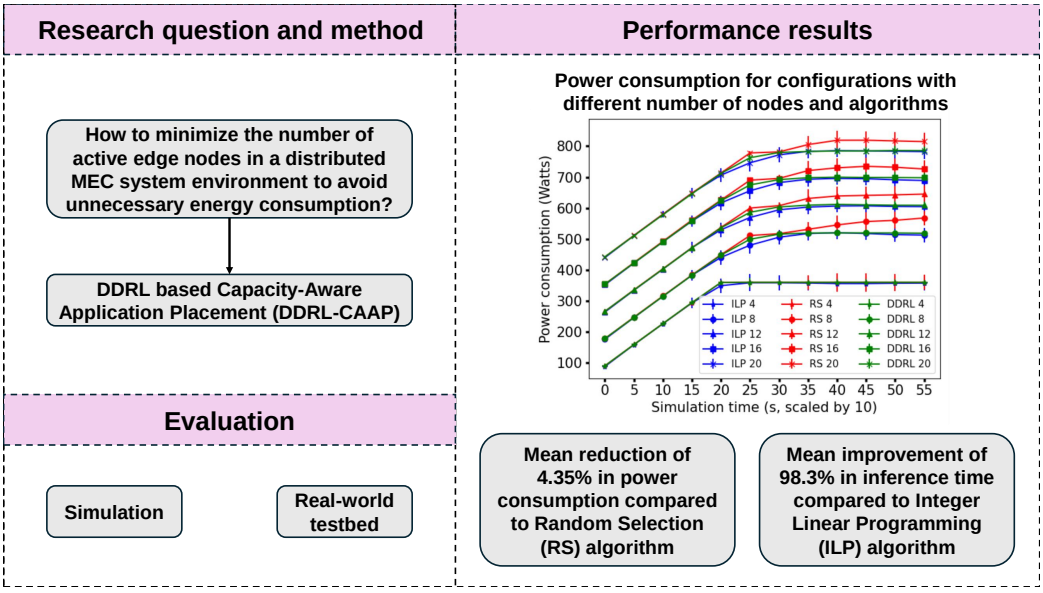
Graphical Abstract

Minimizing Active Nodes in MEC Environments: A Distributed Learning-Driven Framework for Application Placement

Claudia Torres-Pérez, Estefanía Coronado, Cristina Cervelló-Pastor, Javier Palomares, Estela Carmona-Cejudo, Muhammad Shuaib Siddiqui



Highlights

Minimizing Active Nodes in MEC Environments: A Distributed Learning-Driven Framework for Application Placement

Claudia Torres-Pérez, Estefanía Coronado, Cristina Cervelló-Pastor, Javier Palomares, Estela Carmona-Cejudo, Muhammad Shuaib Siddiqui

- Distributed Deep Reinforcement Learning (DDRL) based Capacity-Aware Application Placement (DDRL-CAAP) approach for the placement of applications in multiple MEC system environments.
- Reduction of the number of active edge nodes and compliance with Sevice Level Agreement (SLA).
- Vertical application deployment impact on the DDRL-CAAP approach.
- Testing the model via simulation and on a real testbed to validate its effectiveness under real-world conditions.

Minimizing Active Nodes in MEC Environments: A Distributed Learning-Driven Framework for Application Placement

Claudia Torres-Pérez[a], Estefanía Coronado[a,b], Cristina Cervelló-Pastor[c], Javier Palomares[a], Estela Carmona-Cejudo[a], Muhammad Shuaib Siddiqui[a]

[a]*i2CAT Foundation, Carrer del Gran Capita, 2, Barcelona, 08034, Catalonia, Spain.*
[b]*High-Performance Networks and Architectures, Universidad de Castilla-La Mancha, Campus Universitario de Albacete s/n, Albacete, 02071, Castilla-La Mancha, Spain.*
[c]*Department of Network Engineering, Universitat Politècnica de Catalunya, Campus del Baix Llobregat, Castelldefels, 08860, Catalonia, Spain.*

Abstract

Application placement in Multi-Access Edge Computing (MEC) must adhere to service level agreements (SLAs), minimize energy consumption, and optimize metrics based on specific service requirements. In distributed MEC system environments, the placement problem also requires consideration of various types of applications with different entry distribution rates and requirements, and the incorporation of varying numbers of hosts to enable the development of a scalable system. One possible way to achieve these objectives is to minimize the number of active nodes in order to avoid resource fragmentation and unnecessary energy consumption. This paper presents a Distributed Deep Reinforcement Learning-based Capacity-Aware Application Placement (DDRL-CAAP) approach aimed at reducing the number of active nodes in a multi-MEC system scenario that is managed by several orchestrators. Internet of Things (IoT) and Extended Reality (XR) applications are considered in order to evaluate close-to-real-world environments via simulation

Email addresses: claudia.torres@i2cat.net (Claudia Torres-Pérez), estefania.coronado@i2cat.net (Estefanía Coronado), cristina.cervello@upc.edu (Cristina Cervelló-Pastor), javier.palomares@i2cat.net (Javier Palomares), estela.carmona@i2cat.net (Estela Carmona-Cejudo), shuaib.siddiqui@i2cat.net (Muhammad Shuaib Siddiqui)

and on a real testbed. The proposed design is scalable for different numbers of nodes, MEC systems, and vertical applications. The performance results show that DDRL-CAAP achieves an average improvement of 98.3% in inference time compared with the benchmark Integer Linear Programming (ILP) algorithm, and a mean reduction of 4.35% in power consumption compared with a Random Selection (RS) algorithm.

Keywords: Application placement, Distributed deep reinforcement learning, MEC, Scalability

---

1. Introduction

Multi-Access Edge Computing (MEC) is a critical technology for the advancement of 5G and the development of emerging 6G networks. MEC environments typically involve various stakeholders, such as service providers, system integrators, and application developers, necessitating management mechanisms capable of logical coordination and operation across the system [1]. In this context, it is imperative to acknowledge the presence of not only a single MEC environment as a unified system, but also the potential formation of a distributed network comprising multiple MEC systems. These distinct MEC systems are usually managed by different orchestrators, which adds complexity to the system architecture.

Furthermore, MEC service management techniques must adapt to the different application entry rates, requirements, and Service Level Agreements (SLAs) across distributed scenarios. The literature outlines various objectives that are pursued in order to meet the needs of use cases and fulfill SLAs. One such objective is to minimize power consumption in beyond 5G networks [2] for enhanced sustainability, reduced operational costs, and improved performance. Efficient energy consumption contributes to the overall reliability of networks, as beyond 5G services will consume a significant amount of network resources [3]. Although MEC frequently involves devices with limited resources and low power consumption, it is important to acknowledge that edge computing environments comprise numerous distributed nodes. In this context, an inefficient application placement mechanism may result in resource fragmentation, thus leading to excessive energy consumption.

