DQN-based Intelligent Application Placement with Delay-Priority in Multi MEC Systems

Juan Sebastian Camargo*, Estefanía Coronado*†, Claudia Torres-Pérez*,
Javier Palomares* and Muhammad Shuaib Siddiqui*
*i2CAT Foundation, Barcelona, Spain;
Email: {juan.camargo, estefania.coronado, claudia.torres, javier.palomares, shuaib.siddiqui}@i2cat.net
†High-Performance Networks and Architectures, Universidad de Castilla-La Mancha, Albacete, Spain;
Email: estefania.coronado@uclm.es

***Abstract*—In 5G Multi-access Edge Computing (MEC) is critical to bring computing and processing closer to users and enable ultra-low latency communications. When instantiating an application, selecting the MEC host that minimizes the latency but still fulfills the application's requirements is critical. However, as future 6G networks are expected to become even more geo-distributed, and handled by multiple levels of management entities, this labor becomes extremely difficult and Machine Learning (ML) is meant to be a native part of this process. In this context, we propose a Reinforcement Learning model that selects the best possible host to instantiate a MEC application, looking to minimize the end-to-end delay while fulfilling the application requirements. The proposed ML method uses Deep Q-Learning through several stages of environment state, taking an action and rewarding the model when it chooses correctly and penalizing it otherwise. By modifying the reward incentives, we have successfully trained a model that chooses the best host possible delay-wise on a multi-level orchestration scenario, while meeting the applications' requirements. The results obtained via simulation over a series of MEC scenarios show a success rate of up to 96%, optimizing the delay in the long term.**

***Index Terms*—Reinforcement Learning, Application Placement, DQN, MEC, 5G, 6G, Distributed Orchestration**

I. Introduction

5G introduces Ultra-Reliable Low-Latency Communications (URLLC) as a fundamental use-case that aims to provide services with a tight operational window and high-end restrictions in terms of latency and reliability. URLLC services must provide almost instantaneous access, with almost zero perceived delay in human-oriented services, such as healthcare, entertainment, and safety. To provide applications with this extremely low delay, one solution is to place them in servers as close to the user as possible, in what is known as Multi-access Edge Computing (MEC). A MEC system is defined as a collection of nodes, known as MEC hosts, that can provide computation and virtualization capabilities, and a centralized entity that controls said hosts, known as a MEC Orchestrator (MEO). The MEC host is responsible for installing the URLLC application that will serve the radio nodes connected to it, limiting the use of the URLLC application to a geographically defined area. It is expected that 5G and 6G networks rely heavily on URLLC, creating highly distributed geographical zones, that eventually would need an intelligent application-placement system that can manage where the delay-constrained applications would be efficiently installed.

Being able to choose a proper MEC host that satisfies the application requirements, delay-wise, is fundamental when managing delay-sensitive applications. This is an already complex problem when dealing with a single Orchestrator and various MEC hosts, so considering finding the most suitable host to instantiate an application in an architecture with several MEOs and their corresponding MEC hosts can be an arduous and nontrivial task. Traditionally, this placement problem can be solved using a greedy algorithm to find the best suitable location [1] or using fuzzy logic giving priority to a specific network parameter [2].

The main inconvenience with the previous solving methods is their inability to generalize, a handicap in the ever-changing 5G and future 6G networks. In this respect, Reinforcement Learning (RL) has been proven a valuable asset in solving this type of problem. In particular, RL can use the current network status and propose the most suitable MEC host for the instantiation of the application. RL Systems can handle a high number of parameters and adapt to handle unknown states of the network without having been exposed to them before. Consequently, in this paper, we propose a method that uses RL as a tool for the delay-wise instantiation of lag-constrained applications in 5G or 6G environments across distributed MEC systems. Additionally, we test our system following the delay-constrains suggested by ETSI for URLLC use cases. Finally, we present the results showing the advantages of using this RL method in a simulated environment.

The paper is structured as follows: Sec. II presents the related work. Sec. III describes the system architecture envisioned in this work, while Sec. IV describes the problem formulation. Sec. V introduces the RL algorithm proposed to solve the placement problem, minimizing delay in multi-level orchestration scenarios. Sec. VI discusses the performance evaluation. Finally, Sec. VII draws some concluding remarks.

