BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **24**, No 3
Sofia • 2024 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.2478/cait-2024-0029

Energy-Efficient and Accelerated Resource Allocation in O-RAN Slicing Using Deep Reinforcement Learning and Transfer Learning

Heba Sherif, Eman Ahmed, Amira M. Kotb

Faculty of Computers and Artificial Intelligence, Cairo University, Cairo, Egypt
E-mails: heba.sherif@fci-cu.edu.eg e.ahmed@fci-cu.edu.eg a.kotb@fci-cu.edu.eg

***Abstract*:** *Next Generation Wireless Networks* (*NGWNs*) *have two main components: Network Slicing and Open Radio Access Networks* (*O-RAN*). *NS is needed to handle various Quality of Services* (*QoS*). *O-RAN adopts an open environment for network vendors and Mobile Network Operators* (*MNOs*). *In recent years, Deep Reinforcement Learning* (*DRL*) *approaches have been proposed to solve some key issues in NGWNs. The primary obstacles preventing the DRL deployment are being slowly converged and unstable. Additionally, these algorithms have enormous carbon emissions that negatively impact climate change. This paper tackles the dynamic allocation problem of O-RAN radio resources for better QoS, faster convergence, stability, lower energy and power consumption, and reduced carbon emissions. Firstly, we develop an agent with a newly designed latency-based reward function and a top-k filtration mechanism for actions. Then, we propose a policy Transfer Learning approach to accelerate agent convergence. We compared our model to another two models.*

***Keywords*:** *O-RAN, 6G, Radio resource management, Deep reinforcement learning, Transfer learning.*

1. Introduction

The last few decades have witnessed a prominent development in the Next-Generation Wireless Networks (NGWNs) architecture due to the significant rise in the number of connected wireless devices and applications that require different Quality of Service (QoS) [1-3]. NGWNs, like 5G and 6G, are replacing rigid network architectures with dynamic, flexible, and agile architectures to support multiple heterogeneous services and technologies [4]. The rapid growth of the Internet of Everything (IoE) by having millions of connected devices shifts the network services from not only having enhanced Mobile BroadBand (eMBB) services but also having Ultra-Reliable Low Latency (URLLC) services. Although 5G is considered the key enabler for IoE, it currently supports only basic IoE and URLLC services and there is a doubt if it can support the near future demand for IoE services [5].

The primary element of cellular networks that links between the core network and user equipment is the Radio Access Network (RAN) [1, 6]. It has been developed over time to meet the needs of the rising number of connected users [6]. The RAN of the early generations of cellular networks, 2G and 3G, had controllers to manage and orchestrate the radio resources. Then, 4G networks came with an interface to support communication with the Base Station (BS) for controlling the radio resources [7]. The RAN of the current 5G networks is a Virtualized RAN (V-RAN) which divides the radio resources virtually to support diverse services. This V-RAN has monolithic components that specific vendors support and they are seen as black boxes to the Mobile Network Operators (MNOs) [7-9]. The O-RAN alliance has offered a new solution called Open RAN (O-RAN) for beyond 5G networks, due to the new concepts and intelligence added in 5G, to enable the MNO to have multiple services from multiple vendors [7,10]. With the introduction of the O-RAN concept, the RAN components of hardware and software were separated, and solutions from various vendors were integrated and interoperable [11]. The expectation for O-RAN is to achieve agility, flexibility, adaptability of operations, heterogeneity of services, and many more [11].

The possibility of O-RAN architecture being disaggregated came from the use of virtualized technologies, open standardized interfaces, and Artificial Intelligence (AI) [11]. The usage of these interfaces with AI, especially Deep Learning (DL) techniques, can provide the MNO with the capabilities to optimize the RAN performance and have intelligent Radio Resource Management (RRM) through RAN Intelligent Controllers (RICs) [6,12]. RRM includes admission control, link management, radio resource allocation, power allocation, scheduling, load balancing, handover, etc. [12]. Different types of RICs operate at different timescales according to the network operations [13]. The near Real-Time (near-RT) RIC hosts applications known as xApps while the non-Real-Time (non-RT) RIC hosts applications known as rApps. They interact with each other through interfaces.

Recently, most of the O-RAN issues have been addressed to be solved using Deep Reinforcement Learning (DRL) as it doesn't require training and testing data to be available [14]. The real-time inference of DRL makes it a preferable solution above the other optimization methods [14]. However, the DRL deployment in live networks is still in its early stages. The reasons for that are the instability of exploration during training and the slow convergence of algorithms [14]. Many research efforts today are seeking to solve the DRL slow convergence issue by using Transfer Learning (TL), equivalently named knowledge transfer [15]. TL came to reuse the knowledge of an existing agent, called an expert agent, as a starting point for another agent, called a learner agent, that has a similar task to that of the expert agent. This reduces the number of steps and samples needed to train the agent. TL is an emerging and complicated topic in DRL because the transferred knowledge can take different forms and ways [15]. Another important aspect that some researchers have recently addressed is that with the excessive usage of AI techniques, the amount of greenhouse gases, such as Carbon Dioxide (CO_2) or similar gases, increases. And, as the emission of these gases increases, the problem of global climate change is becoming more severe. As a result, the carbon footprints of the used AI techniques

should be precisely measured and reported. Shortening the training time and optimizing the employed procedures to lower the carbon footprints are critical [16].

In this paper, we tackle the problem of dynamic allocation of radio resources in O-RAN slicing using DRL. We tend to achieve the optimal allocation with fast convergence, learning stability, less energy and power consumption, and reduced carbon emissions. We developed two applications, each is a DRL agent developed using a Proximal Policy Optimization (PPO) algorithm. The first is an expert agent rApp to be deployed in the non-RT RIC. This expert agent has three slices: URLLC, Video, and Voice over Long Term Evolution (VoLTE). The second is a learner agent xApp to be deployed in the near-RT RIC. In the first scenario, it has 2 VoLTE slices and 1 URLLC slice, while in the second scenario, it has 2 Video slices and 1 URLLC slice. Also, the traffic of the learner agent is slightly different from that of the expert agent. Our work's primary contributions are summarized as follows:

- We design a new reward function that depends on latency, for keeping the QoS at a level specified by the Service Level Agreements (SLAs). Then, we add penalties for actions violating the latency required by the SLAs.
- A top-K filtration mechanism is used for filtering actions with high log probabilities for each state. The log probabilities are the logarithm of probabilities assigned to actions by the policy network of the PPO algorithm. We used this filtration during the exploration phase to force the agent to focus on a few numbers of actions and finally choose one action from them. This mechanism accelerates performance and improves learning efficiency.
- A policy transfer learning approach is proposed which accelerates the learner agent convergence by 6000 learning steps faster than the expert agent.
- A complete comparison is conducted with the model presented in [17] in terms of convergence time, number of learning steps, average packet delivery ratio (PDR), average Latency Violation Ratio (LVR), TL acceleration steps, achieved rewards, consumed energy, consumed power, and carbon footprint. Also, our TL model is compared with the hybrid TL model presented in [18]. This comparison shows that our model shows a great improvement in the number of learning steps, convergence time, consumed energy, consumed power, and carbon footprint. Additionally, our TL model achieves the maximum reward and converges faster than the TL models presented in [17, 18].