A common practice in service management strategies is balancing the workloads between the network nodes [4, 5, 6]. However, each active node has an overhead, which could result in excessive power consumption. Concentrat-

ing the load onto a minimal number of nodes makes it possible to maintain a certain number of them in an idle state. Due to idle power consumption, inactive nodes typically consume less power than those that are only lightly loaded. The curve in [7] suggests that power consumption remains a linear function up to approximately 70% of CPU consumption. Beyond this point, the curve adopts a constant trend from 70% to 100%. If the load is distributed across all nodes, most nodes will operate at a lower CPU usage, potentially in the linear low-power range. While this balances the thermal and computational stress, it increases the overall power consumption because it maintains the base power draw of multiple nodes. Operating the minimum number of nodes at a higher load allows a more efficient use of energy.

The approach of minimizing the number of active nodes is a powerful solution in MEC environments, wireless sensor networks [8, 9] and containerized platforms such as Docker and Kubernetes. Keeping more active nodes than those that are necessary leads to higher energy consumption, which is particularly problematic in environments such as wireless sensor networks, where nodes are often battery-powered. Minimizing the number of active nodes enhances resource efficiency by optimizing computational resources, thus reducing operational costs. Besides reducing power consumption, which is important in pay-as-you-go models, such as distributed cloud environments [10], this approach can decrease system complexity, mitigate potential points of failure, and ensure free space for placing services when needed. Thus, research work on the minimization of the number of active nodes can lead to sustainable and cost-effective systems in distributed computing environments.

Although considerable research has been conducted into reducing energy consumption in edge computing [11, 12, 13] by minimizing the number of active nodes or balancing the load between them, there are still several gaps in the research that need to be addressed. For instance, the work in [11] describes an application placement approach considering the limited energy load of edge servers. However, this study did not consider the potential energy savings and improved adaptability to dynamic user mobility that could result from minimizing the number of active nodes, which would allow the system to more efficiently reallocate resources and maintain optimal performance and energy efficiency. Similarly, the authors of [12] introduced algorithms to minimize service usage and distribute the load among the servers, effectively reducing energy drainage, and the number of active edge servers. However, their approach excluded the study of infrastructures with multiple MEC systems. The work in [13] modeled the placement of deep neural network

(DNN) services on a minimal number of edge nodes. Nevertheless, it did not consider the dynamic ingress/egress of applications in the system. Thus, there is a need to develop a comprehensive strategy that not only aims to minimize energy consumption on edge nodes, but also ensures node availability for critical tasks and applications, leading to savings in the operational costs of large edge infrastructures.

Moreover, the assessment of application placement solutions in the literature predominantly relies on simulation-based studies rather than real testbed environments. The analysis needs to take into account how the real hardware influences application placement algorithms. Additionally, an important approach to consider is the scalability of a proposed solution. An algorithm must be efficiently scalable and seamlessly manage a variable number of nodes, applications and MEC systems, without impacting performance. This capacity of adaptation to different scenarios improves reliability and maintains Quality of Service (QoS).

Building on our previous research [14], this paper focuses on minimizing the number of active nodes in the edge infrastructure while considering several vertical application requirements. We propose a strategic approach for application placement within MEC systems, that is aimed at enhancing overall system performance in response to dynamic demands. Additionally, we examine the approach's scalability for different numbers of nodes, for diverse MEC systems, and for applications of varying durations within the system, with a particular emphasis on MEC systems featuring Internet of Things (IoT) and Extended Reality (XR) applications. This strategy intends to address the challenges of operating in dynamic network environments that closely mirror real-world scenarios. This paper also evaluates the potential power consumption savings that can be achieved by applying our strategy for active node minimization, as well as the approach's scalability in terms of the maximum number of MEC systems and nodes it can support, the application requirements, and the entry rate. Furthermore, we study the extent to which a real infrastructure impacts model performance and inference time. Thus, the contributions of this work are as follows:

- This paper presents a Distributed Deep Reinforcement Learning-based Capacity-Aware Application Placement (DDRL-CAAP) approach for the placement of applications in highly distributed environments, considering an architecture that is fully aligned with the European Telecommunications Standards Institute (ETSI MEC) specifications [15]. This

approach aims to reduce the number of active edge nodes and ensure SLA compliance.

- We evaluate the impact of deploying vertical applications with variable duration and service requirements on the performance of the DDRL approach.
- The model is tested via simulation and on a real testbed to validate its effectiveness under real-world conditions compared with an Integer Linear Programming (ILP) approach.

The remainder of this paper is organized as follows. Section 2 reviews related works on the topic of application placement. The system model is presented in Section 3, and Section 4 describes the problem formulation. The proposed DDRL-CAAP model is introduced in Section 5, and Section 6 explains how the algorithm is incorporated into a real MEC system. Section 7 describes the configuration of the test scenarios, and the performance evaluation is discussed in Section 8. Finally, Section 9 concludes the paper.

2. Related Work

Research into application and workload placement employs diverse methodologies such as numerical analysis, mathematical optimization, and Machine Learning (ML). Our literature review systematically categorizes works related to application placement in edge and fog computing, focusing on energy minimization approaches. Several surveys have detailed the different techniques in scheduling and application management strategies in edge computing [16] and serverless environments [17, 18]. A review study is presented in [16] that looks at proactive caching strategies divided into heuristic-based, model-based, and ML-based strategies. The authors of [17] classified the scheduling approaches according to the objectives pursued by the works surveyed. A study in serverless computing is performed in [18], in which the authors classify the approaches as ML-based, framework-based and model-based. The above studies help to understand current optimization objectives in scheduling and application placement problems, and the techniques applied to solve them.

The authors of [19, 20, 21, 22, 23, 24, 25] proposed various techniques for service placement, aiming to optimize different objectives, such as quality of experience, QoS, latency, throughput and service cost. The work in [19] employed fuzzy logic to handle application placement on the basis of user

experiences. The approach involved dividing each application's workload into tasks, with the application being hosted on computational and gateway nodes within fog nodes. Badidi et al. [20] presented a QoS-aware placement method for IoT applications, utilizing a task scheduler to orchestrate all tasks across a fog node cluster. A particle swarm optimization method was proposed in [22] to perform IoT services placement. The research in [21] introduced a further optimization technique for service placement, which was evaluated with lightweight and heavyweight applications composed of microservices. Yi et al. [23] proposed an approach for social-aware device to device content sharing with proactive caching employing a basis transformation with low complexity. The results demonstrated acceptable performance and explain the complexity of the method. The work in [24] introduced a multi-dimensional resource optimization problem to maximize the network power control. The authors solved the mixed integer linear programming problem with a branch-and-price solution and a suboptimal greedy algorithm to facilitate the process, effectively maximizing the network management profit. The greedy approach achieves less computational complexity compared with the optimal solution, considering the average running time. The authors of [25] presented a two-timescale accuracy-aware optimization approach for human digital twin deployment at the network edge for assisting human-centric task execution. The analyzed approaches obtained acceptable performance in terms of completion time. Thus, applying reinforcement learning strategies to adapt to real-time changes in network conditions and user behavior could lead to an improvement in completion time, enhancing the system's responsiveness and efficiency.

