Intelligent Admission and Placement of O-RAN Slices Using Deep Reinforcement Learning

Nabhasmita Sen
Indian Institute of Technology Hyderabad, India
cs17resch11001@iith.ac.in

Antony Franklin A
Indian Institute of Technology Hyderabad, India
antony.franklin@iith.ac.in

***Abstract*—Network slicing is a key feature of 5G and beyond networks. Intelligent management of slices is important for reaping its highest benefits which needs further exploration. Focusing only on one goal as revenue maximization or cost minimization may not generate the highest profit for infrastructure providers in the long run. In this paper we jointly consider online admission and placement of Radio Access Network (RAN) slices with two objectives - a) maximizing revenue from accepting slices which are more profitable in the long run, and b) minimizing the cost to deploy them in Open RAN (O-RAN) enabled network by placing the slices efficiently. We formulate it as an optimization problem and propose a Deep Reinforcement Learning (DRL) based solution using Proximal Policy Optimization (PPO). We compare our model with a state-of-the-art DRL based admission control solution and a greedy heuristic. We show that our proposed solution can efficiently adapt to dynamic load conditions. We also show that the proposed solution results in better performance to maximize the overall profit for infrastructure providers in comparison to the baselines.**

***Index Terms*—Radio Access Network, Deep Reinforcement Learning, O-RAN, Energy Efficiency, Network Slicing.**

I. Introduction

With the introduction of network slicing, the physical network is treated as multiple logical networks that share underlying infrastructures. To maximize the profit of infrastructure providers, the long-term impact of admitting a slice needs to be evaluated. On the other hand, deployment of slices should be done efficiently, failing which the Operational Expenditure (OPEX) will grow high. Authors of [1]–[3] consider slice/user function placement in RAN to optimize processing and bandwidth resources. However, these works do not consider admission control of slices or their long-term impact. In [4], a utility maximization problem for wireless networks is proposed. The given solution maximizes utility for each request at a time but does not consider when some requests need to be dropped to accommodate more profitable upcoming requests. In [5], DRL based virtual network embedding is proposed, however different revenue factor of requests is not considered. Also, the cost of idle power consumption is not considered which is crucial to reduce OPEX in the network. In [6], [7], DRL based RAN slice admission control is proposed to maximize long-term revenue of infrastructure providers. However the cost of their deployment is not considered by the authors. The idle servers in the network cause a significant cost for infrastructure providers. So, to minimize OPEX this cost must be reduced. On the other hand, to maximize revenue, high profitable slices need to be admitted intelligently. However, not many works have considered these two things together. In [8], different revenue factors of services and idle cost of virtual machines are considered. Though, the proposed solution doesn't capture long-term impact of accepting a request. Different from the above works in this paper, we formulate an optimization model which maximizes profit in different load scenarios by considering two factors - a) long-term revenue from accepted slices, and b) idle cost of servers to deploy them. In case of high load, higher profit can be achieved by intelligently rejecting less profitable slices to accommodate more profitable slices. In case of low load, profit can be maximized by consolidating slices so that fewer servers are used to deploy them. The main contributions of this paper are as follows-

- We formulate an optimization problem for online admission and placement of RAN slices to maximize the long-term profit of infrastructure providers in O-RAN considering different revenue factors of slices and the idle cost of servers to deploy them.
- We present a Deep Reinforcement Learning (DRL) based solution to solve the optimization problem intelligently.
- We compare our proposed model with a state-of-the-art DRL based admission control solution and a greedy heuristic. We show that the proposed solution efficiently adapts to dynamic load scenarios and outperforms other strategies in maximizing the long-term profit for infrastructure providers.

II. System Model

We consider a hybrid cloud architecture (Fig. 1) as our system model. In O-RAN the base station is disaggregated into three components- a) Radio Unit (RU), which holds the lower physical layer functions of a base station, b) Distributed Unit (DU) that holds higher physical layer, Radio Link Control (RLC) layer and Medium Access Control (MAC) layer, and c) Centralized Unit (CU) that holds Packet Data Convergence Protocol (PDCP) layer and Radio Resource Control (RRC) layer. Due to this splitting of baseband processing, O-RAN components can be flexibly placed in separate locations. There are many RUs deployed in the network. Several edge clouds are there near the RUs where DU processing functions of slices are placed. A regional cloud is there where CU processing functions are placed. The regional cloud is further connected to the core network (not shown in Fig. 1). A link between