The rest of the paper is structured as follows: Section 2 reveals an overview of the O-RAN architecture. Section 3 outlines the related work. In Section 4, the suggested model is designed. Section 5 explains the implementation details. Our results and model evaluation are discussed in Section 6. Finally, the conclusion and future work are revealed in Section 7.

2. The O-RAN architecture

The architecture of O-RAN is composed of a collection of protocols and open interfaces that are used. It defines a disaggregated approach to split hardware from software and having different components that are connected by using some open

interfaces to support multiple services and vendors [11, 12]. Fig. 1 reveals the main components of O-RAN which are described below.

1. Service Management and Orchestration (SMO): It is an essential element used to manage the domain of RAN through the usage of some open interfaces such as; O1, O2, and A1 [7, 19].

2. RAN Intelligent Controller (RIC): The O-RAN key component that helps make automated and intelligent decisions [7]. It is composed of two logical functions:

- The non-RT RIC which resides in the SMO framework outside RAN to obtain external data useful in RAN optimization [20]. It comprises two sub-components: non-RT RIC applications (called rApps) that benefit from the SMO services and offer new services to provide non-real-time (i.e., longer than one second) RAN resources optimization. The second sub-component is the non-RT RIC framework which supports the needed services to rApps [12].
- The near-RT RIC which controls both the O-RAN Distributed Unit (O-DU) and O-RAN Centralized Unit (O-CU) nodes in a near real-time scale (i.e., few milliseconds). It hosts applications, called xApps, which use both the E2 and A1 interfaces to offer new services such as the management of spectrum, radio resources, power resources, mobility, etc. The xApps uses the E2 interface to gather the near-RT data and the A1 interface to collect the non-RT RIC policies and data [12, 21].

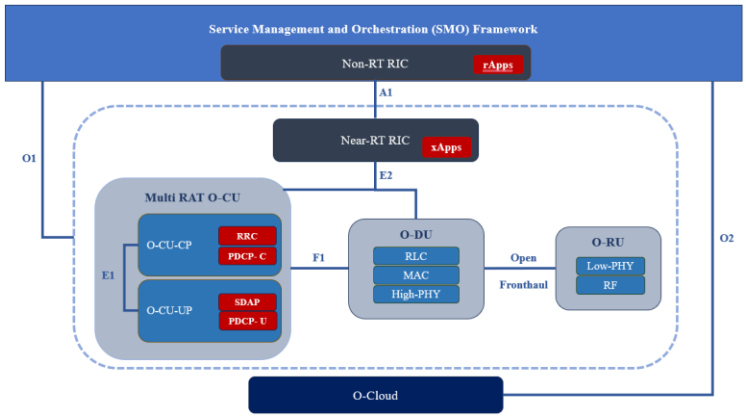


Fig. 1. High-level O-RAN architecture

3. O-RAN Centralized Unit (O-CU): It is a virtual component in the O-RAN that is divided into the User Plane (O-CU-UP) and Control Plane (O-CU-CP) [7]. The Service Data Adaptation Protocol (SDAP) and the user plane of the Packet Data Control Protocol (PDCP) are hosted by the O-CU-UP part. The Radio Resource Control (RRC) protocol and the control plane of PDCP are hosted by the O-CU-CP part. The E1 interface connects O-CU-UP and O-CU-CP [12].

4. O-RAN Distributed Unit (O-DU): It is a virtual component in the O-RAN that supports the functionalities of the High PHYsical (High-PHY) layer, Medium Access Control (MAC) layer, and Radio Link Control (RLC) layer [12]. The O-DU node is connected to the two O-CU planes via F1-c and F1-u interfaces to support some functions related to the High-PHY, MAC, and RLC layers [7].

5. O-RAN Radio Unit (O-RU): It is a logical node in the O-RAN that hosts the Low PHYsical (Low-PHY) layer functions and the Radio Frequency (RF) operations. The O-RU node is connected to the O-DU node via the open fronthaul interface [12].

6. O-RAN Cloud (O-Cloud): It is a physical component in the O-RAN. It hosts the functions of O-CU, O-DU, and near-RT RIC as Virtual Network Functions (VNFs) [12, 22]. The O-Cloud is connected to the SMO framework via the O2 interface.

3. Related work

Many researchers target the usage of DRL in solving the allocation problem of RAN resources as it can reach the optimal solution without prior knowledge about the traffic of the slices. The main problem of deploying DRL algorithms in live networks is the slow convergence. All the work done is trying to reach the optimal solution in fewer learning steps. To achieve this goal, they tend to design a good reward function, fine-tune the hyperparameters, and use different DRL algorithms. Also, some of the recent work suggested applying TL techniques. Some of these works are presented in this section.

In [17], a sigmoid-based reward function and a hybrid TL-based approach are proposed for accelerating the DRL agent. The techniques applied are reward-shaping, policy distillation, policy reuse, and a hybrid approach of policy reuse and policy distillation. For intra-slice scheduling, a round-robin with a 0.5 msec slot is used. They employed PPO as the underlying algorithm for their proposed agent. They used the needed learning steps to evaluate their approach. Their results show that the expert agent converges after 18000 learning steps using the sigmoid function, while it converges after 11000 learning steps using reward shaping. The convergence of their learner agent using policy reuse, policy distillation, and the hybrid approach is compared and the hybrid approach reveals the best convergence after 10000 learning steps. The Tensorforce Python package is used to implement the DRL algorithms.

The work in [23] presented the implementation of three DRL-based O-RAN xApps. The implementation of their architecture is done using the Colosseum network emulator. The training of DRL agents is performed offline on a gathered dataset using Colosseum. The first xApp is developed using one DRL agent for controlling the slicing and scheduling of a single BS. The second xApp has one DRL agent per slice for selecting the scheduling policy for each slice in parallel. The third xApp trains online DRL agents. The DRL agents are trained with the PPO algorithm and implemented using Tensor Flow 2.4 and the TF-Agents library. Autoencoder is used for reducing the number of observations fed as inputs to the DRL agent. The results show that the expert agent converges after 17460 learning steps and that the online learner agent converges after 12360 learning steps.