In this regard, Deep Reinforcement Learning (DRL) has been widely used in application placement [26, 27] due to its ability to model unforeseen scenarios, handle learning directly from data and interactions with the environment, and adapt to variations in the problem without the need to redesign the approach. Sami et al. [26] sought an auto-scaling and placement solution with DRL for multi-application integration into a service-based clustering environment. The resource provisioning approach was based on a MEC architecture comprising a single orchestrator and a set of MEC servers. The work in [27] considered a multi-orchestrator system for assigning the applications to a specific MEC host while minimizing latency. While excellent work has been presented in the application placement field, it is paramount to increase the ratio of accepted applications, minimize power consumption, and consider multiple MEC systems.

A promising area of research in the context of service management is the reduction in energy consumption [28, 29, 30, 11, 31, 32]. The application of distributed learning to the problem of energy-aware application placement has been proposed to minimize inference time during exploitation and improve performance when several workers intervene in the training process. For instance, the authors of [28] sought to minimize execution time and energy consumption for IoT applications. A DDRL approach is applied in [29], with cooperative exploring and prioritized experience replay to increase energy task efficiency and reduce the average processing time over an unmanned aerial vehicle MEC network, considering its mobility and possible failures. Also addressing energy consumption minimization, the authors of [30] presented an energy-efficient scheduling algorithm to process user applications with real-time requirements, implementing a trade-off between energy consumption and task execution time. Similarly, Badri et al. [11] considered an energy-aware application placement to maximize QoS, given the constrained energy budget of the servers. For this purpose they utilized a greedy heuristic method. An energy-efficient computation offloading approach was presented in [31], in which the authors seek a trade-off between delay and energy consumption. The authors of [32] employed a federated learning technique to optimize energy consumption, execution cost, network usage, delay, and fairness in a multi-user offloading approach for mobile edge computing. Although the above works addressed the trade-offs between energy consumption, task execution time, and task efficiency, they did not consider the advantages of minimizing the number of active nodes.

The problem of minimizing the number of active nodes is not exclusive to MEC systems, as it is also a challenge in other distributed architectures, such as wireless sensor networks [8, 9], and containerized platforms such as Docker and Kubernetes. The work in [33] provided an architecture approach for scheduling service workloads to minimize latency to end-users. The authors of [34] presented a network-aware framework to perform the placement of dependent microservices in long-running applications while also minimizing latency and bandwidth reservations. They compared the approach in simulation and testbed environments, and, in this case, the pods were allocated next to each other to minimize the expected network latency. Both works [33, 34] were focused on optimizing the service placement problem, but they overlooked the reduction of power consumption through minimizing the number of active nodes and the possible minimization of costs this could bring. Despite extensive research on performance optimization, load

balancing, and latency reduction, the energy efficiency of distributed systems remains unexplored. Specifically, there is a critical need to investigate how minimizing the number of active nodes affects energy consumption. This includes dynamically adjusting active nodes in response to real-time workload demands while ensuring service continuity and maintaining node availability for unexpected high-demand scenarios.

Performing load balancing to achieve optimization objectives is the subject of several research works. These objectives could include reliability maximization, delay reduction and energy consumption minimization. For instance, the work in [4] addressed the problem of microservice deployment with a view to maximizing the system-wide reliability and satisfaction of task delay requirements. Similarly, Ying et al. [5] proposed an Age of Task-oriented Information (AoTI) approach for industrial tasks, minimizing long-term AoTI for wireless sensor network applications, and optimizing access selection and sampling frequencies for all sensors. Both works introduced a practical approach to meet the said objectives. Nevertheless, none of them examine how the proposed approaches impact the power consumption of the nodes. The authors of [6] aimed to minimize the long-term service delay of mobile devices and ensure system stability by considering constrained energy consumption and caching capacities. They proposed an algorithm to select between task rerouting and service migration, based on the Lyapunov optimization method. Their approach effectively reduced energy consumption through a system that balances the computational workload of mobile devices and edge servers. Applying the approach of minimizing the number of active nodes in these works could provide valuable insights into its impact on energy consumption.

Conversely, demonstrating the scalability of an algorithm in a distributed environment is a crucial factor to consider. The work in [28] achieved a satisfactory performance in two scenarios involving variability in application datasets. However, the authors did not present an explanation for variability in the number of edge servers. The authors of [30] identified the influence of varying time rates, but no discussion was provided on how a variation in the number of servers would affect the algorithm's outcome. An analysis of algorithm scalability in relation to the number of channels and end users is presented in [24].

Further study is required to extend the research that has already been conducted. This should include multi-domain orchestration, with the potential reduction in the number of active nodes considered as a technique to minimize power consumption. Furthermore, it is necessary to evaluate the influence

Ref.	Evaluation tools	Techniques Used	Performance metrics	Datasets
[6]	Simulations	Lyapunov method	average service delay, average energy	simulated data of range of variables and distributions obtained by references
[11]	GCC, OpenMP	greedy heuristic SAA	relative objective ratio, request satisfaction ratio, execution time	smartphones dataset for cybersecurity research
[26]	Python	DQN	average cost of algorithm, availability, average cpu usage, resource load of services	Google cluster Usage Trace dataset
[28]	Python, testbed	X-DDRL	execution time, energy consumption, weighted cost, placement time overhead	Values obtained based on testbed experiments
[29]	Python	DDRL-PER	convergence rate, energy-task efficiency, average processing task	simulated data of range of variables and distributions obtained by references
[30]	Simulations	heuristic solutions, GA	energy consumption, success rate, computation time	simulated data of range of variables and distributions obtained by references
[31]	iFogSim	HMM	energy consumption, execution cost, delay, network resource usage, time interval	Author-generated data
[22]	MATLAB R2019a	PSO	services performed, waiting time, failed services, services cost, services remaining, runtime	simulated data of range of variables and distributions obtained by references
[32]	iFogSim	FL	energy consumption, execution cost, network usage, delay, fairness	data from VRGAMEFOG application
Our work	Python, testbed	DDRL-CAAP	power consumption, resource usage, inference time	simulated data of range of variables and distributions obtained by references

Table 1: Comparison of energy-aware methods in related works.

Note: GCC: GNU Compiler Collection, SAA: Sample Average Approximation, GA: Genetic Algorithm, FL: Federated Learning, PSO: Particle Swarm Optimization, HMM: Hidden Markov model.

of different applications, edge configuration scenarios in simulation and real-world testbeds, and the application acceptance rate in varying multi-edge configurations. The findings of the works presented on minimizing energy consumption are summarized in Table 1, which highlights the differences between those works that pursued the same objective.

