required while examining the transferability of the trained expert policies. Controlling the inflection point of latency violation in the sigmoid-based reward function and its effect on reward convergence performance is another point to be explored.

References

[1] Z. Xiong, Y. Zhang, D. Niyato, R. Deng, P. Wang, and L.-C. Wang, "Deep reinforcement learning for mobile 5G and beyond: Fundamentals, applications, and challenges," *IEEE Veh. Technol. Mag.*, vol. 14, no. 2, pp. 44–52, Jun. 2019.

[2] R. Shafin, L. Liu, V. Chandrasekhar, H. Chen, J. Reed, and J. C. Zhang, "Artificial intelligence-enabled cellular networks: A critical path to beyond-5G and 6G," *IEEE Wireless Commun.*, vol. 27, no. 2, pp. 212–217, Apr. 2020.

[3] P. H. Masur, J. H. Reed, and N. K. Tripathi, "Artificial intelligence in open-radio access network," *IEEE Aerosp. Electron. Syst. Mag.*, vol. 37, no. 9, pp. 6–15, Sep. 2022.

[4] A. Feriani and E. Hossain, "Single and multi-agent deep reinforcement learning for AI-enabled wireless networks: A tutorial," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 2, pp. 1226–1252, 2nd Quart., 2021.

[5] L. Bonati, S. D'Oro, M. Polese, S. Basagni, and T. Melodia, "Intelligence and learning in O-RAN for data-driven NextG cellular networks," *IEEE Commun. Mag.*, vol. 59, no. 10, pp. 21–27, Oct. 2021.

[6] A. S. Abdalla, P. S. Upadhyaya, V. K. Shah, and V. Marojevic, "Toward next generation open radio access networks: What O-RAN can and cannot do!" *IEEE Netw.*, vol. 36, no. 6, pp. 206–213, Nov./Dec. 2022.

[7] H. Chergui et al., "Zero-touch AI-driven distributed management for energy-efficient 6G massive network slicing," *IEEE Netw.*, vol. 35, no. 6, pp. 43–49, Nov./Dec. 2021.

[8] A. M. Nagib, H. Abou-Zeid, and H. S. Hassanein, "Toward safe and accelerated deep reinforcement learning for next-generation wireless networks," *IEEE Netw.*, early access, Sep. 26, 2022, doi: 10.1109/MNET.106.2100578.

[9] C. T. Nguyen et al., "Transfer learning for wireless networks: A comprehensive survey," *Proc. IEEE*, vol. 110, no. 8, pp. 1073–1115, Aug. 2022.

[10] A. M. Nagib, H. Abou-Zeid, and H. S. Hassanein, "Transfer learning-based accelerated deep reinforcement learning for 5G RAN slicing," in *Proc. IEEE 46th Conf. Local Comput. Netw. (LCN)*, 2021, pp. 249–256.

[11] O. Sallent, J. Perez-Romero, R. Ferrus, and R. Agusti, "On radio access network slicing from a radio resource management perspective," *IEEE Wireless Commun.*, vol. 24, no. 5, pp. 166–174, Oct. 2017.

[12] W. Wu et al., "AI-native network slicing for 6G networks," *IEEE Wireless Commun.*, vol. 29, no. 1, pp. 96–103, Feb. 2022.

[13] Y. Azimi, S. Yousefi, H. Kalbkhani, and T. Kunz, "Applications of machine learning in resource management for RAN-slicing in 5G and beyond networks: A survey," *IEEE Access*, vol. 10, pp. 106581–106612, 2022.

[14] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[15] J. A. H. Sánchez, K. Casilimas, and O. M. C. Rendon, "Deep reinforcement learning for resource management on network slicing: A survey," *Sensors*, vol. 22, no. 8, p. 3031, 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/8/3031

[16] C. Ssengonzi, O. P. Kogeda, and T. O. Olwal, "A survey of deep reinforcement learning application in 5G and beyond network slicing and virtualization," *Array*, vol. 14, Jul. 2022, Art. no. 100142. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2590005622000133

[17] M. Wang, Y. Lin, Q. Tian, and G. Si, "Transfer learning promotes 6G wireless communications: Recent advances and future challenges," *IEEE Trans. Rel.*, vol. 70, no. 2, pp. 790–807, Jun. 2021.

[18] M. Elsayed, M. Erol-Kantarci, and H. Yanikomeroglu, "Transfer reinforcement learning for 5G new radio mmWave networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 5, pp. 2838–2849, May 2021.

[19] T. Dong et al., "Generative adversarial network-based transfer reinforcement learning for routing with prior knowledge," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 2, pp. 1673–1689, Jun. 2021.

[20] H. Zhou, M. Erol-Kantarci, and H. V. Poor, "Learning from peers: Deep transfer reinforcement learning for joint radio and cache resource allocation in 5G RAN slicing," *IEEE Trans. Cogn. Commun. Netw.*, vol. 8, no. 4, pp. 1925–1941, Dec. 2022.

[21] N. Q. Hieu, D. T. Hoang, D. Niyato, P. Wang, D. I. Kim, and C. Yuen, "Transferable deep reinforcement learning framework for autonomous vehicles with joint radar-data communications," *IEEE Trans. Commun.*, vol. 70, no. 8, pp. 5164–5180, Aug. 2022.