Z h a n g, Z h o u and E r o l-K a n t a r c i [24] proposed a Federated DRL (FRL) algorithm for allocating power and radio resources in O-RAN and coordinating between independent xApps. They designed two xApps; both of them use the DQN algorithm. The first xApp is the power agent which decides the transmission power level for the BS then, this power is uniformly distributed among all the available

PRBs. The second xApp is the radio agent that decides the number of PRBs to be assigned to each slice. The federated process in the proposed model is done in three steps. First, each agent calculates its local Q-table then, the two local Q-tables are submitted to the global model for calculating a joint global Q-table. Finally, two calibrated Q-tables for action selection are separated from this global Q-table. The implementation of the model is developed by MATLAB 5G toolbox. For intra-slice allocation, PPF is deployed. The proposed model is compared with Independent Reinforcement Learning (IRL) and Centralized Reinforcement Learning (CRL) algorithms. They reveal that the proposed FRL achieves lower delay for URLLC slices and better throughput for eMBB slices than the IRL algorithm. Although it achieves a lower reward than CRL, it converges faster.

Two TL-based approaches are investigated in [25] for jointly allocating computation and radio RAN resources in multi-access edge computing for 5G networks. The scenario includes one expert agent that knows radio resources, one expert agent that knows computation resources, and one learner agent that takes its knowledge from the two expert agents to jointly allocate computation and radio resources. The expert agents are developed using a Q-learning Algorithm. They used two TL methods: Q-value Transfer-based DRL (QTDRL) and Action-selection Transfer-based DRL (ATDRL). In QTDRL, the learner agent uses the expert agent Q-values as extra incentives when updating its Q-values. In ATDRL, the learner agent reduces its action space by selecting only actions that achieve high rewards in the expert agents. They compared their proposed methods with Priority Proportional Fairness (PPF) and Deep Q-learning (DQN) algorithms. Their simulation shows a faster convergence, higher average rewards, and lower probability of delays for the proposed methods compared to the DQN and PPF algorithms. Moreover, they stated that the convergence of ATDRL is better than that of QTDRL because of the reduction in action space and better exploration efficiency.

A TL Multi-agent DRL approach for the partitioning of resources between cells is presented in [26]. They examined the similarity between agents (cells) in terms of both domain and task based on the features extracted by the variational auto-encoder. Then, they designed a knowledge transfer approach to transfer both policy and instance from the selected expert agent to the target learner agent. The instance transfer strategy combines the instances (domain and action) from both expert and learner agents and saves them in the learner replay buffer. To capture intercell interference, each agent shares the network load in each slice with its neighboring agents. Each local agent is implemented using the TD3 Algorithm. The inter-agent distance is measured by the KL divergence. They stated that the proposed TL approach provides a higher start at the beginning of training and achieves about 12% failure to satisfy 0.95 of the SLA requirements.

The same authors of [17] presented another hybrid TL-based approach in [18]. Their proposed method uses a combination of policy reuse and policy distillation. The training of DRL agents is done in the O-RAN non-RT RIC while the deployment is done in the O-RAN near-RT RIC. The PPO Algorithm is deployed and implemented using the Tensor force Python package. The proposed approach converges after 4000 to 6000 learning steps.

A bi-level model for joint allocation of power and sub-channels of radio resources in 5G networks is proposed in [27]. The Multi-Agent Twin Delayed deep deterministic policy gradient (MATD3) algorithm is used for the first level to allocate resources to the network slices. The Discrete and Continuous Twin Delayed deep deterministic policy gradient (DCTD3) algorithm is used for the second level to allocate each slice's resources fairly to its users. The MATD3 algorithm has two agents for each base station: one agent for continuous power allocation and the other for discrete sub-channel allocation. The DCTD3 has one agent for each slice which is responsible for the allocation for both power and sub-channel. The simulation is developed in Python using PyTorch-GPU. The proposed model is compared with the nested bi-level evolutionary and multi-agent deep deterministic policy gradient algorithms. The model shows high reward in less than 1000 epochs but it converges after 2250 epochs and each epoch has 50 TTIs which means that it converges after 112500 learning steps.

The primary goal of this paper is to improve both expert agent rApp and learner agent xApp performance in terms of convergence time, consumed energy and power, carbon emissions, and achieved QoS. To achieve this objective, we first designed a reward function that guides the expert agent to the optimal allocation of radio resources among slices. Then, we accelerate the performance of the expert agent by filtering the actions. The MNO will train many expert agents with different slices and different traffic patterns therefore; convergence of these agents should be done in as few learning steps as possible, low carbon footprint, low energy, and low power. Finally, we propose a policy transfer learning approach for accelerating the performance of the learner agent xApp to be deployed in live networks.

4. Model design

Our model uses the PPO Algorithm to distribute O-RAN radio resources among slices. We choose to use the PPO for its implementation simplicity, reliability, and stability of its policy update, which is introduced by the clipped objective function. This is in addition to its efficiency as it can achieve great performance using fewer computational resources [28]. We focus on the downlink direction of three slices (Video, URLLC, and VoLTE). The PPO agent is responsible for the interslice allocation of available PRBs. Then, the round-robin algorithm is used by each slice for intra-slice allocation of resources among users. Requests from users are created according to the traffic pattern described in Table 1.

The traffic in commercial networks is different from that of the offline simulation. Also, the MNO may change the type or the number of the slices, the state representation, the action space, the priorities of the slices, the SLAs, etc. Therefore, training the agent from scratch in the commercial network will result in degrading the QoS and subsequently violating the SLAs for a long time. That's why transfer learning is highly recommended to be used with DRL agents. We propose a policy transfer learning model for accelerating the performance of the DRL agent when the traffic and type of slices are slightly changed. We developed an expert agent, a non-RT RIC rApp, and a learner agent, a near-RT RIC xApp. The workflow between the

two agents is shown in Fig. 2. The expert agent rApp is trained by interacting with the simulated O-RAN environment through the O1 interface. Then, the expert agent transfers its policy to the learner agent xApp through the A1 interface. Finally, the learner agent xApp is guided by the expert agent policy until it trains its policy by interacting with the commercial O-RAN environment through the E2 interface. The interaction with the environment occurs at each Transmission Time Interval (TTI) in which the agent notices the state of the environment and it acts accordingly to maximize the reward for all slices. The agent is designed in the following sub-sections.