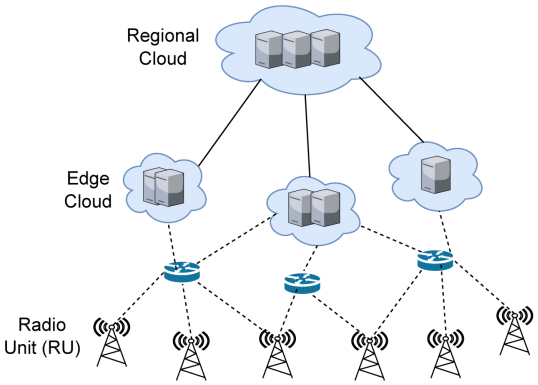


Fig. 1. RAN System Model.

a RU and edge cloud, between edge and regional cloud and between regional cloud and the core network are known as fronthaul, midhaul and backhaul respectively.

III. Problem Formulation

In this section, we propose our optimization model for online admission and placement of RAN slices. The deployment of a RAN slice considers placement of its CU, DU and corresponding links. To simulate a real network, we consider dynamic arrival and departure of slices with different holding time, slices with different revenue factors etc. The variables used in the formulation are defined in Table I. To maximize the profit, we consider the following two factors, a) revenue generated from accepting more profitable slices, b) cost of idle power consumption of servers in various clouds for deploying the slices. Total revenue generated by slice acceptance is described as follows,

$$REV = \sum_{s \in S} x_s r v_s h t_s \tag{1}$$

where x_s is a binary variable which denotes if the slice s is admitted or not. rv_s is the revenue of slice s in Monetary Unit (MU) per timestep whereas ht_s is the holding time of slice s. The cost for deploying the slices in terms of server usage is described as follows,

$$C = \sum_{t \in T} (\sum_{j \in E} u_{jt} c_j + \sum_{k \in R} v_{kt} c_k) \tag{2}$$

where u_{jt} denotes at particular timestep t, a server j in the edge clouds is being used or not. v_{kt} denotes the same for the server k in the regional cloud. c denotes the cost in MU per timestep for keeping a server *on* which is decided by infrastructure providers. To maximize the overall profit, we define our objective as follows,

$$Maximize : REV - C \tag{3}$$

where REV and C are defined in Eqn. (1) and (2) respectively. REV and C are normalized under same scale as both denote some amount in monetary unit for a period of time. We use a variable l_{st} which represents if a slice s is in service at time t, i.e.,

$$l_{st} = \begin{cases} x_s, & t_s \leq t < (t_s + ht_s) \\ 0, & \text{otherwise} \end{cases}$$

where t_s is the time when a slice request comes.

The optimization constraints are defined as follows,

a) Capacity constraints of server in edge/regional cloud: The total processing placed in any server of an edge/regional cloud can not exceed its capacity at any timestep t.

$$\sum_{s \in S} y_{sj} l_{st} du_s <= CE_j, \forall j \in E, \forall t \in T \tag{4}$$

$$\sum_{s \in S} z_{sk} l_{st} cu_s <= CR_k, \forall k \in R, \forall t \in T \tag{5}$$

b) Capacity constraint of transport links: The traffic flow in the fronthaul, midhaul, and backhaul cannot exceed their total capacity.

$$\sum_{s \in S} \sum_{j \in e} y_{sj} l_{st} \lambda_s^{ru} f_s <= F_{ru}^e, \forall e \in EC, \forall ru \in RU, \forall t \in T \tag{6}$$

$$\sum_{s \in S} \sum_{j \in e} y_{sj} l_{st} m_s <= M_e, \forall e \in EC, \forall t \in T \tag{7}$$

$$\sum_{s \in S} \sum_{k \in R} z_{sk} l_{st} b_s <= B, \forall t \in T \tag{8}$$

Here F, M, and B contain the total capacity of fronthaul, midhaul, and backhaul, respectively. f_s, m_s, and b_s are the transport requirement of the slice s for the same.

c) Delay constraint for slices: The total delay imposed by the fronthal, midhaul, and backhaul links used for routing a slice can not exceed the delay budget of that slice.

$$y_{sj} \delta 1_{ru_s e_j} + y_{sj} z_{sk} \delta 2_{e_j} + z_{sk} \delta 3 <= D_s \quad \forall s \in S, \forall j \in E, \forall k \in R \tag{9}$$

Here, $\delta 1$, $\delta 2$, and $\delta 3$ denote the delay of the fronthaul, midhaul, and backhaul links, respectively.