[22] G. Sun, D. Ayepah-Mensah, R. Xu, V. K. Agbesi, G. Liu, and W. Jiang, "Transfer learning for autonomous cell activation based on relational reinforcement learning with adaptive reward," *IEEE Syst. J.*, vol. 16, no. 1, pp. 1044–1055, Mar. 2022.

[23] J. J. A. Esteves, A. Boubendir, F. Guillemin, and P. Sens, "A heuristically assisted deep reinforcement learning approach for network slice placement," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 4, pp. 4794–4806, Dec. 2022.

[24] L. Wang, C. Yang, X. Wang, J. Li, Y. Wang, and Y. Wang, "Integrated resource scheduling for user experience enhancement: A heuristically accelerated DRL," in *Proc. IEEE 11th Int. Conf. Wireless Commun. Signal Process. (WCSP)*, 2019, pp. 1–6.

[25] Y. Hu, M. Chen, W. Saad, H. V. Poor, and S. Cui, "Distributed multi-agent meta learning for trajectory design in wireless drone networks," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 10, pp. 3177–3192, Oct. 2021.

[26] Y. Yuan, G. Zheng, K.-K. Wong, B. Ottersten, and Z.-Q. Luo, "Transfer learning and meta learning-based fast downlink beamforming adaptation," *IEEE Trans. Wireless Commun.*, vol. 20, no. 3, pp. 1742–1755, Mar. 2021.

[27] M. K. Motalleb, V. Shah-Mansouri, S. Parsaeefard, and O. L. A. López, "Resource allocation in an open RAN system using network slicing," *IEEE Trans. Netw. Service Manag.*, vol. 20, no. 1, pp. 471–485, Mar. 2023.

[28] L. Maggi, A. Valcarce, and J. Hoydis, "Bayesian optimization for radio resource management: Open loop power control," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 7, pp. 1858–1871, Jul. 2021.

[29] R. Li et al., "Deep reinforcement learning for resource management in network slicing," *IEEE Access*, vol. 6, pp. 74429–74441, 2018.

[30] P. Gawłowicz and A. Zubow, "Ns-3 meets OpenAI Gym: The playground for machine learning in networking research," in *Proc. 22nd Int. ACM Conf. Model. Anal. Simul. Wireless Mobile Syst.*, 2019, pp. 113–120.

[31] T. Leibovich-Raveh, D. J. Lewis, S. A.-R. Kadhim, and D. Ansari, "A new method for calculating individual subitizing ranges," *J. Numer. Cogn.*, vol. 4, no. 2, pp. 429–447, Sep. 2018.

[32] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 12, pp. 2346–2363, Dec. 2019.

[33] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Mach. Learn.*, vol. 63, pp. 3–42, Mar. 2006.

[34] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated Machine Learning: Methods, Systems, Challenges*. Cham, Switzerland: Springer Nat., 2019.

[35] E. LeDell and S. Poirier, "H2O AutoML: Scalable automatic machine learning," in *Proc. AutoML Workshop ICML*, 2020, pp. 1–16.

[36] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 3, pp. 2224–2287, 3rd Quart., 2019.

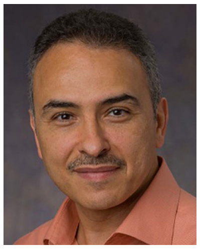
[37] A. M. Nagib, H. Abou-Zeid, H. S. Hassanein, A. B. Sediq, and G. Boudreau, "Deep learning-based forecasting of cellular network utilization at millisecond resolutions," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2021, pp. 1–6.

Ahmad M. Nagib (Graduate Student Member, IEEE) received the B.Sc. and M.Sc. degrees from the Faculty of Computers and Artificial Intelligence, Cairo University. He is currently pursuing the Ph.D. degree with the School of Computing, Queen's University, where he is a Graduate Research Fellow.

He also works as an Assistant Lecturer with Cairo University and as a Machine Learning Ph.D. co-op in the area of Cloud RAN with Ericsson, Canada. His research mainly addresses the practical challenges of applying machine learning, and specifically, reinforcement learning, in next-generation wireless networks.

Mr. Nagib served as a Reviewer and a TPC Member in several IEEE flagship venues, such as IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, GLOBECOM, ICC, and LCN.

Hatem Abou-Zeid (Member, IEEE) received the Ph.D. degree from Queen's University in 2014. He is an Assistant Professor with the University of Calgary. Prior to that, he was with Ericsson leading 5G Radio Access Research and IP in RAN intelligence, low-latency communications, and spectrum sharing. Several wireless access and traffic engineering techniques that he co-invented and co-developed are deployed in mobile networks and data centers worldwide. His work has resulted in 19 patent filings and 50 journal and conference publications in several IEEE flagship venues. His research interests are broadly in 5G/6G networks, extended reality communications, and robust machine learning.



Hossam S. Hassanein (Fellow, IEEE) is a Leading Authority in the areas of broadband, wireless and mobile networks architecture, protocols, control and performance evaluation. His record spans more than 600 publications in journals, conferences and book chapters, in addition to numerous keynotes and plenary talks in flagship venues. He has received several recognition and best paper awards at top international conferences. He is the Former Chair of the IEEE Communication Society Technical Committee on IoT, AdHoc, and Sensor Networks. He is an IEEE Communications Society Distinguished Speaker (Distinguished Lecturer from 2008 to 2010).