Table 1. Simulation parameters for traffic generation of the slices

Parameter	Video	URLLC	VoLTE
Bandwidth	20 MHz		
Intra-slice scheduling Algorithm	Round-Robin Algorithm (0.25 ms slot for URLLC, 0.5 ms slot for Video and VoLTE)		
Bandwidth allocation window size	20 ms		
Connected Users	50	80	50
Packet interarrival time distribution	Truncated Pareto [max = 10 ms, mean = 5 ms]	Exponential [mean = 100 ms]	Constant [20 ms]
Packet size distribution	Truncated Pareto [max = 1500 Byte, mean = 1000 Byte]	Truncated log-normal [max = 20 KB, standard deviation = 5 KB, mean = 10 KB]	Uniform [min = 60 Byte, max = 120 Byte]
SLAs: Latency	7 ms	1 ms	10 ms

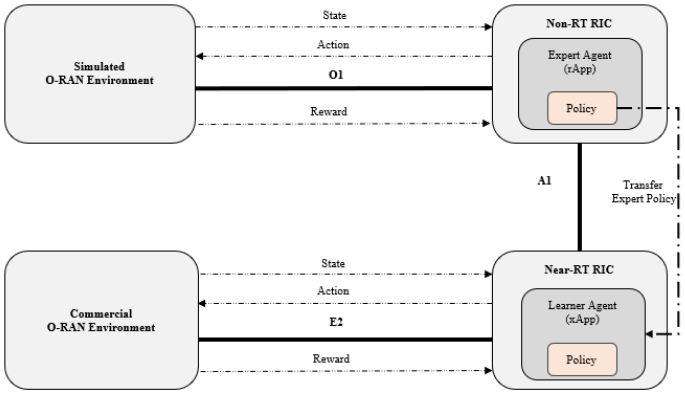


Fig. 2. The workflow of policy transfer between agents

4.1. State space

The system's current state S as defined below is the traffic demand of each slice in the previous TTI relative to the total demand of all slices, where, D_n is the demand of the slice n, and N is the number of slices:

(1) $$S = \left(\frac{D_{\text{Video}}}{\sum_n^N D_n}, \frac{D_{\text{URLLC}}}{\sum_n^N D_n}, \frac{D_{\text{VoLTE}}}{\sum_n^N D_n}\right).$$

4.2. Action space

The target task of the proposed DRL agent is to allocate PRBs to all slices. Therefore, the action of our agent is defined by the percentage of resources to be allocated to each slice. We use 15 allocation actions for the agent to choose from them. Those actions are shown in Table 2.

Table 2. Agent allocation actions

Actions	[33.3, 33.3, 33.3], [50, 30, 20], [70, 20, 10], [80, 10, 10], [10, 80, 10], [10, 70, 20], [20, 70, 10], [10, 60, 30], [30, 60, 10], [30, 50, 20], [20, 50, 30], [30, 40, 30], [10, 50, 40], [40, 40, 20], [60, 30, 10]

4.3. Reward function design

Our agent's goal is to find an optimal allocation of PRBs without violating the SLAs of the slices. Our reward function reflects the SLAs of the slices by latency. The latency is calculated using two terms: the first represents the ratio between the maximum latency required by the SLAs and the average latency of packets. The second term represents how many packets violate the required latency compared to the total number of packets. Each term is clipped to be between zero and one for stable learning, normalization, and better exploration. We choose to penalize actions with an average delay greater than the required delay by the SLAs. This penalization is done by using a power parameter θ_n for the first term which means that as the average delay increases, the penalization increases (i.e., the reward decreases). The reward R is calculated as

$$R = \sum_n^N w_n \times \min\left(\frac{D_{\mathrm{SLA}_n}}{D_{\mathrm{Avg}_n}}, 1\right)^{\theta_n} \times \left(1 - \frac{\mathrm{LVP}_n}{P_n}\right), \quad (2)$$

where: w_n is the weight of slice n; D_{SLA_n} is the maximum packet delay required by the SLAs of slice n; D_{Avg_n} is the average delay of packets in slice n; θ_n is the penalty parameter of slice n; LVP_n is the number of latency violation packets; P_n is the total number of packets of slice n.

5. Implementation details

This section presents the used environment and simulation and the implementation of both agents is explained in detail.

5.1. Environment and simulation

The implementation of agents' applications is performed using Python. The PPO algorithm is implemented using the Stable-Baselines3 package [29]. We use Py5cheSim [30], a 5G network simulator, for simulating the network slices, traffic generation, and resource allocation. The amount of consumed energy, power, and carbon emitted are calculated using the CodeCarbon package [31]. The agent environment is developed using OpenAI Gym [32]. All experiments are conducted on the Central Processing Unit (CPU) of a laptop with (11th Generation) Intel Core i7-11800H @ 2.30GHz processor, 16GB of RAM, and NVIDIA GeForce RTX 3060 Graphics Processing Unit (GPU).

5.2. Expert agent (rApp)

We use the PPO algorithm for allocating available PRBs to the slices. All the parameters used by our expert agent after doing many experiments are shown in Table 3. The expert agent rApp is explained in Algorithm 1. We use a top-k actions filtration technique to focus on a subset of actions to achieve faster convergence and improve learning efficiency.

Table 3. Expert agent parameters

No	Parameter	Value
1	epsilon	1
2	epsilon decay	0.95
3	decay_every_xsteps	1000
4	max_exploration_steps	9000
5	total_steps	20,000
6	filtration_step	7000
7	learning_rate	0.01
8	batch_size	32
9	wvideo, wurllc, wvolte	0.333, 0.333, 0.333
10	$D_{\mathrm{SLA}_{\mathrm{video}}}$, $D_{\mathrm{SLA}_{\mathrm{urllc}}}$, $D_{\mathrm{SLA}_{\mathrm{volte}}}$	7, 1, 10
11	θ_{video}, θ_{urllc}, θ_{volte}	1.75, 2, 1.5
12	K	5

This technique considers the top ‘k’ actions with the highest log probabilities for each state. The drawback of this approach is that it might miss valuable actions with initially low probabilities. To overcome this shortage, we don’t apply the filtration from the beginning and start using it after 7000 learning steps. Then, we focus on training the agent with those filtered actions for another 2000 steps. Finally, the optimal solution is achieved after 9000 learning steps.

Algorithm 1. Expert Agent rApp

Initialize

All the parameters described in Table 3

Current state = (0, 0, 0)

step = 0

Step. 1. While step < total_steps **do**

Step. 2. If step >= max_exploration_steps **Then**

Step. 3. epsilon = 0

Step. 4. End If

Step. 5. If (step % decay_every_xsteps == 0) & (step > 0) **Then**

Step. 6. epsilon = epsilon * epsilon_decay

Step. 7. End If

Step. 8. p = random number between 0 and 1

Step. 9. If p < epsilon **Then** *# Exploration*

Step. 10. If step >= filtration_step **Then**

Step. 11. filter actions and choose the top-k actions

Step. 12. choose random action from the top-k actions

Step. 13. Else

Step. 14. choose a random action from all actions
Step. 15. End If
Step. 16. Else *# Exploitation*
Step. 17. choose the best action with the highest log probability
Step. 18. End If
Step. 19. Allocate resources based on the selected action
Step. 20. Calculate the reward using (2)
Step. 21. Set the next state according to (1)
Step. 22. Train the agent on every batch_size step
Step. 23. End While

5.3. Learner agent (xApp)

We employ policy transfer learning in our model. Policy transfer is a TL technique where a teacher policy is transferred to a student agent having a task similar to that of the teacher agent. We first train an expert agent rApp at the non-RT RIC, then, the policy of this expert agent is transferred via the A1 O-RAN interface to a learner agent xApp at the near-RT RIC to accelerate its learning in a live network. We design the learner agent to explore all the actions for the first 2000 learning steps. Then by comparing the learner policy with that of the expert, we filter the actions to be explored by choosing only actions whose log probabilities increase or remain the same and ignore others. We explore those filtered actions, which are considered sub-optimal solutions, for 1000 learning steps and finally, the action with the highest increased probability is chosen for another 1000 steps. Convergence is achieved after 3000 learning steps. Our proposed learner agent is described in Algorithm 2 and it uses the parameters stated in Table 4.