TABLE I
Notation

Notation	Description
x_s	1 if Slice s is admitted, 0 otherwise
rv_s	Revenue from slice s per unit time
ht_s	Holding time of slice s
c_v	Cost of server v per unit time
u_{jt}	1 if Server j in edge cloud is used at time t, 0 otherwise
v_{kt}	1 if Server k in regional cloud is used at time t, 0 otherwise
y_{sj}	Slice s is connected to edge server j at time t or not
z_{sk}	Slice s is connected to regional server k at time t or not
S	Set of all slices
CE	Capacity of servers in edge cloud
CR	Capacity of servers in regional cloud
E	Set of servers in edge clouds
R	Set of servers in the regional cloud
ru_s	RU of slice s
e_j	Edge cloud of server j
cu_s	Processing requirement in CU for slice s
du_s	Processing requirement in DU for slice s
D_s	Delay budget of slice s
EC	Set of edge clouds
RU	Set of Radio Units
λ_s^{ru}	1 if slice s is originated from RU ru
l_{st}	1 if a slice s is admitted and is in service at time t
T	Overall time duration.

d) Other constraints:
If a server in edge or regional cloud is used by any of the slice requests at time t then the server is declared as a used server.

$$u_{jt} >= y_{sj} l_{st}, \forall s \in S, \forall j \in E, \forall t \in T \tag{10}$$

$$v_{kt} >= z_{sk} l_{st}, \forall s \in S, \forall k \in R, \forall t \in T \tag{11}$$

A slice can be connected to at most one edge/regional cloud server.

$$\sum_{j \in E} y_{sj} <= 1, \forall s \in S \tag{12}$$

$$\sum_{k \in R} z_{sk} <= 1, \forall s \in S \tag{13}$$

If a slice s is admitted at time t_s, it must be connected to servers in the edge cloud and the regional cloud for its holding time period ht_s.

$$\sum_{i \in E} y_{sj} l_{st} = x_s, \forall s \in S, \forall t \in [t_s, t_s + ht_s) \tag{14}$$

$$\sum_{k \in R} z_{sk} l_{st} = x_s, \forall s \in S, \forall t \in [t_s, t_s + ht_s) \tag{15}$$

Additional binary constraints are defined as follows,

$$\begin{aligned} &y_{sj}, z_{sk}, u_{jt}, v_{kt}, x_s, l_{st} \in \{0, 1\}, \\ &\forall s \in S, \forall j \in E, \forall k \in R, \forall t \in T \end{aligned} \tag{16}$$

IV. Deep Reinforcement Learning Based solution

Deep Reinforcement Learning (DRL) has been proven to be effective in solving problems of high complexity without having any prior knowledge. Generally, to solve a problem with RL, the problem is modeled as a Markov Decision Process (MDP). We now describe the MDP model and the DRL algorithm to solve our optimization problem.

A. MDP Model Formulation

The state space, action space, and reward function of our MDP model are defined as follows-

1) State Space: We denote a state $s_t \in S$ as a tuple $\{R_t, E_t, B_t, SD_t, I_t\}$. R_t and E_t denote the remaining capacities in all the servers in regional and edge cloud, respectively at timestep t. B_t denotes the remaining capacities in all the transport links and I_t holds the information of the slice request to be processed at timestep t. SD_t contains the information when each of the servers in edge and regional cloud will sleep next based on the slices it is serving. The information of a slice request I_t is defined as $\{dr, ht, type, RU\}$. Here, dr is the data rate requirement of the slice, ht denotes the holding time of the slice request. $type$ represents the slice type from which its revenue and QoS requirement can be determined. RU denotes the RU from where the slice request is generated. From the information of a slice, its CU/DU processing and bandwidth requirements are calculated.

2) Action Space: In each time step, the RL agent selects an action a_t from a set of available actions A, which is known as action space. We denote an action as $\{R, E\}$ where R and E represent the selected servers in the regional and edge cloud for the placement of CU and DU processing of a slice, respectively. Action {-1,-1} denotes the action to reject a slice as no server is selected for its placement. From the selected server locations, the fronthaul and midhaul links for routing the slice are identified.