II. Related Work

Application placement at the edge is a sub-problem studied in the context of resource allocation. Application and service placement has been analyzed at several levels (from radio access to the cloud) in the last few years, employing several

methods, such as numerical analysis, optimization problems, and other algorithms [1], [2], [3], [4]. The paper in [2] indicates a Quality of Experience (QoE)-aware application placement, through fuzzy logic, that prioritizes requests and classifies Fog instances according to user expectations. The priority is based on access rate, required resources and processing time. Conversely, the classification considers round trip time, resource availability and processing speed. Meanwhile, the work in [1] uses a greedy algorithm with a multi-stage stochastic programming model to minimize the total execution cost of user applications, considering computation, communication, and relocation cost. By contrast, the works in [3], [4] intend to optimize time factors in Internet of Things (IoT) scenarios. The real-time processing of applications is considered in [3] with an approximation scheme in terms of delay and bandwidth requirements. By contrast, the authors of [4] focus on the mathematical model computation of finite state machines optimizing completion time, energy consumption and economic cost.

Many works have relied on ML to solve the previous problem and, in particular, RL, as it has been proven to develop effective performance in scenarios where the characteristics and variations of the environment are not known in advance. In this context, the works focused on IoT tend to firstly transform the problem into an offloading problem with similar requirements [5], [6], [7]. A Deep RL (DRL) approach is studied in [5] to address task scheduling of fog-based IoT applications. The algorithm considers transmission, propagation, waiting, and execution delay to achieve a trade-off between them. The authors of [6] propose a distributed DRL model for IoT in Edge and Fog Computing. The applications are organized in Directed Acyclic Graphs (DAGs). Using the same ML aproach, the paper in [8] proposes the placement of Virtualized Network Functions (VNFs) in 5G networks using DRL. This approach pursues the optimization of the resource utilization and the minimization of the operational costs of 5G networks while satisfying the Quality of Service (QoS) requirements of the network. A similar work for Edge Computing is described in [7]. DRL is employed for data-intensive application deployment, aiming to reduce users' latency and monetary costs for application service providers.

Similarly, a resource allocation framework for IoT applications is designed in [9]. It employs a novel two-stage DRL scheme maximizing users' QoE, and tunes the application requirements to align with the available edge resources. The alignment is performed by selecting the QoS range that can be satisfied by the available resources. The model considers packet loss rate, packet error rate and latency. The authors of [10] perform application placement in Fog Computing, considering user QoE. It is a model-based Value expansion to determine long-term Q-value. The research in [11] studies the application placement problem based on the ETSI MEC architecture to balance the computational load. It allows the MEC orchestrator to decide on the platform to host MEC applications. However, the proposed work relies on an Integer Linear Programming method instead of an RL algorithm.

The majority of the research examined is focused on IoT application placement while reducing delay parameters in Fog or Edge Computing environments. Different from previous works, in this paper we solve the application placement problem by prioritizing delay minimization in an ETSI MEC compliant architecture while considering a novel multi-MEC orchestration environment, aware of the available computational resources. We propose a multi-level architecture that employs a primary orchestrator in charge of several MEC orchestrators, as opposed to the examined studies, which mainly do the analysis on just one level. The model developed leverages a well-tested deep RL approach that is based on the application's requirements rather than its nature.

III. System architecture

The system architecture envisioned in this work is based on the MEC reference architecture proposed by ETSI [12]. A MEC system is composed of various distributed nodes, known as MEC hosts. Each MEC host has computing, storage, RAM, and network resources, which are employed to instantiate applications, seeking to decrease the overall user's delay. A MEC system also comprises an entity managing the MEC hosts, known as MEO. In future 5G and beyond and 6G networks, it is expected to have topologies with multiple MEC systems, which would increase the complexity of choosing a host to meet the applications' requests, also considering that each of them may belong to a different administrative domain. In this scenario, we introduce a centralized control entity, on top of the MEC architecture, and responsible for managing all the MEC systems. This entity, named Multi-MEC Orchestrator (MMO) can access all the information from its subordinated MEC systems, and based on those parameters, can choose the best MEC host to instantiate an application.