Algorithm 2. Learner Agent xApp
Initialize
All the parameters described in Table 4
Current state = (0, 0, 0)
step = 0
Step. 1. While step < total_steps **do**:
Step. 2. If step >= max_exploration_steps **The**
Step. 3. epsilon = 0
Step. 4. End If
Step. 5. If (step % decay_every_xsteps == 0) & (step > 0) **Then**
Step. 6. epsilon = epsilon * epsilon_decay
Step. 7. End If
Step. 8. p = random number between 0 and 1
Step. 9. If p < epsilon **Then** *# Exploration*
Step. 10. If step >= first_filtration_step **Then**
Step. 11. compare the log probabilities of actions in the learner policy with that in the expert policy
Step. 12. filter actions whose probability increases or remains the same in the learner policy
Step. 13. If step <= second_filtration_step **Then**

Step. 14. choose a random action from the filtered actions
Step. 15. Else
Step. 16. choose the action with the highest increased probability
Step. 17. End If
Step. 18. Else
Step. 19. choose a random action from all actions
Step. 20. End If
Step. 21. Else *# Exploitation*
Step. 22. choose the best action with the highest log probability
Step. 23. End If
Step. 24. Allocate resources based on the selected action
Step. 25. Calculate the reward using (2)
Step. 26. Set the next state according to (1)
Step. 27. Train the learner agent in every batch_size step
Step. 28. End W**hile**

Table 4. Learner agent parameters

No	Parameter	Value
1	epsilon	1
2	epsilon_decay	0.95
3	decay_every_xsteps	1000
4	max_exploration_steps	4000
5	total_steps	20000
6	first_filtration_step	2000
7	second_filtration_step	3000
8	learning_rate	0.01
9	batch_size	32
10	w_{video}, w_{urllc}, w_{volte}	0.333, 0.333, 0.333
11	$D_{\text{SLA}_{\text{video}}}$, $D_{\text{SLA}_{\text{urllc}}}$, $D_{\text{SLA}_{\text{volte}}}$	7, 1, 10
12	θ_{video}, θ_{urllc}, θ_{volte}	1.75, 2, 1.5

6. Results and discussion

This section presents the results obtained from the proposed model and then, the performance is evaluated and comprehensively compared with the models presented by Nagib, Abou-zeid and Hassanein [17, 18] using several metrics. We compared our results with [17, 18] because their methodology is very similar to our model and their implementations are available online with all the needed parameters. We also chose them because they are published in reputable journals (IEEE Network and IEEE Journal on Selected Areas in Communications). These reasons serve as a suitable baseline to assess how well our suggested model performs.

6.1. Expert agent (rApp)

We first test the expert agent using our proposed reward function without filtration of actions. Then, we apply the filtration mechanism and evaluate its effect on convergence. We use six metrics to evaluate our expert agent's performance: the number of learning steps needed to converge, the convergence time, the average

PDR, the average Latency Violation Ratio (LVR), the consumed energy, the consumed power, and the carbon footprint. The PDR is calculated as the ratio of packets sent in a TTI to the total packets required to be sent in the same TTI. In training, the non-transmitted packets with latency violations are dropped at each TTI's end. We integrate the model proposed in [17] with our simulation to have the same environment and traffic pattern. We also gave the slices in all the models equal weights, reflecting the equal priority of fulfilling the SLAs for all slices. The analysis of the results is explained in the following sub-sections.

6.1.1. Learning steps

The proposed reward function guides the expert agent to converge after 11000 learning steps. At the same time, the reward function with the filtration mechanism results in convergence after 9000 learning steps. The authors in [17] proposed a model with a sigmoid reward function that converges after 18,000 learning steps. The main reasons for the delay in convergence of their model are as follows:

- Their sigmoid reward function not only penalizes actions but, also gives a bonus to actions with an average delay below the required one. Our model doesn't give any bonuses and only penalizes actions with a high average delay.
- The difference between the rewards achieved by actions is small because it depends only on the average delay but our reward function depends on the average delay and the number of packets violating the required delay.

The authors in [17] also proposed a model with reward shaping which results in converging after 11,000 learning steps. Fig. 3 shows the convergence of rewards for all models. The average reward is calculated for each 1000 learning steps.

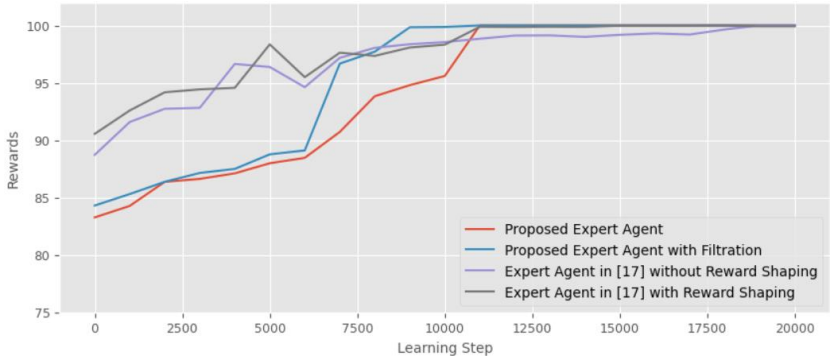


Fig. 3. Comparison of learning performance

6.1.2. Convergence time

Our model shows a great improvement in the convergence time compared to the one in [17] because of the following:

- Our model converges in a smaller number of steps.
- We use a batch size of 32 while the authors of [17] use a batch of size 8.