3) Reward Function: The reward function defines, given the current state s_t, how the RL agent will be rewarded when a particular action a_t is taken. The reward function $R(s_t, a_t)$ denoting the profit for infrastructure provider is formulated as the revenue from admitting a slice request minus the cost of using servers after deploying it.

$$R(s_t, a_t) = \begin{cases} REV_t - C_t, & \text{when slice is accepted} \\ 0, & \text{when slice is rejected} \end{cases}$$

where REV_t and C_t comes from Eqn.(1) and (2) respectively. REV_t is the revenue generated from the slice at timestep t. To find the value of C_t, we check the next sleeping time of the selected servers from SD_t stored in the state space. Let's denote the next sleeping time of server v as $next_v$. If slice s is placed in server v at timestep t, then the cost incurred in server v for slice s is denoted as,

$$C_t^v = \begin{cases} ht_s c_v, & \text{Case 1} \\ ((t + ht_s) - next_v) c_v, & \text{Case 2} \\ 0, & \text{Case 3} \end{cases}$$

where c_v denotes the idle cost of v^{th} server per unit time and ht_s is the holding time of slice s. Case 1 denotes the scenario when the slice is placed on a previously *off* server. Case 2 denotes when the slice is placed on a already *on* server and $next_v < (t + ht_s)$ i.e. the duration of the slice partially overlaps with previous slices in server v. Case 3 denotes the scenario when the slice is placed on a already *on* server and $next_v >= (t + ht_s)$, i.e. the duration of the slice fully overlaps with previous requests in server v. C_t will include the cost of both the selected servers in edge and regional cloud at timestep t. So, to maximize the profit, the RL agent tries to place a request in an already *on* server. This is how the agent learns to consolidate slices in different servers.

B. DRL Algorithm

To solve our problem, we consider an efficient DRL algorithm named Proximal Policy Optimization (PPO) [9]. During training, many invalid actions are often taken by the RL agent unnecessarily. We use invalid action masking to restrict the RL agent from taking an unwanted action which leads its convergence with less number of episodes. The implementation details are given in Section V-A.

V. Simulation and Results

A. Simulation setup

We have implemented the simulation environment in Python using OpenAI Gym [10]. We build a simulator to simulate

TABLE II
SIMULATION PARAMETERS

Simulation Parameters	Description
Number of Clouds	6 (1 Regional, 5 edge)
Total number of servers	8 servers
Number of server in regional cloud	3
Number of servers in edge cloud	1 in each cloud
Number of RUs	3 RUs
Slice-type	eMBB and URLLC
URLLC and eMBB Data-rate Requirement	1 & 3 Mbps
URLLC and eMBB Delay Requirement	2 & 4 ms
Slice holding time	10-20 timesteps
Arrival rate of slice	1-2 in each time step
Normalized Revenue of URLLC	1 MU per timestep
Normalized Revenue of eMBB	0.8 MU per timestep
Normalized Cost of server usage	1 MU per timestep
Number of slice requests	5-30 slices
Server capacity in Regional cloud	100 GOPS
Server capacity in Edge cloud	50 GOPS

the random arrival and departure of slices using Simpy [11] and interfaced it with our Gym environment. We incorporate changes in the implementation of PPO with action masking [12] which is based on Stable Baselines [13]. We mask the actions that violate the capacity constraint of servers in edge and regional clouds. All implementations are done in Google Colab [14] where Intel(R) Xeon(R) CPU@2.20GHz were assigned during runtime. We perform extensive simulations varying the number of slices, their arrival rate, and holding time in the network. The simulation parameters used are shown in Table II. We consider two types of slices (eMBB and URLLC) with different requirements and revenue factors. For simulation we assume that sufficient transport resources are there to support all slices regarding capacity and delay requirements. The processing requirement of CU and DU of slices are calculated based on the energy model in [15] and [1] and the bandwidth requirement is calculated based on [16].

B. Baseline Methods

We compare the proposed solution with following two methods -

1) A Greedy Heuristic (Heu): In this heuristic, a slice is always accepted if resource is available. For placement of the slice, a server is selected using first-fit strategy from a list of servers sorted based on their capacity in decreasing order.
2) A DRL based solution (RMAX): We consider the reward model of [7] as a comparison strategy where a slice is admitted based on its revenue factor to maximize the long-term revenue of infrastructure provider. In case of high load, it intelligently accepts high profitable slices. However, efficient deployment of slices is not considered here.

C. Evaluation Metrics

For the evaluation, we consider following metrics -

1) *Revenue*: Infrastructure providers gain income by accepting slice requests, which is termed as revenue.
2) *Cost for using servers*: To deploy the admitted slice requests various servers are switched *on* in edge and regional clouds which incurs this cost.
3) *Total Profit of infrastructure providers*: This indicates the total profit generated from serving the slice requests that is revenue minus the cost of server usage.