Given the currently restricted delay times needed for URLLC applications and the high number of parameters involved, a human-controlled network is not only unfeasible but also error-prone. Additionally, with the network's heterogeneity, and the time-changing environments, having a fixed pre-configuration to manage the network is also not an option as it would rapidly turn itself obsolete. However, that same environment of high performance and an elevated number of parameters allows us to introduce an RL model that works together with the MMO to successfully fulfill the network's requirements. A high-level view of the architecture can be seen in Fig 1. The application instantiation requests come first from the MMO, which searches for the possible MEC host that complies with the application requirement, prioritizing the fulfillment of the delay constraints of the application. Having selected the best possible MEC host, the MMO forwards the request to the corresponding MEC orchestrator, which instantiates the application.

IV. Problem Formulation

This section presents the mathematical formulation of the application placement problem considering multiple MEC systems. The method uses the information available in the

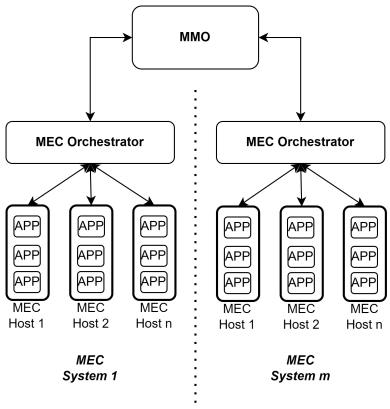

Fig. 1. High-level view of the multi MEC architecture.

TABLE I
SUMMARY OF THE SYSTEM MODEL PARAMETERS.

Notation	Description
$\mathcal{N}$	Number of MEC systems composing the network, $n \in [1, \mathcal{N}]$
$\mathcal{M}$	Number of nodes of each MEC system, $m \in [1, \mathcal{M}]$
$\mathcal{S}$	States of the network, $s \in \mathcal{S}$
$\vec{\mathcal{A}}^s_{n,m}$	Available resource vector associated to any state $s \in \mathcal{S}$ in a node $m \in \mathcal{M}$ of the MEC system $n \in \mathcal{N}$ $\vec{\mathcal{A}}^s_{n,m} = \left[A^{CPU^s_{n,m}}, A^{RAM^s_{n,m}}, A^{ST^s_{n,m}}, A^{D^s_{n,m}}\right]$
$\vec{\mathcal{C}}_{n,m}$	Maximum capacity of each computational resource in each MEC host $m \in \mathcal{M}$ of the MEC system $n \in \mathcal{N}$ $\vec{\mathcal{C}}_{n,m} = [CPU_{n,m}\|_{max}, RAM_{n,m}\|_{max}, ST_{n,m}\|_{max}]$
$\mathcal{I}$	Set of application instantiation requests, $i \in [1, \mathcal{I}]$
$\vec{R}_i$	Resource request vector with computational resources and maximum delay time $\vec{R}_i = \left[R^{CPU}_i, R^{RAM}_i, R^{ST}_i, R^{D}_i\right]$
F_s	Reward function of any given state of the network $s \in \mathcal{S}$ $F_s = [F_1, F_2, ..., F_n]$

MMO regarding the capacity of all the nodes of each MEC system and its current end-to-end delays. It is assumed that the MMO has all network's information and can share it under requirement to the MEC host choosing method.

Let $\mathcal{N}$ be the total number of MEC systems in our network architecture and $\mathcal{M}$ be the number of nodes in each MEC system. Each $m \in \mathcal{M}$ has the corresponding current resource availability of the evaluated parameters: i.e., CPU (CPU), RAM (RAM), storage (ST) and delay parameters (D) associated with any given state $s \in \mathcal{S}$. For each s there is a capacity vector showing the currently available $\vec{\mathcal{A}}$ resources of each MEC system, such as $\vec{\mathcal{A}}^s_{n,m} = [A^{CPU^s_{n,m}}, A^{RAM^s_{n,m}}, A^{ST^s_{n,m}}, A^{D^s_{n,m}}]$. This availability vector is updated whenever a new application is instantiated in any MEC system. Additionally, there is a maximum capacity vector C, containing the maximum computational capabilities of each of the resources in each MEC host, such as $\vec{C}_{n,m} = [CPU_{n,m}|_{max}, RAM_{n,m}|_{max}, ST_{n,m}|_{max}]$.