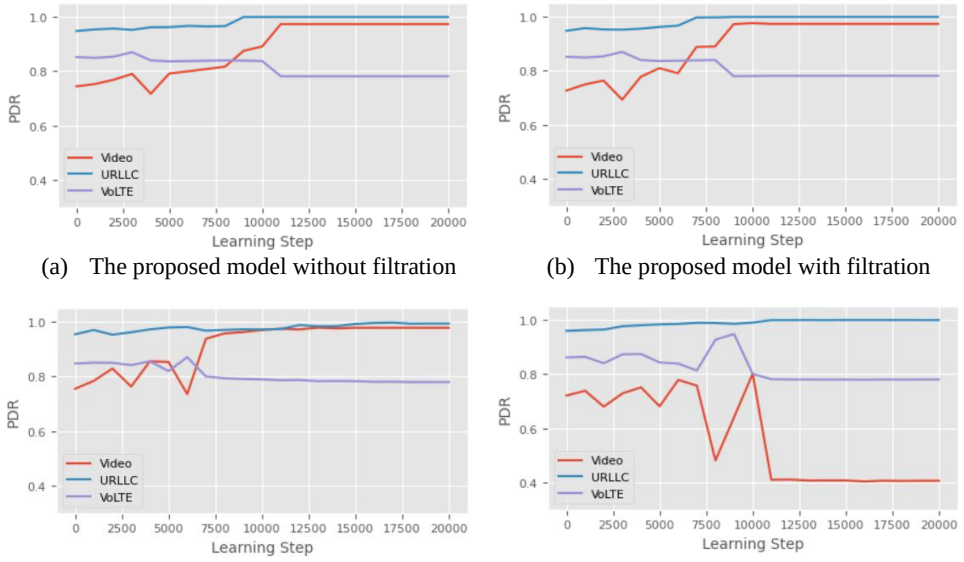
The average convergence time of 50 runs is shown in Table 5.

Table 5. Comparison of convergence time

Model	Time, s
Proposed model without filtration	2751.125
Proposed model with filtration	2198.991
Model in [17] without reward shaping	4866.046
Model in [17] using reward shaping	3273.694

6.1.3. Average packet delivery ratio and latency violation ratio

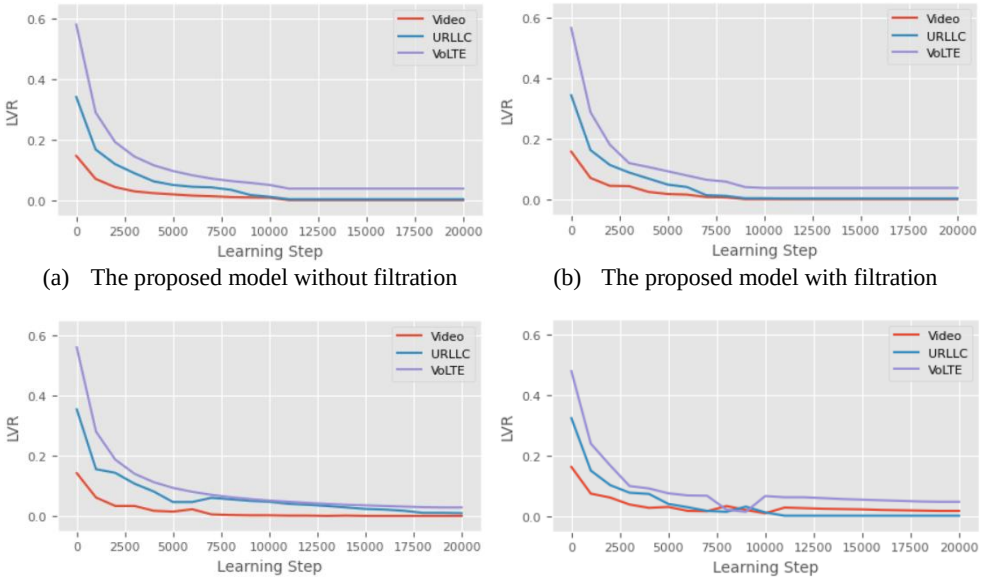
The proposed model achieves a low average LVR for all slices, a high average PDR for video and URLLC slices, and an average PDR of 80% for VoLTE slice after 9000 learning steps. The model proposed in [17] without reward shaping achieved the same average LVR and PDR as our model after 18000 learning steps. Moreover, the model in [17] using the reward shaping achieved a high average PDR for only the URLLC slice because the reward shaping gives priority to the URLLC slice and ignores other slices. Therefore, the URLLC slice will reserve many more PRBs than needed without any enhancement in its performance, subsequently, the remaining PRBs are not sufficient for the requests of the other two slices. The average PDR and LVR of all models are revealed in Fig. 4 and Fig. 5, respectively. The average values of PDR and LVR are calculated for each 1000 learning steps.



(a) The proposed model without filtration (b) The proposed model with filtration

(c) Model in [17] without reward shaping (d) Model in [17] with reward shaping

Fig. 4. Comparison of average Packet Delivery Ratio (PDR)



(a) The proposed model without filtration (b) The proposed model with filtration

(c) Model in [17] without reward shaping (d) Model in [17] with reward shaping

Fig. 5. Comparison of average latency violation ratio

6.1.4. Energy consumption, power consumption, and carbon footprint

We compared the consumed energy, the consumed power, and the amount of carbon emitted by our proposed expert agent using filtration with those of the expert agent in [17] that used the reward function without shaping because it is the one used by authors when applying TL. The average consumed power, consumed energy, and carbon footprint of 5 runs are shown in Table 6. Our proposed expert agent saves approximately 72% of the GPU power, 57% of the energy, and 59% of the carbon emissions.

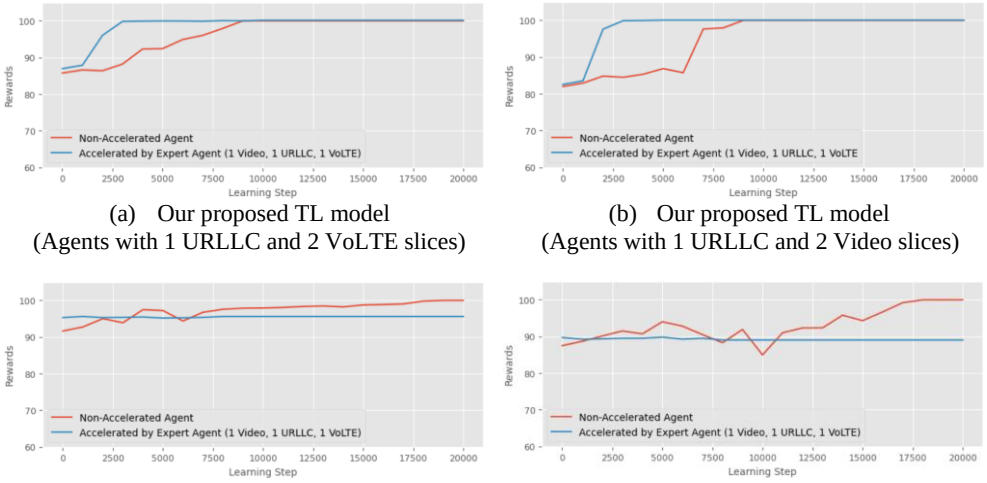
Table 6. Comparison of energy and power consumption and carbon footprint