3. System Model

The system model presented in this paper is built upon an extended version of the ETSI MEC reference architecture [15], which is specifically tailored to accommodate distributed MEC systems through the ETSI MEC federation interface [35]. The interface is used for information sharing between federated MEC systems, enabling federation capabilities and operations to support

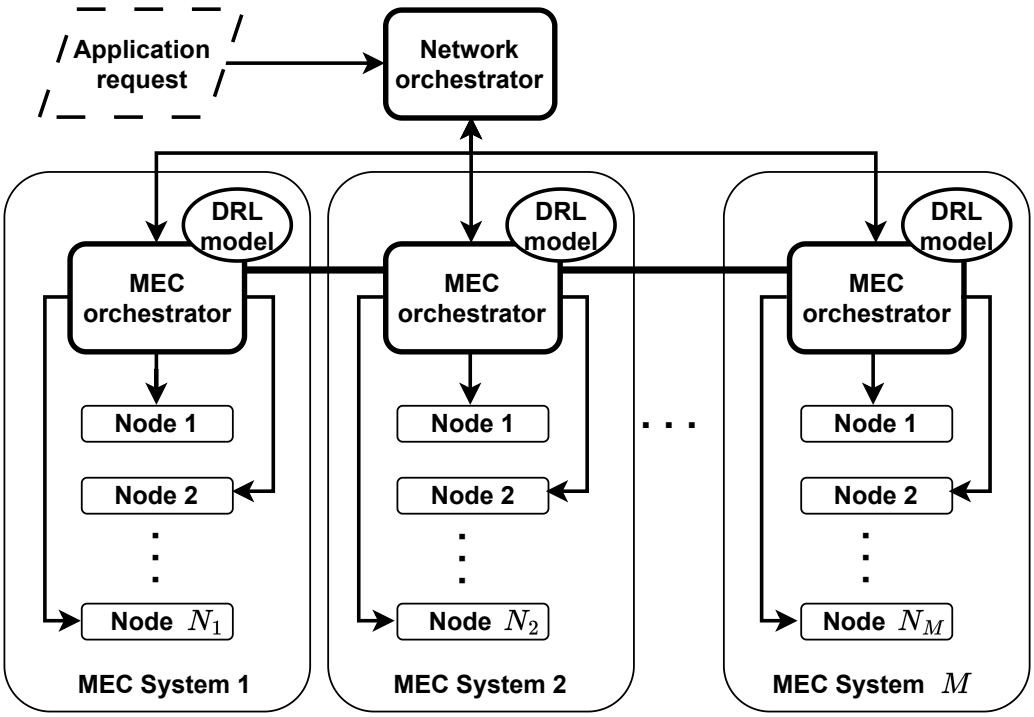


Figure 1: System model: distributed MEC system architecture.

complex federated environments. As shown in Figure 1, the architecture includes a network orchestrator at the highest level that oversees M MEC systems managed by one MEC orchestrator (MEO) each. Without loss of generality, we assume that application requests are forwarded from the network orchestrator to the MEO of a given MEC system, and applications are placed directly on the MEC nodes.

The application placement process begins with the network orchestrator receiving an application placement request and forwarding the application to a target MEC system. The placement request contains the application descriptor, including the application's requirements for storage, CPU, and RAM. Within each MEC system m, each MEO executes a DRL model to select the optimal MEC node to deploy the application. The DRL model considers the application's requirements and the MEC nodes' availability to make informed decisions about the MEC node for application placement, thereby minimizing the number of active nodes. Section 5 presents the DRL-based mechanism for MEC node selection.

Let $\mathcal{M} = \{1, \cdots, M\}$ represent the set of MEC systems in our network architecture and $\mathcal{N}_m = \{1, \cdots, N_m\}$ the set of nodes in the m-th MEC system. For any given MEC system m, the computational resources of each node n_m in terms of storage, RAM, and CPU are respectively given by elements in the tuple (1),

$$\mathcal{C}_{n_m} = \{\mathcal{C}_{n_m}^{ST},\ \mathcal{C}_{n_m}^{CPU},\ \mathcal{C}_{n_m}^{RAM}\}. \tag{1}$$

Let $\mathcal{R} = \{1, \cdots, R\}$ denote the set of application requests arriving at the network orchestrator following a Poisson distribution. Each request has a duration that follows a uniform distribution within the range $[T_{\min},\ T_{\max}]$. Moreover, each request $r \in \mathcal{R}$ is associated with a given type of application $k \in \mathcal{K} = \{1, \cdots, K\}$ that requires specific resource capacities in terms of storage, RAM, and CPU, represented in (2),

$$D_r(k) = \{D_r^{ST}(k),\ D_r^{CPU}(k),\ D_r^{RAM}(k)\}. \tag{2}$$

In order to analyze the scalability of our problem, experiments in relation to the variability in the number of nodes N_m, number of MEC systems M, and types of applications K are conducted in Section 8.

4. Problem Formulation

This section provides the mathematical formulation for the application placement problem, considering multiple MEC systems with several nodes. For the sake of simplicity, we assume that each MEC system comprises an equal number of nodes. We define an ILP problem to determine the placement of each application request by selecting the MEC system and the node with enough resources to meet the requirements. The objective is to keep the maximum number of nodes on standby, thereby leaving nodes free of applications whenever feasible, and minimizing resource fragmentation. This ILP problem is called a One-Step Capacity-Aware Application Placement (One-Step-CAAP) problem.

Thus, we define an iterative procedure that uses the One-Step-CAAP model, at each step locating each incoming application request received following a Poisson process. The iterative procedure shown in Figure 2 includes recovering resources from a request that ends at a given instant. It also provides the migration of previously located requests if the completion of a request has fragmented the available resources between several nodes in such a way that they could be optimized. The overall goal is to minimize

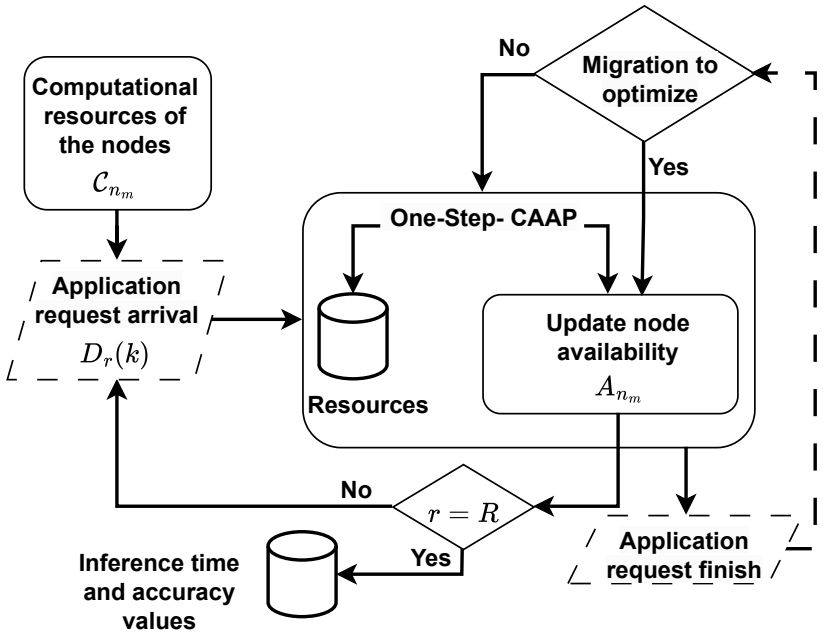


Figure 2: Iterative One-Step-CAAP procedure.