Let $\mathcal{I}$ be the set of application instantiation requests. Each instantiation $i \in \mathcal{I}$ contains a request vector (R) composed by the demanded computational resources (CPU, RAM, and storage) and the maximum application's delay budget, such as $\vec{R}_i = [R^{CPU}_i, R^{RAM}_i, R^{ST}_i, R^{D}_i]$. Given a reward function F, the reward values of any state s are $F_s = [F_1, F_2, ..., F_m]$. Table I outlines all the model's parameters.

The main objective of the host-choosing method is to find an optimum MEC host with enough capacity to meet the application requirements while fulfilling its delay constraints. To formulate this problem, the following two binary variables have been defined. First, we propose $x_{n,m}$, which indicates if a node m in a MEC system n is active or not:

$$x_{n,m} = \begin{cases} 1 \text{ , if node } m \text{ in MEC systems } n \text{ is active} \\ 0 \text{ , otherwise} \end{cases} \quad (1)$$

We also define $y^i_{n,m}$, which indicates if the request $i \in \mathcal{I}$ has been successfully deployed in the node m of MEC system n:

$$y^i_{n,m} = \begin{cases} 1 \text{ , if request } r \text{ is successfully placed and deployed} \\ 0 \text{ , otherwise} \end{cases} \quad (2)$$

The main objective is to maximize the reward function, which minimizes the delay, by selecting the MEC system that generates the highest value for a given request, such as:

$$max: \quad F_s \quad \forall s \in \mathcal{S} \quad (3)$$

Additionally, we would like to maximize the number of application requests deployed successfully, such as:

$$max: \quad \sum_{i \in \mathcal{I}} y^i_{n,m} \; \forall m \in \mathcal{M} \; \forall n \in \mathcal{N} \quad (4)$$

The constraints of the proposed problem are the following:

- The computational resources demand of all requests allocated in any node $m \in \mathcal{M}$ of any MEC system $n \in \mathcal{N}$ cannot exceed its total capacity in terms of CPU (CPU), RAM (RAM), storage (ST):

$$\sum_{\forall i \in \mathcal{I}} R^{CPU}_i \leq A^{CPU^s_{n,m}} \cdot x_{n,m} \;\; \forall m \in \mathcal{M}, \forall n \in \mathcal{N}, \forall s \in \mathcal{S} \quad (5)$$

$$\sum_{\forall i \in \mathcal{I}} R^{RAM}_i \leq A^{RAM^s_{n,m}} \cdot x_{n,m} \;\; \forall m \in \mathcal{M}, \forall n \in \mathcal{N}, \forall s \in \mathcal{S} \quad (6)$$

$$\sum_{\forall i \in \mathcal{I}} R^{ST}_i \leq A^{ST^s_{n,m}} \cdot x_{n,m} \;\; \forall m \in \mathcal{M}, \forall n \in \mathcal{N}, \forall s \in \mathcal{S} \quad (7)$$

- The application delay parameter cannot exceed the value demanded in any request:

$$R^D_i \leq A^{D^s_{n,m}} \;\; \forall i \in \mathcal{I}, \forall m \in \mathcal{M}, \forall n \in \mathcal{N}, \forall s \in \mathcal{S} \quad (8)$$

In addition, given the dependency between the variables x and y, a relationship between them has to be added, as follows:

$$\text{if} \sum_{\forall m \in \mathcal{M}} \sum_{\forall n \in \mathcal{N}} x^i_{n,m} = 0 \Rightarrow y^i_{n,m} = 0 \Rightarrow F_s = 0 \quad (9)$$

V. DELAY-AWARE REINFORCEMENT LEARNING PLACEMENT ALGORITHM

The control method presented in this paper uses a RL technique mixed with a deep neural network predictor, where an agent is exposed to an environment with a pool of actions regarding placement decisions. If the agent chooses to