Metric	Proposed expert agent	Expert agent in [17]
CPU power consumption	22.5 W	22.5 W
GPU power consumption	10.22 W	36.39 W
Energy consumption	0.0237 kW.h	0.0552 kW.h
Carbon footprint	0.011 kg.CO2eq	0.027 kg.CO2eq

6.2. Learner agent (xApp)

In the transfer learning scenario, the expert agent has three slices: one Video, one URLLC, and one VoLTE. Then, we use two learner agents with a slight difference in the traffic pattern from that of the expert agent and different types of services. The first learner agent has one URLLC, and two VoLTE slices while the second one has one URLLC and two Video slices. The authors in [17] proposed three models for TL: policy transfer, policy distillation, and a hybrid model between both of them. They stated that the hybrid model is the best, therefore, we compared our proposed TL model with this hybrid one. Our proposed model reaches above 95% of the maximum reward after 2000 learning steps. Then, it converges to the maximum reward after

3000 steps. The hybrid model presented in [17] converges after 10000 learning steps and doesn't reach the maximum reward reached by the non-accelerated agent. The reason is that they used a too small exploration rate and their model exploits only an action between the best expert action and the best learner action, which is the same at the beginning. Therefore, the learner agent takes the expert actions and does not learn anything new and consequently, it could not reach an optimal solution. The comparison of the learning performance of the TL models with the non-TL models is shown in Fig. 6. Also, the same authors in [17] presented another hybrid TL model in [18] using the same sigmoid reward function. Their results reveal that the convergence is achieved after 4000 to 6000 learning steps which means that our TL model converges faster.



(a) Our proposed TL model
(Agents with 1 URLLC and 2 VoLTE slices)

(b) Our proposed TL model
(Agents with 1 URLLC and 2 Video slices)

(c) The hybrid TL model in [17]
(Agents with 1 URLLC and 2 VoLTE slices)

(d) The hybrid TL model in [17]
(Agents with 1 URLLC and 2 Video slices)

Fig. 6. Comparison of learning performance of the TL models with the non-TL models

7. Conclusion and future work

In this paper, we addressed the issue of RRM in O-RAN slicing of the NGWNs to achieve optimal dynamic radio resource allocation with better QoS for all slices, less energy and power consumption, and reduced carbon emissions. The deployment of DRL agents to distribute radio resources among various slices needs thousands of steps to learn and converge to the optimal policy. Consequently, this results in SLA violation and having unsatisfied QoS for a long time. Moreover, they increase the amount of energy and power consumption and carbon emissions. Designing a good reward function and using TL help in guiding the agent in the exploration phase to converge to the optimal solution faster. Therefore, we proposed a model that uses a latency-based reward function with penalties, a top-k filtration mechanism for actions, and a policy transfer technique that helps the agent to converge in fewer learning steps. We compared our model with the one proposed in [17] and the results revealed that our expert agent saved 50% of the learning steps, 72% of the GPU

power, 57% of the energy, and 59% of the carbon emissions. Also, it shows that our learner agent can achieve the maximum reward in 3000 steps while other TL models cannot. In the future, we want to address the applicability of our model in a real network environment and apply other DRL algorithms and TL approaches.

References

1. Ghosh, A., A. Maeder, M. Baker, D. Chandramouli. 5G Evolution: A View on 5G Cellular Technology Beyond 3GPP Release 15. – IEEE Access, Vol. **7**, 2019, pp. 127639-127651. DOI: 10.1109/access.2019.2939938.
2. Zong, B., C. Fan, X. Wang, X. Duan, B. Wang, J. Wang. 6G Technologies: Key Drivers, Core Requirements, System Architectures, and Enabling Technologies. – IEEE Vehicular Technology Magazine, Vol. **14**, 2019, No 3, pp. 18-27. DOI: 10.1109/mvt.2019.2921398.
3. Singh, S. K., R. Singh, B. Kumbhani. The Evolution of Radio Access Network towards Open-RAN: Challenges and Opportunities. – In: Proc. of IEEE Wireless Communications and Networking Conference Workshops (WCNCW'20), IEEE, Korea, 2020, pp. 1-6. DOI: 10.1109/wcncw48565.2020.9124820.
4. Abdalla, A. S., P. S. Upadhyaya, V. K. Shah, V. Marojevic. Toward Next Generation Open Radio Access Networks: What O-RAN Can and Cannot Do! – IEEE Network, Vol. **36**, 2022, No 6, pp. 206-213. DOI: 10.1109/mnet.108.2100659.
5. Saad, W., M. Bennis, M. Chen. A Vision of 6G Wireless Systems: Applications, Trends, Technologies, and Open Research Problems. – IEEE Network, Vol. **34**, 2019, No 3, pp. 134-142. DOI: 10.1109/mnet.001.1900287.
6. Azariah, W., F. A. Bimo, C. W. Lin, R. G. Cheng, N. Nikaein, R. Jana. A Survey on Open Radio Access Networks: Challenges, Research Directions, and Open Source Approaches. – Sensors, Vol. **24**, 2024, No 3, p. 1038. DOI: 10.3390/s24031038.
7. Liyanage, M., A. Braeken, S. Shahabuddin, P. Ranaweera. Open RAN Security: Challenges and Opportunities. – Journal of Network and Computer Applications, Vol. **214**, 2023, p. 103621. DOI: 10.1016/j.jnca.2023.103621.
8. Polese, M., L. Bonati, S. D'oro, S. Basagni, T. Melodia. Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges. – IEEE Communications Surveys & Tutorials, Vol. **25**, 2023, No 2, pp. 1376-1411. DOI: 10.1109/comst.2023.3239220.
9. Parvez, I., A. Rahmati, I. Guvenc, A. I. Sarwat, H., H. Dai. A Survey on Low Latency towards 5G: RAN, Core Network and Caching Solutions. – IEEE Communications Surveys & Tutorials, Vol. **20**, 2018, No 4, pp. 3098-3130. DOI: 10.1109/comst.2018.2841349.
10. Abeta, S., T. Kawahara, A. Umesh, R. Matsukawa. O-RAN Alliance Standardization Trends. – NTT DOCOMO Technical Journal, Vol. **21**, 2019, No 1, pp. 38-45.
11. Hamdan, M. Q., H. Lee, D. Triantafyllopoulou, R. Borralho, A. Kose, E. Amiri, D. Mulvey, W. Yu, R. Zitouni, R. Pozza, B. Hunt. Recent Advances in Machine Learning for Network Automation in the O-RAN. – Sensors, Vol. **23**, 2023, No 21, p. 8792. DOI: 10.3390/s23218792.
12. Brik, B., K. Boutiba, A. Ksentini. Deep Learning for B5G Open Radio Access Network: Evolution, Survey, Case Studies, and Challenges. – IEEE Open Journal of the Communications Society, Vol. **3**, 2022, pp. 228-250. DOI: 10.1109/ojcoms.2022.3146618.
13. Bonati, L., M. Polese, S. D'Oro, S. Basagni, T. Melodia. Intelligent Closed-Loop RAN Control with xApps in OpenRAN Gym. – In: Proc. of 27th European Wireless Conference, Germany, 2022, pp. 1-6.
14. Feriani, A., E. Hossain. Single and Multi-Agent Deep Reinforcement Learning for AI-Enabled Wireless Networks: A Tutorial. – IEEE Communications Surveys & Tutorials, Vol. **23**, 2021, No 2, pp. 1226-1252. DOI: 10.1109/comst.2021.3063822.