the number of active nodes subject to capacity constraints. This procedure optimizes the model's accuracy in selecting the most appropriate node, and the following formulation models the behavior of the One-Step-CAAP problem. Throughout the iterative execution, the nodes' resources decrease according to the occupation of the previous arrivals. Each step determines whether the current request r can be located given the current available resources by minimizing the number of active nodes (only activating new ones if there is insufficient capacity in the active nodes). To formulate the One-Step-CAAP problem, we define a binary variable $x_m^{n_m}$ (3) as follows:

$$x_m^{n_m} = \begin{cases} 1 \text{ , if current request is successfully deployed} \\ \quad \text{at node } n_m \in \mathcal{N}_m \text{ of MEC } m \in \mathcal{M} \\ 0 \text{ , otherwise.} \end{cases} \tag{3}$$

In addition, a binary variable z represents whether the new arrival is placed in the system. Let's assume that $r-1$ requests have been successfully placed. Then, the available resources in storage, CPU, and RAM resources upon

request r are given by (4),

$$\mathcal{A}_{n_m} = \{\mathcal{A}^{ST}_{n_m},\ \mathcal{A}^{CPU}_{n_m},\ \mathcal{A}^{RAM}_{n_m}\} = C_{n_m} - \sum_{\rho=1}^{\rho=r-1} x_m^{n_m} \cdot D_\rho(k), \quad \forall \rho \in \mathcal{R}, \quad (4)$$

being ρ the set of ordered requests between 1 and R, then the One-Step-CAAP model is:

$$\min \quad \frac{1}{M \cdot N_m} \sum_{m=1}^{M} \sum_{n_m=1}^{N_m} x_m^{n_m} - z \quad (5)$$

$$\text{s.t.} \quad x_m^{n_m} \cdot D_r^{ST}(k) \leq \mathcal{A}^{ST}_{n_m} \qquad \forall n_m \in \mathcal{N}_m,\ \forall m \in \mathcal{M} \quad (6)$$

$$x_m^{n_m} \cdot D_r^{RAM}(k) \leq \mathcal{A}^{RAM}_{n_m} \qquad \forall n_m \in \mathcal{N}_m,\ \forall m \in \mathcal{M} \quad (7)$$

$$x_m^{n_m} \cdot D_r^{CPU}(k) \leq \mathcal{A}^{CPU}_{n_m} \qquad \forall n_m \in \mathcal{N}_m,\ \forall m \in \mathcal{M} \quad (8)$$

$$\text{if } x_m^{n_m} = 1 \Rightarrow z = 1 \qquad \forall n_m \in \mathcal{N}_m,\ \forall m \in \mathcal{M} \quad (9)$$

$$\text{if } \sum_{m=1}^{M} \sum_{n_m=1}^{N_m} x_m^{n_m} = 0 \Rightarrow z = 0 \quad (10)$$

$$x_m^{n_m},\ z \in \{0, 1\} \qquad \forall n_m \in \mathcal{N}_m,\ \forall m \in \mathcal{M} \quad (11)$$

The objective function (5) consists in minimizing the number of active nodes while maximizing the probability of placing the current request. Constraints (6), (7), and (8) ensure that the sum of the computational resources demanded by the current request and all the previous requests allocated on any node $n_m \in \mathcal{N}$ of any MEC $m \in \mathcal{M}$ does not exceed the available capacity in terms of storage, CPU or RAM. The non-linear constraints (9) and (10) are implications that define the dependency between the variables $x_m^{n_m}$ and z, establishing that z is equal to 1 if the current request is placed at whatever node of any MEC; and it is equal to 0 if insufficient availability exists to locate the request at any node. By using linearization techniques, both non-linear constraints can be linearized. Constraint (9) is equivalent to:

$$z \geq x_m^{n_m} \qquad \forall n_m \in \mathcal{N}_m,\ \forall m \in \mathcal{M},$$

and constraint (10) is linearized with the following expression:

$$z \leq \sum_{m=1}^{M} \sum_{n_m=1}^{N_m} x_m^{n_m}.$$

Finally, all the variables are defined as binary in (11).

The ILP requires high computational time to obtain a solution, making its use in real-time unfeasible. For this reason, this work chooses a DDRL-based method, which makes it possible to find a time-efficient solution with low computational complexity that is close to the optimal solution once the trained model is obtained. Additionally, in scenarios where different MEC systems host various applications, ILP models may need help with scalability and responsiveness due to their deterministic nature and computational intensity. A DDRL approach allows distributed DRL models to collaborate and learn efficient application placement strategies tailored to each MEC system's application requirements and node availabilities. This enables the overall system to optimize active node counts dynamically, enhancing efficiency while adapting to changing conditions and demands in real-time. Additionally, when we need to update the type of applications or their range of requirements, a distributed DRL model can adapt to such changes. Section 5 presents the modeling of the problem following the DDRL-CAAP approach.

5. Modeling Approach for Application Placement

This section builds upon the preceding discussion by describing the DRL model and the DDRL-CAAP technique employed for distributed training and inference. It also outlines a real-world use case to which the DDRL-CAAP approach could be applied and the time complexity of the proposed algorithm.

In the context of DDRL training, one of the actors (the DRL model) is placed in conjunction with each MEC orchestrator. The nodes' availability and application requirements, which are managed locally by each orchestrator, are the inputs of the DRL model's learning processes. Therefore, this method can handle the learning processes performed by various DRL models, potentially using different application types and requirements across various management domains.

5.1. Deep Reinforcement Learning Model

Deep Q Network (DQN) is a variant of DRL [36] that employs experience replay and deep neural networks to stabilize the training of the action value function. DQN is comprised of a main and a target network, where the main DNN estimates the current state and the action Q-values, and the target DNN is utilized to approximate the Q-values of the subsequent state and action. It remains static until a sufficient number of iterations have been

completed. The parameters from the main network are copied to the target network, transferring the learning from one to the other and making the estimates computed by the target network more accurate.

This work describes the application placement problem as a Markov Decision Process (MDP). The DRL method selects the optimal policy for the MDP, employing the local data for each MEC system. The MDP's state, action space, and reward function are crucial to performing the model training.