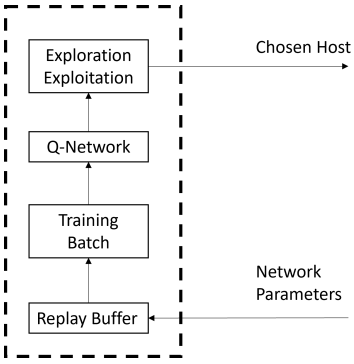

Fig. 2. Deep Q-Learning cycle.

place an application in a MEC host that completely fulfills the requirements, especially the delay characteristics of the application to instantiate, it receives a high reward value. On the other hand, if the agent chooses a MEC host that does not fulfil the application requirements, it is penalized with a negative reward. The reward function symbolizes the instant-value that the model obtains by selecting a specific action to a given state. If we continue generating reward values for each pair of action-states, we obtain a long-term value for each one named Q. As this process is computationally expensive, we use a neural network to predict an estimated Q value, which is known as Deep Q-Learning [13].

The model determines the best action by obtaining the Q value followed by an exploration/exploitation step. Exploration allows the model to take random actions initially and then focus on exploitation where the best actions are chosen. This is important when the reward is not instant and the model needs to explore other options for long-term benefit. The complete process is shown in Fig 2

An index is associated to each of the computational resources (RAM, CPU, and storage), which represents how much free space there would be in each of the hosts after the application would have been instantiated. The value is proportional for the free space left in the host, meaning that the normalized value is closer to 1 if there is plenty of free space and closer to zero otherwise. Eq. 10 depicts how the value of the index is calculated. It takes into consideration the subtraction of the values of the request $\vec{R}_i$ and the resource availability $\vec{A}^s_{n,m}$ vectors.

A higher value shows that the MEC host is more suitable for the instantiation than another host with a lower number. In explanation, a higher number prevents selecting a low-resource host able to instantiate the application and favors another host (also with enough capacity for the instantiation) with more resources available.

$$CPU_{index} = \frac{A^s_{n,m}|_{CPU} - R_i|_{CPU}}{C_{n,m}[CPU|_{max}]} \tag{10}$$

$$RAM_{index} = \frac{A^s_{n,m}|_{RAM} - R_i|_{RAM}}{C_{n,m}[RAM|_{max}]} \tag{11}$$

$$ST_{index} = \frac{A^s_{n,m}|_{ST} - R_i|_{ST}}{C_{n,m}[ST|_{max}]} \tag{12}$$

Additionally, after having obtained all the parameter indexes of the three capacity indicators, we proceed to obtain the delay index separately. The delay here refers to the end-to-end delay from the UE to the MEC host, including the transmission, processing and queueing delay. This index is bounded by the maximum and minimum delays that the model uses, in the way described in Eq. 13.

$$D_{index} = \begin{cases} 0 & if \qquad A^{D^s_{n,m}} < R^D_{m,n} \\ \frac{D_{max} - R^D_{m,n}}{D_{max} - D_{mins}} & otherwise \end{cases} \tag{13}$$

Having found the four indexes, the next step is to make a summation among them and then divide them by the delay index. As the normalized delay is bounded between zero and one, the lower the delay index, the higher the value obtained in the reward, encouraging the RL model to select this low delay options. Finally, with these four indexes we obtain the function $f^s_{m,n}(R^D_{m,n})$, described in Eq. 14, that is used in the reward function proposed in this paper in Eq. 15.

$$f^s_{m,n}(R^D_{m,n}) = \frac{CPU_{index} + RAM_{index} + ST_{index}}{D_{index}} \tag{14}$$

$$r^s = \begin{cases} 0 & \sum_{\forall n \in \mathcal{N}} \sum_{\forall m \in \mathcal{M}} x_{n,m} = 0 \\ 0 & \vec{\mathcal{A}}^s_{n,m} < \vec{R}^s_i \ \forall n \in \mathcal{N}, \forall m \in \mathcal{M}, \forall s \in \mathcal{S} \\ f^s_{m,n}(R^D_{m,n}) & otherwise \end{cases} \tag{15}$$

This reward function first filters the scenarios in which there are no active nodes present in the architecture and assigns a total reward of 0. Conversely, the next step is to check if the current capacity of the hosts, in terms of capacity and delay, is enough to fulfill the application requirements. Again, if there is no capacity in any of the nodes, the reward is zero, as there is no action that can provide a satisfactory solution to the instantiation petition. Finally, if these two filters are passed successfully, the reward value is obtained based in the function presented in Eq. 14.