All metrics defined above can be expressed in any monetary unit (MU).

D. Result and Analysis

In this section, we show comparison among different strategies. We consider the two load scenarios for performance evaluation-

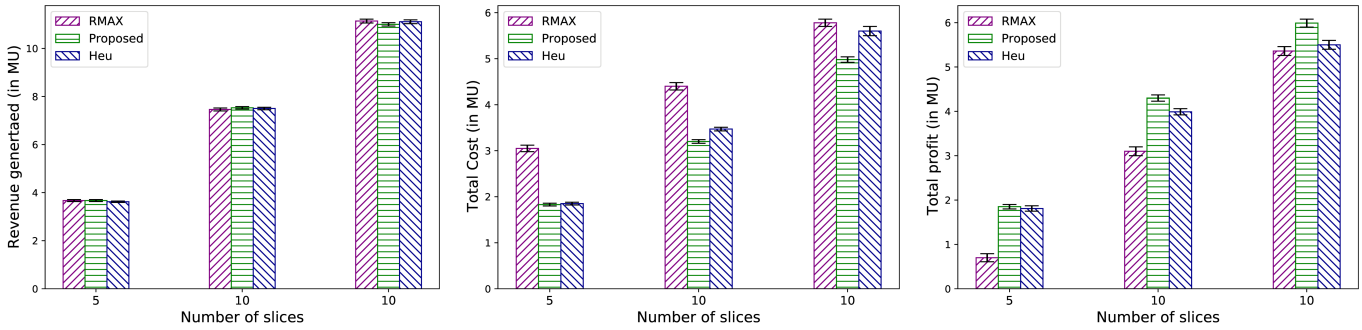
1) Low load: In this case, the arrival rate (α) is 1 slice per timestep and the holding time (ht) of the slices varies in range of [10, 15] timesteps. In this case, sufficient resources are there to accept all slice requests.
2) High load: In this case, the arrival rate (α) is 2 slices per timestep and the holding time (ht) of the slices varies in the range of [10, 20] timesteps. In this case, the available resource is not sufficient to accept all slice requests and some of the requests need to be rejected.

We randomly generate slice requests based on the load scenario and report the profit, cost, and revenue with a 95% confidence interval for each experiment repeated 100 times.

1) Total Revenue generated from accepting slices: In Fig. 2a and 3a, we show the revenue generated using each strategy for low and high load scenarios, respectively. In case of low load, the arrival rate and holding time for slices are low. So, network resources are always available for the placement of slices. We see that all strategies achieve similar revenue as there is no competition for resources and all requests can be admitted. But in the case of high load (Fig. 3a), as the number of requests grows and most of the server resources are used up, the heuristic generates lower revenue. This is because when a request comes it always accepts it if resource is available since it cannot evaluate the long-term profit of accepting any slice. However, the DRL based algorithms intelligently accept those slices which are more profitable in the long term.

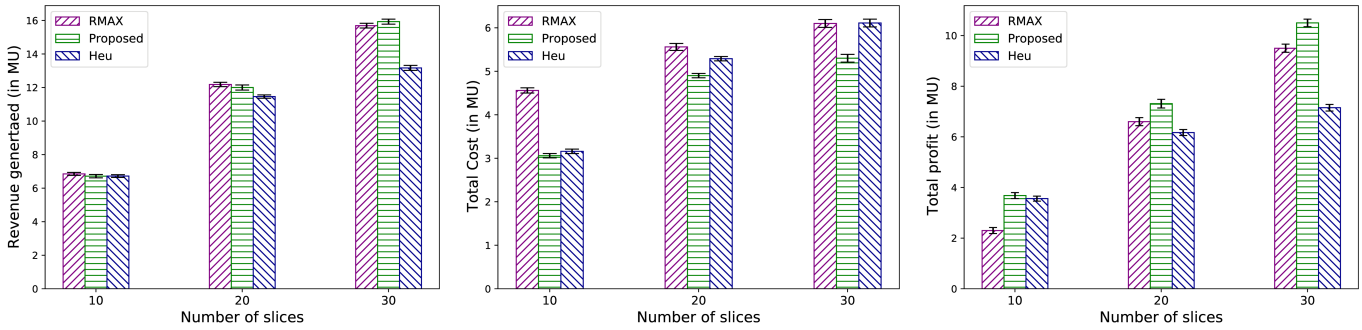
2) Cost for server usage: In Fig. 2b and 3b, we compare the cost from server usages in the different clouds for deploying the slices in low and high load respectively. We observe that in the case of low load, even though the revenue generated from slice acceptance is almost similar in all cases, the proposed solution incurs the lowest cost as it is able to consolidate the CU and DU processing in different servers more efficiently. Since heuristic cannot capture the dynamism of the network, it is not able to consolidate RAN functions efficiently. On the other hand, RMAX is agnostic of deployment cost and focuses mainly on revenue maximization. Hence, it also does not reduce the cost of server usage efficiently.