The state space S_m of the DRL model corresponding to MEC system m, defined in (12), contains the availability of storage, CPU, and RAM resources on the nodes in the MEC system m, as well as the incoming application requirements for each of these resources. Each DRL model receives and processes the information about the availability of the nodes and the application request corresponding to each specific MEC system.

$$S_m = \{\mathcal{A}_{n_m}^{ST},\ \mathcal{A}_{n_m}^{CPU},\ \mathcal{A}_{n_m}^{RAM},\ D_r(k)\}, \tag{12}$$

where $\mathcal{A}_{n_m}^{ST}$, $\mathcal{A}_{n_m}^{CPU}$, and $\mathcal{A}_{n_m}^{RAM}$ define the remaining storage, CPU, and RAM of all nodes thus:

$$\begin{aligned}
\mathcal{A}_{n_m}^{ST} &= \{\mathcal{A}_1^{ST},\ \mathcal{A}_2^{ST}, \ldots,\ \mathcal{A}_{\mathcal{N}_m}^{ST}\},\\
\mathcal{A}_{n_m}^{CPU} &= \{\mathcal{A}_1^{CPU},\ \mathcal{A}_2^{CPU}, \ldots,\ \mathcal{A}_{\mathcal{N}_m}^{CPU}\},\\
\mathcal{A}_{n_m}^{RAM} &= \{\mathcal{A}_1^{RAM},\ \mathcal{A}_2^{RAM}, \ldots,\ \mathcal{A}_{\mathcal{N}_m}^{RAM}\}.
\end{aligned}$$

The action space A is defined as the group of all the nodes in the m-th MEC system, i.e., $A = \mathcal{N}_m$. An action a is associated with the specific node n_m where the request is placed. The reward function is given by Eq. (13). We define the reward w^{n_m} for a request r, demanding $D_r^i(k)$ resources $\forall i \in b$, and for the action $a = n_m$, as:

$$w^{n_m} = \begin{cases} 0 & \text{if } \mathcal{A}_{n_m}^i = \mathcal{C}_{n_m}^i, \forall i \in b \\ 10 \cdot (2^{2l} - 1) & otherwise, \end{cases} \tag{13}$$

where

$$l = \begin{cases} \sum_{\forall i \in b} \left(1 - \frac{\mathcal{A}_{n_m}^i + D_r^i(k)}{\mathcal{C}_{n_m}^i}\right) & \text{if all } \mathcal{A}_{n_m}^i \geq D_r^i(k), \\ \sum_{\forall i \in b} \left(\frac{\mathcal{A}_{n_m}^i - D_r^i(k)}{\mathcal{C}_{n_m}^i} - 1\right) & \text{if any } D_r^i(k) > \mathcal{A}_{n_m}^i \end{cases} \tag{14}$$

and $b \in \{ST, CPU, RAM\}$.

Eq. (14) presents a variable l, which acts as a scaling factor based on the usage levels and demand. For the first case (if all $\mathcal{A}^i_{n_m} \geq D^i_r(k)$), if the constraints (6), (7), and (8) are satisfied, the reward is positive and increases in value as the availability of the node decreases. The fraction $\frac{\mathcal{A}^i_{n_m} + D^i_r(k)}{\mathcal{C}^i_{n_m}}$ represents the proportion of the node's capacity that is available if the application demand is placed on the selected node. By subtracting this fraction from 1, we obtain a measure of the capacity occupied, which indicates the node's proximity to its maximum capacity after placement. Lower values of this expression mean more unused capacity after placing the demand, while higher values indicate a usage closer to full capacity. The total l is computed by summing this expression across all resources. The sum allows the placement score to consider the combined availability across different resource types, which is important since applications may demand different combinations of resources.

The second case (if any $D^i_r(k) > \mathcal{A}^i_{n_m}$, and the constraints (6), (7), or (8) are not satisfied) applies if the request exceeds the availability of the node, and the function becomes negative, acting as a penalty and discouraging the use of nodes that cannot satisfy the demand of the application. Therefore, $\mathcal{A}^i_{n_m} - D^i_r(k)$ is the deficit in resources if the demand is larger than the available resources. By dividing by the computational capacity, we obtain a normalized measure of this deficit relative to the node's capacity, and by subtracting 1, the negative value obtained serves as a penalty. Total l is calculated as in the first case.

An exponential reward function significantly impacts the learning dynamics in DRL agents, leading to faster convergence and more effective policies, as an exponential function produces larger gradients for high-reward actions, helping the agent distinguish between actions more effectively. In this case, an exponential function boots the placement of applications on nodes with less availability to ensure this case maintains the number of active nodes to a minimum. With an exponential function, the agent becomes more sensitive to high-reward outcomes, encouraging it to explore placements that maximize system efficiency without overly favoring suboptimal configurations. Therefore, in the following points, we explain how the reward function (Eq. (13)) minimizes the number of active nodes.

- Zero reward for idle nodes (if $\mathcal{A}^i_{n_m} = \mathcal{C}^i_{n_m}, \forall i \in b$): By assigning a

MEC node	Condition	l	w^{n_m}
1	$\mathcal{A}^i_{n_m} = \mathcal{C}^i_{n_m}$	-	0
2	$\mathcal{A}^i_{n_m} \geq D^i_r(k)$	1.25	46.56
3	$\mathcal{A}^i_{n_m} \geq D^i_r(k)$	1.5	70
4	$D^i_r(k) > \mathcal{A}^i_{n_m}$	-2.6875	-9.75

Table 2: Example of reward function and l values for a specific case.

reward of zero to completely idle nodes, the function discourages the activation of nodes that are not required. This encourages the system to activate the idle nodes only when necessary.

- Penalty for overloaded nodes with the incoming request (if any $D^i_r(k) > \mathcal{A}^i_{n_m}$): The penalty term in l discourages placing applications on nodes that would exceed their available capacity. The penalty prevents the selection of nodes that cannot handle the incoming request.

- Higher reward for efficient utilization (if all $\mathcal{A}^i_{n_m} \geq D^i_r(k)$): When the node capacity can meet the request, the model receives a positive reward scaled by how closely the availability matches the demand. Therefore, the decision encourages placing applications on nodes where resource usage is efficient, maximizing utilization of active nodes and minimizing the need to activate additional ones.

Let us outline an example to illustrate how the reward function addresses the goal of minimizing the number of active nodes. If we consider a MEC system composed of four nodes, with the same initial capacities $\mathcal{C}_1 = \{128$ GB, 16 vCPU, 32 GB$\}$, then $\mathcal{C}_1 = \mathcal{C}_2 = \mathcal{C}_3 = \mathcal{C}_4$. After a certain number of application requests have been placed in the system, the current availability of the nodes is the following: $\mathcal{A}_1 = \{128$ GB, 16 vCPU, 32 GB$\}$, $\mathcal{A}_2 = \{64$ GB, 10 vCPU, 6 GB$\}$, $\mathcal{A}_3 = \{64$ GB, 6 vCPU, 6 GB$\}$, $\mathcal{A}_4 = \{64$ GB, 2 vCPU, 4 GB$\}$. Then the request r arrives with the requirements given by $D_r(1) = \{8$ GB, 4 vCPU, 4 GB$\}$. Table 2 shows the values of the variable l and the reward function w^{n_m} for each node. In this case, the reward function promotes placement on $n_m = 3$, selecting the node that is closest to the limits of its capacity.