Eq. 14 provides a simple yet useful reward function. It is bounded in the range [0,1], generating rewards closer to 1 when the host is empty and closer to 0 in opposite circumstances. This continuous range help the RL model to learn easier as for subtle changes between one state and its respective adjacent state there is a corresponding subtle change in their reward functions.

VI. Performance Evaluation

A. Training Methodology

The training follows the architecture described in Section III, having two MEC systems connected to a centralized MMO. Each MEC system manages two MEC hosts, each of

which has associated three computing parameters, i.e., RAM, CPU and storage, and an additional delay parameter. Both the training and evaluation is performed via simulation, which is based on a series of episodes where the RL model interacts with the environment and takes an action, being rewarded accordingly, and then transitions to the next episode.

According to [14], the maximum end-to-end latency for URLLC varies between 10 and 60 ms. Therefore, the delay constraints are defined randomly in the [10-60 ms] range. Note that the delay parameter refers to the end-to-end delay. The application requirements, including the delay budget, are generated randomly following a normal distribution. The simulation uses values in the [0,100] range for the capacity parameters and scales the delay values to match the same range. Throughout all the episodes, the maximum capacity of all nodes is fixed at 100. Each episode represents a new application instantiation request. At the beginning of each episode, the capacity, and delay of all the MEC hosts is randomly assigned, including a historical value that simulates the already installed applications in said MEC host.

The environment selects a random action and obtains a reward (as described in Eq. 15). With time, the selection of actions becomes less random and more focused on choosing the action that generates the highest reward. This exploration-exploitation trade-off is necessary for the system to explore different options and identify those that generate the highest reward in the long-term. The training process produces new application requests until all MEC hosts are full and no more instantiations are possible.

B. Neural Network Configuration

The neural network used by the RL model consists of four layers: one input layer, two hidden layers and one output layer. The input layer uses as input the number of parameters per MEC hosts times the number of MEC hosts working under the MMO, plus the parameters of the application request. The two hidden layers are fully connected and have 16 neurons each, using a Rectified Linear Unit (ReLU) as the activation function. The output layer of the neural network contains an equal number of nodes as the total number of MEC hosts, with each output value representing the probability of selecting the corresponding MEC host.

C. Performance Evaluation

We propose two scenarios to test the method's capabilities. *Scenario 1* is composed of two MEC systems, each of them managing two MEC hosts, for a total of four MEC hosts. *Scenario 2* envisions four MEC systems, each of them containing two MEC hosts, for a total of eight MEC hosts. Following the previous methodology, the obtained results from the training process in these environments is shown in Fig. 3 for *Scenario 1* and Fig. 5 for *Scenario 2*. Both figures describe the accuracy of the last 100 episodes throughout the training process. The accuracy is calculated by comparing the model's output against the optimal answer obtained from the reward function. It can be observed that *Scenario 1* reaches a plateau around 1000 episodes, whereas *Scenario 2* does it around 2000 episodes. For *Scenario 1*, the final training accuracy is around 96%, slightly more than 91% value for *Scenario 2*.

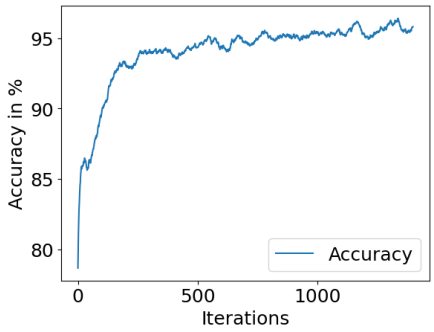

Fig. 3. Average accuracy through training for *Scenario 1*.