15. Zhu, Z., K. Lin, A. K. Jain, J. Zhou. Transfer Learning in Deep Reinforcement Learning: A Survey. – IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. **45**, 2023, No 11, pp. 13344-13362. DOI: 10.1109/TPAMI.2023.3292075.
16. Henderson, P., J. Hu, J. Romoff, E. Brunskill, D. Jurafsky, J. Pineau. Towards the Systematic Reporting of the Energy and Carbon Footprints of Machine Learning. – Journal of Machine Learning Research, Vol. **21**, 2020, pp. 1-43.
17. Nagib, A. M., H. Abou-zeid, H. S. Hassanein. Toward Safe and Accelerated Deep Reinforcement Learning for Next-Generation Wireless Networks. – IEEE Network, Vol. **37**, 2022, No 2, pp. 182-189. DOI: 10.1109/mnet.106.2100578.
18. Nagib, A. M., H. Abou-zeid, H. S. Hassanein. Safe and Accelerated Deep Reinforcement Learning-Based O-RAN Slicing: A Hybrid Transfer Learning Approach. – IEEE Journal on Selected Areas in Communications, Vol. **42**, 2023, No 2, pp. 310-325. DOI: 10.1109/jsac.2023.3336191.
19. Wang, T. H., Y. C. Chen, S. J. Huang, K. S. Hsu, C. H. Hu. Design of a Network Management System for 5G Open RAN. – In: Proc. of IEEE Asia-Pacific Network Operations and Management Symposium (APNOMS'21), IEEE, Taiwan, 2021, pp. 138-141. DOI: 10.23919/apnoms52696.2021.9562627.
20. Wypiór, D., M. Klinkowski, I. Michalski. Open RAN-Radio Access Network Evolution, Benefits, and Market Trends. – Applied Sciences, Vol. **12**, 2022, No 1, p. 408. DOI: 10.3390/app12010408.
21. Orhan, O., V. N. Swamy, T. Tetzlaff, M. Nassar, H. Nikopour, S. Talwar. Connection Management xAPP for O-RAN RIC: A Graph Neural Network and Reinforcement Learning Approach. – In: Proc. of IEEE International Conference on Machine Learning and Applications (ICMLA'21), IEEE, USA, 2021, pp. 936-941. DOI: 10.1109/icmla52953.2021.00154.
22. Tamim, I., A. Saci, M. Jammal, A. Shami. Downtime-Aware O-RAN VNF Deployment Strategy for Optimized Self-Healing in the O-Cloud. – In: Proc. of IEEE Global Communications Conference (GLOBECOM'21), IEEE, Spain, 2021, pp. 1-6. DOI: 10.1109/globecom46510.2021.9685775.
23. Polese, M., L. Bonati, S. D'Oro, S. Basagni, T. Melodia. ColO-RAN: Developing Machine Learning-Based xApps for Open RAN Closed-Loop Control on Programmable Experimental Platforms. – IEEE Transactions on Mobile Computing, Vol. **22**, 2022, No 10, pp. 5787-5800. DOI: 10.1109/tmc.2022.3188013.
24. Zhang, H., H. Zhou, M. Erol-Kantarci. Federated Deep Reinforcement Learning for Resource Allocation in O-RAN Slicing. – In: Proc. of IEEE Global Communications Conference (GLOBECOM'22), IEEE, Brazil, 2022, pp. 958-963. DOI: 10.1109/globecom48099.2022.10001658.
25. Zhou, H., M. Erol-Kantarci, V. Poor. Knowledge Transfer and Reuse: A Case Study of AI-Enabled Resource Management in RAN Slicing. – IEEE Wireless Communications, Vol. **30**, 2022, No 5, pp. 160-169. DOI: 10.1109/mwc.004.2200025.
26. Hu, T., Q. Liao, Q. Liu, G. Carle. Network Slicing via Transfer Learning Aided Distributed Deep Reinforcement Learning. – In: Proc. of IEEE Global Communications Conference (GLOBECOM'22), IEEE, Brazil, 2022, pp. 2909-2914. DOI: 10.1109/globecom48099.2022.10000763.
27. Yu, Z., F. Gu, H. Liu, Y. Lai. 5G Multi-Slices Bi-Level Resource Allocation by Reinforcement Learning. – Mathematics, Vol. **11**, 2023, No 3, p. 760. DOI:10.3390/math11030760.
28. Schulman, J., F. Wolski, P. Dhariwal, A. Radford, O. Klimov. Proximal Policy Optimization Algorithms. – arXiv preprint arXiv:1707.06347, 2017.
29. Raffin, A., A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, N. Dormann. Stable-Baselines3: Reliable Reinforcement Learning Implementations. – Journal of Machine Learning Research, Vol. **22**, 2021, No 268, pp. 1-8.

30. P e r e y r a, G., L. I n g l´e s, C. R a t t a r o, P. B e l z a r e n a. Py5cheSim: a 5G Multi-Slice Cell Capacity Simulator. – In: Proc. of XLVII L. Conell, N. Laskaris, D. Blank, J. Wilson, S. Friedler, S. Luccioni, Eds. CodeCarbon: Estimate and Track Carbon Emissions from Machine Learning Catin American Computing Conference (CLEI'21), IEEE, Costa Rica, 2021, pp. 1-8.
https://github.com/lingleslogia/py5chesim
31. S c h m i d t, V., K. G o y a l, A. J o s h i, B. F e l d, L. C o n e l l, N. L a s k a r i s, D. B l a n k, J. W i l s o n, S. F r i e d l e r, S. L u c c i o n i. CodeCarbon: Estimate and Track Carbon Emissions from Machine Learning Computing. 2021. DOI: 10.5281/zenodo.4699491.
32. B r o c k m a n, G., V. C h e u n g, L. P e t t e r s s o n, J. S c h n e i d e r, J. S c h u l m a n, J. T a n g, W. Z a r e m b a. Open AI Gym. – arXiv preprint arXiv:1606.01540, 2016.
https://github.com/openai/gym.

Received: 20.07.2024; Second version: 26.08.2024; Accepted:29.08.2024 (fast track)