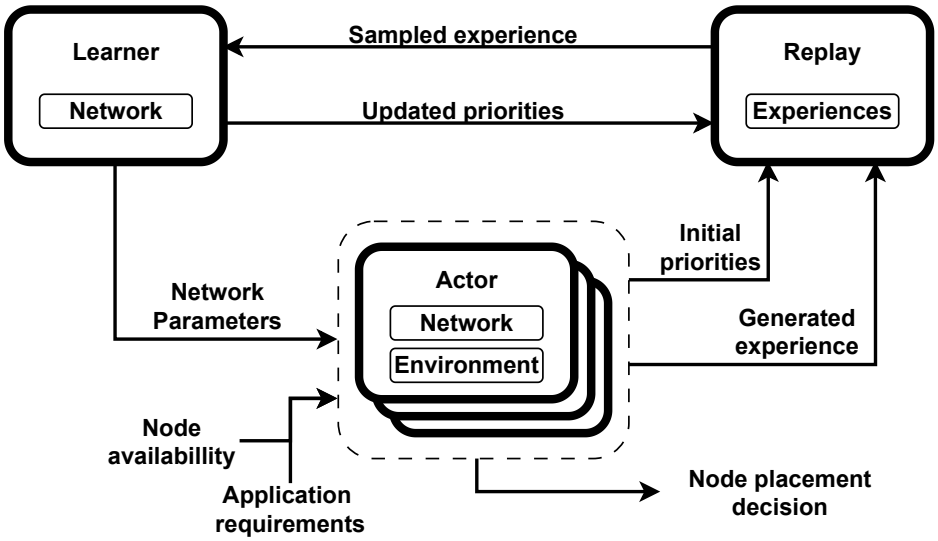


Figure 3: DDRL selected scheme: ApeX architecture [37].

5.2. Distributed Deep Reinforcement Learning Model

This subsection describes the architecture utilized to build the DDRL-CAAP approach. The training distribution allows parallelization, making it particularly suitable for scenarios in which data management and ownership may be independent across various domains. This is the case for the architecture proposed in this work, as described in Section 3. DRL, which relies on trial and error, involves processing large volumes of data, which can be computationally demanding. However, distributed algorithms for DRL allow agents to learn efficiently and manage multiple tasks concurrently. The DDRL algorithm distributes DRL models across multiple machines, with each model replica using the data obtained from its specific environment. Each model replica is a DRL model, and each of them receives the data arriving from each MEC system environment. The data contains the application request arriving at this specific MEC system, which was previously selected by the orchestrator, and the availability of the nodes of each specific MEC system. The approach chosen for distributed training in this work is ApeX [37].

Figure 3 shows the ApeX architecture. The main components are the learner, the prioritized replay buffer, and the multiple actors performing a dueling DQN algorithm. Each actor runs in distinct MEC system environments, enabling the collection of a diverse dataset. ApeX includes the concept of distributed prioritized experience replay, which is based on the intuition that

some experiences can contribute more to an agent's learning than others. The objectives are accelerating convergence, reducing variance, and benefiting from importance sampling and prioritized experience replay [38].

The actors perform a remote call to the learner to obtain the latest network parameters, and each one starts interacting with its environment, selecting and applying an action. In each actor, initial priorities are computed and sent to the prioritized replay buffer, where they are combined with the experience generated from transitions in the environment. The learner samples a prioritized batch of transitions from the replay buffer, which the actors previously sent there. It then applies the dueling DQN rule, calculates priorities from experience, and updates it in the replay buffer. In addition, the learner sends network parameters to the actors using the parameters gathered from computed experiences. A sequence diagram illustrating this explanation is depicted in Figure 4, which shows the relationship between the system model components and the distributed algorithm at the training phase.

The implementation of distributed training, with agents operating at each MEC system level, is designed to facilitate the acquisition of experiences based on the diverse environments inherent to each of them. These environments may exhibit discrepancies in the type of application or the type of nodes. The experiences acquired by each agent are disseminated together with the weights during the training phase. Consequently, in the model inference phase, if a MEC system encounters features not observed during the training process, retraining is unnecessary, and the model will respond appropriately.

The observations of the DRL agents are distributed to facilitate the learning process. Different techniques have been proposed in the literature to model the interactions between distinct components in an algorithm. For instance, the work in [39] presents a novel hierarchical game to model interactions between legitimate users, eavesdroppers, and existing jammers. The authors implemented a hierarchical game between three parties to ensure strategic interactions, first considering the maximization of the utility of each party and then constructing a hierarchical game to model the decision sequence and correlation of these problems. In this case, it is necessary to organize the hierarchy because these parties could act strategically and selfishly. Nevertheless, in our algorithm, no selfish behavior occurs among the actors; they send the observation obtained for each of them: reward, state information, and action obtained to the replay buffer. The experiences are sent in equal conditions to the replay buffer and the learner, and the decision of which experiences to share can be made by prioritizing sequences of past