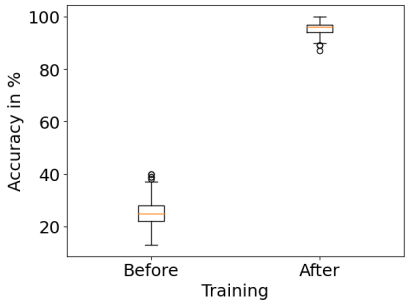

Fig. 4. Accuracy before and after training for *Scenario 1*.

During inference, a set of random application instantiation requests is generated at the MMO. Therefore, once received, the MMO forwards it to the already trained RL model, which must decide the best MEC host to instantiate the application across all available ones in all MEC systems. Once the model has taken a decision, the network parameters are updated with a new host status, a random delay time for the hosts is chosen, and the cycle is restarted. This cycle procedure is repeated 100 times for each scenario, obtaining an average accuracy of 95% and 92% for *Scenario 1* and *Scenario 2*, respectively.

Additionally, we evaluate *Scenario 1* and *Scenario 2* using the same inference process as previously, but this time we choose the applications from a pre-defined list rather than randomly, wanting to evaluate the model's behavior before and after training. To do so, we execute inference in an entirely untrained model, sending the application list through the untrained model 100 times and determining the overall accuracy level after each pass. The identical application set is then run 100 times over a completely untrained model. We obtain an average accuracy of 23% for the untrained model in *Scenario 1*. However, considering that it is a four-host architecture, the untrained RL model is choosing the host totally at random (a random choice for 4 hosts is 25%). On the other hand, testing the fully trained model using the same application list, we obtain around 93% of accuracy rate. The training accuracy before and after the training can be

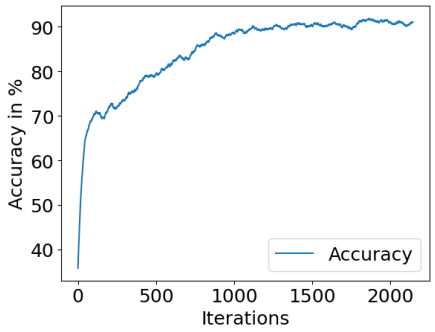

Fig. 5. Average accuracy through training for *Scenario 2*.

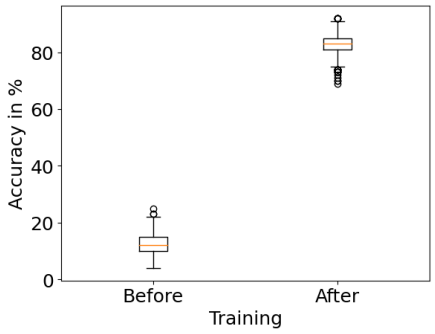

Fig. 6. Accuracy before and after training for *Scenario 2*.

seen in Fig. 4, including the 95% confidence interval for *Scenario 1*. We performed the same inference process using the architecture in *Scenario 2*, with an average accuracy level of 11% (a complete random choice for 8 hosts is 12.5%) before and 83% after training, respectively, as shown in Fig. 6. It can be observed that, although both scenarios reached a comparable accuracy level, it takes almost the double of episodes for *Scenario 2* to converge compared with *Scenario 1*. Considering that there are fewer hosts in *Scenario 1*, the RL model was able to cover a sufficient state space during training to function accurately. However, as the number of nodes increases, the required state space to achieve the same values also increases proportionally. Similarly, by having been able to explore a larger portion of the state space, the level of effectiveness of the model could potentially be higher. However, the accuracy presented by doubling the number of hosts is within the range of use in a demanding environment, showing that the RL model could be scaled successfully.

VII. Conclusion

In this paper, we have presented a RL method that allows a MMO to choose the best host delay-wise for application instantiation in multi-level MEC orchestration. The proposed DQN model can predict with 96% accuracy the best possible host destination for the application. Additionally, even though the model was designed with end-to-end delay in mind as the key parameter, together with the awareness of the infrastructure status, we aim to expand it and use other network parameters. Thanks to the flexibility of RL, the main variable could be adjusted, and any other technical parameter could be used instead. As another line of future work, we aim to improve the accuracy levels by modifying the neural network's parameters and perform an exhaustive fine-tune of the model.