3) Total Profit for infrastructure providers: The main objective of our work is to maximize the overall profit of the infrastructure providers under different load scenarios. In Fig. 2c and 3c, we show the total profit generated for all strategies with the varying number of slice requests in low and high load respectively. We observe that the proposed solution is able to achieve maximum profit in all the scenarios. In case of low load, the revenue generated by all strategies is similar as all



(a) Total Revenue (b) Total Cost (c) Total Profit

Fig. 2. Comparison of strategies in case of low load (α=1, $ht \in [10, 15]$)



(a) Total Revenue (b) Total Cost (c) Total Profit

Fig. 3. Comparison of strategies in case of high load (α=2, $ht \in [10, 20]$)

requests are accepted in each case. However, the proposed solution incurs lower cost than the other two strategies due to its efficient placement of slices which results in maximum profit. In case of high load, the revenue generated by RMAX and the proposed solution is more than the heuristic as they selectively reject low profitable slices. However, RMAX incurs a higher cost than the proposed solution as it does not consider intelligent slice placement. Hence, for this case also, we can see that the proposed solution generates maximum profit by taking advantage of intelligent admission and placement jointly.

VI. Conclusion and Future Work

In this paper, we addressed the problem of infrastructure providers' profit maximization in O-RAN considering different revenue factors of slices and idle cost of servers in various clouds. We formulated an optimization model and a DRL based solution to maximize the desired long-term profit. Using a simulation study, we showed that the proposed solution is able to make an appropriate decision that can optimize resources in different load scenarios. We also showed that our model outperforms other strategies in maximizing long-term profit of infrastructure provider. As future work, we want to consider both RAN and core functions while intelligently placing slices in the network.

References

[1] S. Matoussi, I. Fajjari, S. Costanzo, N. Aitsaadi, and R. Langar, "A user centric virtual network function orchestration for agile 5g cloud-ran," in *2018 IEEE International Conference on Communications (ICC)*, 2018.

[2] B. O. et. al., "Sliced-ran: Joint slicing and functional split in future 5g radio access networks," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019.

[3] A. Alabbasi, X. Wang, and C. Cavdar, "Optimal processing allocation to minimize energy and bandwidth consumption in hybrid cran," *IEEE Transactions on Green Communications and Networking*, 2018.

[4] F. Esposito and F. Chiti, "Distributed consensus-based auctions for wireless virtual network embedding," in *2015 IEEE International Conference on Communications (ICC)*, 2015.

[5] M. Dolati, S. Hassanpour, M. Ghaderi, and A. Khonsari, "Deepvine: Virtual network embedding with deep reinforcement learning," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2019.

[6] M. R. Raza, C. Natalino, P. Öhlen, L. Wosinska, and P. Monti, "Reinforcement learning for slicing in a 5g flexible ran," *Journal of Lightwave Technology*, vol. 37, no. 20, pp. 5161–5169, 2019.

[7] S. Bakri, B. Brik, and A. Ksentini, "On using reinforcement learning for network slice admission control in 5G: Offline vs. online," *International Journal of Communication Systems*, May 2021.

[8] M. Golkarifard, C. F. Chiasserini, F. Malandrino, and A. Movaghar, "Dynamic vnf placement, resource allocation and traffic routing in 5g," *Computer Networks*, 2021.

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, 2017.

[10] G. Brockman and et al., "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[11] "Simpy," "https://simpy.readthedocs.io/en/latest/".

[12] C. Tang, C. Liu, W. Chen, and S. You, "Implementing action mask in proximal policy optimization (ppo) algorithm," *ICT Express*, 2020.

[13] A. Hill and et al., "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.

[14] "Google Colab," https://colab.research.google.com/.

[15] A. Garcia-Saavedra, G. Iosifidis, X. Costa-Perez, and D. J. Leith, "Joint optimization of edge computing architectures and radio access networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 11, pp. 2433–2443, 2018.

[16] S. C. Forum, "Small cell virtualization functional splits and use cases," Small Cell Forum, Tech. Rep., 2016.