Acknowledgment

This work has been performed in the framework of the European Union's Horizon 2020 AI@EDGE project co-funded by the EU under grant agreement No 101015922. The authors would also like to acknowledge CERCA Programme / Generalitat de Catalunya for sponsoring part of this work. This work has been also supported by the EU "NextGenerationEU/PRTR", MCIN and AEI (Spain) under project IJC2020-043058-I, by the grant ONOFRE-3 PID2020-112675RB-C43 funded by MCIN/AEI/10.13039/501100011033 and also by the Horizon European project NANCY under grant agreement 101096456.

References

[1] H. Badri, T. Bahreini, D. Grosu, and K. Yang, "A Sample Average Approximation-Based Parallel Algorithm for Application Placement in Edge Computing Systems," in *Proc. of IEEE IC2E*, Orlando, FL, USA, Apr. 2018, pp. 198–203.

[2] R. Mahmud, S. N. Srirama, K. Ramamohanarao, and R. Buyya, "Quality of Experience (QoE)-aware placement of applications in Fog computing environments," *Journal of Parallel and Distributed Computing*, vol. 132, pp. 190–203, Oct. 2019.

[3] R. Yu, G. Xue, and X. Zhang, "Application Provisioning in FOG Computing-enabled Internet-of-Things: A Network Perspective," in *Proc. of IEEE INFOCOM*, Honolulu, USA, Apr. 2018, pp. 783–791.

[4] N. Mehran, D. Kimovski, and R. Prodan, "MAPO: A Multi-Objective Model for IoT Application Placement in a Fog Environment," in *Proc. of ACM IoT*, Bilbao, Spain, Oct. 2019, pp. 1–8.

[5] P. Gazori, D. Rahbari, and M. Nickray, "Saving time and cost on the scheduling of fog-based IoT applications using deep reinforcement learning approach," *Future Generation Computer Systems*, vol. 110, pp. 1098–1115, Sep. 2020.

[6] M. Goudarzi, M. S. Palaniswami, and R. Buyya, "A Distributed Deep Reinforcement Learning Technique for Application Placement in Edge and Fog Computing Environments," *IEEE Transactions on Mobile Computing*, pp. 1–1, Oct. 2021.

[7] Y. Chen, S. Deng, H. Zhao, Q. He, Y. Li, and H. Gao, "Data-Intensive Application Deployment at Edge: A Deep Reinforcement Learning Approach," in *Proc. of IEEE ICWS*, Milan, Italy, Jul. 2019, pp. 355–359.

[8] A. Dalgkitsis, P.-V. Mekikis, A. Antonopoulos, G. Kormentzas, and C. Verikoukis, "Dynamic resource aware vnf placement with deep reinforcement learning for 5g networks," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE, 2020, pp. 1–6.

[9] I. Alqerm and J. Pan, "DeepEdge: A New QoE-Based Resource Allocation Framework Using Deep Reinforcement Learning for Future Heterogeneous Edge-IoT Applications," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 3942–3954, Dec. 2021.

[10] B. Krishnamurthy, S. G. Shiva, and S. Das, "MVE-based Reinforcement Learning Framework with Explainability for improving Quality of Experience of Application Placement in Fog Computing," in *Proc. of AIIoT*, Seattle, WA, USA, Jun. 2022, pp. 084–090.

[11] B. Brik, P. A. Frangoudis, and A. Ksentini, "Service-Oriented MEC Applications Placement in a Federated Edge Cloud Architecture," in *Proc. of IEEE ICC*, Dublin, Ireland, Jun. 2020, pp. 1–6.

[12] "Multi-access Edge Computing (MEC). Framework and Reference Architecture," European Telecommunications Standards Institute (ETSI), France, Standard, Mar. 2022.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[14] "Service Requirements for Next Generation New Services and Markets," European Telecommunications Standards Institute (ETSI), France, Standard, Oct. 2018.