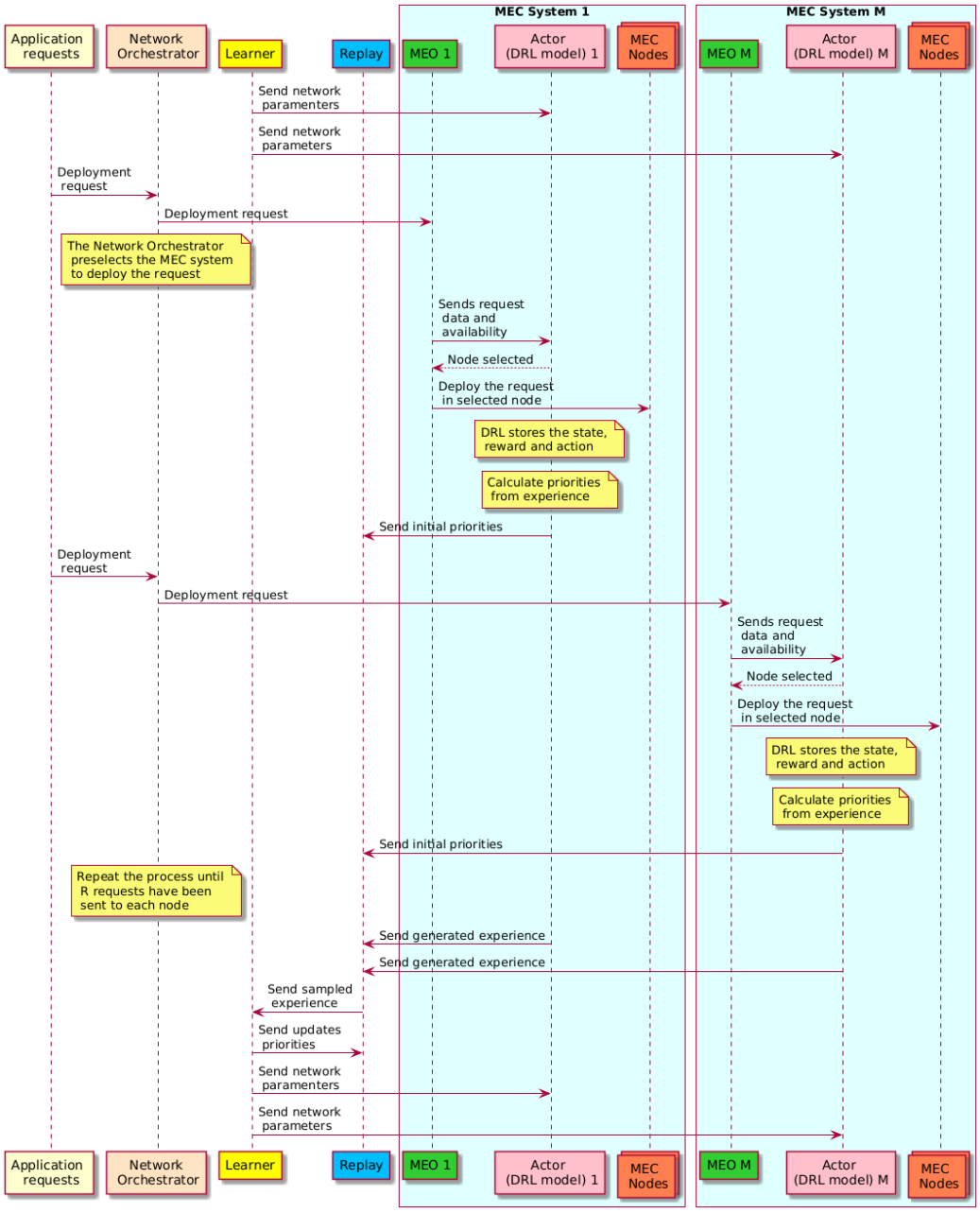


Figure 4: DDRL-CAAP Sequence diagram.

experiences and not individual transitions. The experiences that contribute most to the model are selected, and the learner does not discriminate between agents.

5.3. Real-world Case Study

To showcase the applicability of this work, let's consider the example of a multiple MEC system environment in a city. The different MEC systems could be geographically distributed depending on citizens' demands. The application requests are unpredictable, with specific areas having fluctuating demands; IoT Fog demands could have high traffic in heavy traffic zones, while XR applications could have more demands in certain events. Even if the model has been trained on common application scenarios (e.g., XR in public parks or IoT Smart applications in residential zones), it can generalize placement strategies for unexpected scenarios (e.g., XR requests at intersections).

In high-demand events, an activation strategy needs to be coordinated. If an event requiring high demand for XR applications arises, the applications must be placed on nodes with high utilization, avoiding additional nodes whenever possible; if the existing nodes activated are overloaded, the strategy must go to those inactive nodes to maintain QoS. Therefore, these nodes are available to host applications in high demand. Fewer active nodes translate to lower energy consumption in a large MEC deployment across a city. Ensuring nodes are used optimally without leaving nodes underutilized could avoid resource fragmentation. Furthermore, minimizing active nodes makes the system scalable without excessive energy or resource wastage, as only essential nodes remain active at any given time.

5.4. Time Complexity of DDRL-CAAP

The time complexity of DDRL-CAAP is analyzed using the Big $\mathcal{O}$ notation, and considers the different parts that intervene in the learning process: experience collection by the actors, model training by the learner, data transfer from actors to the learner and model parameter broadcast from the learner to the actors. These aspects are discussed below.

- Experience collection by the actors: Each actor interacts with its MEC system environment, which is composed of N_m nodes. Each of them involves a constant time operation by step, so the complexity per actor

is $\mathcal{O}(N_m)$. Thus, the number of actors is the number of MEC systems in the system model, so the complexity of presenting M actors, each of them managing N_m nodes, is $\mathcal{O}(M \cdot N_m)$.

- Experience sent to replay memory: The experiences generated by the actors are stored in the replay buffer, and the complexity of including the experiences in the replay buffer is $\mathcal{O}(1)$. The total insertion for M actors, based on N_m nodes each, is $\mathcal{O}(M \cdot N_m)$.

- Model training by the learner: If the learner samples B experiences per update, the complexity for sampling experiences is $\mathcal{O}(B \log N_m))$. The neural network update complexity is $\mathcal{O}(B \cdot P))$, where P is the complexity of a single backpropagation step.

The total time complexity (TTC) of the DDRL-CAAP algorithm depends on the number of nodes N_m and the number of MEC systems M. By combining the elements of the training described above, the total complexity is given by Eq. (15),

$$TTC = \mathcal{O}(M \cdot N_m + B \log N_m + B \cdot P) \tag{15}$$

6. Orchestration Activities

This section describes how the inference model presented in Section 5 is integrated into a real testbed and the architecture's endpoints, interfaces, and workflow for handling application allocation requests needed for the tests. To this end, it also examines into the features and capabilities of the MEO used, highlighting the improvements introduced by this work.

6.1. Integration of the Application Placement Model with the MEC Orchestrator

To distribute and deploy the application placement model, we containerize the trained DRL model and its dependencies using Docker for a production environment. This encapsulates the dependencies within the container, eliminating compatibility problems and the need for manual environment setup on different machines. Encapsulating the DRL model ensures an isolated environment, preventing the issues related to dependencies and system configurations across different MEC systems. Docker enables easy scalability and deployment, allowing the model to run reliably in production without

interference from the host system. In addition, we introduce a REST API to communicate with the MEC orchestrator in order to gather the main features of storage, RAM, and CPU from the nodes, and with the Operation Support System (OSS) to obtain the application requirements. The OSS is responsible for regulating the input of applications, specifying the requirements, arrival rate and duration of the applications coming into the system. After acquiring this data, the REST API engages with the DRL model to initially seek any migration directives, if present, followed by determining the placement decision. Finally, this API returns the node to perform the application placement output. This setup makes it possible to configure specific tests and triggering the model when a new application arrives, facilitating performance metric determination.

The design of the MEO proposed in [40] provides novel endpoints to seamlessly incorporate intelligence into its functionalities. The MEO plays a central role, functioning as both a manager and a singular entry point within the MEC system through a northbound API, enabling functionalities such as onboarding, deployment, information retrieval regarding existing applications, application migration, and deletion. After successful integration, the key operational functionalities of the system DRL and MEO include:

- For incoming requests, the MEO extracts application prerequisites, retrieves system telemetry via customized node-exporters, and subsequently relays this data to the intelligent module.
- Once the model running together with a MEO instance identifies the optimal node for placement, the orchestrator manages resource and application allocation, and lifecycle on the basis of the model decision.

6.2. Application Placement Request Workflow

Figure 5 illustrates the sequence of actions initiated when the MEO receives an instantiation request from the user application through the OSS via the REST API. Note that the diagram presented in this section depicts the inference process of the model for application placement, while the diagram presented previously in Figure 4, Section 5, shows the training process. Initially, the MEO onboards the descriptor file, generating a unique ID for the request. Subsequently, the MEO sends a metric retrieval request to the metric aggregator to obtain telemetry system information for application placement. Then, the MEO triggers a node selection request to the DRL model, with

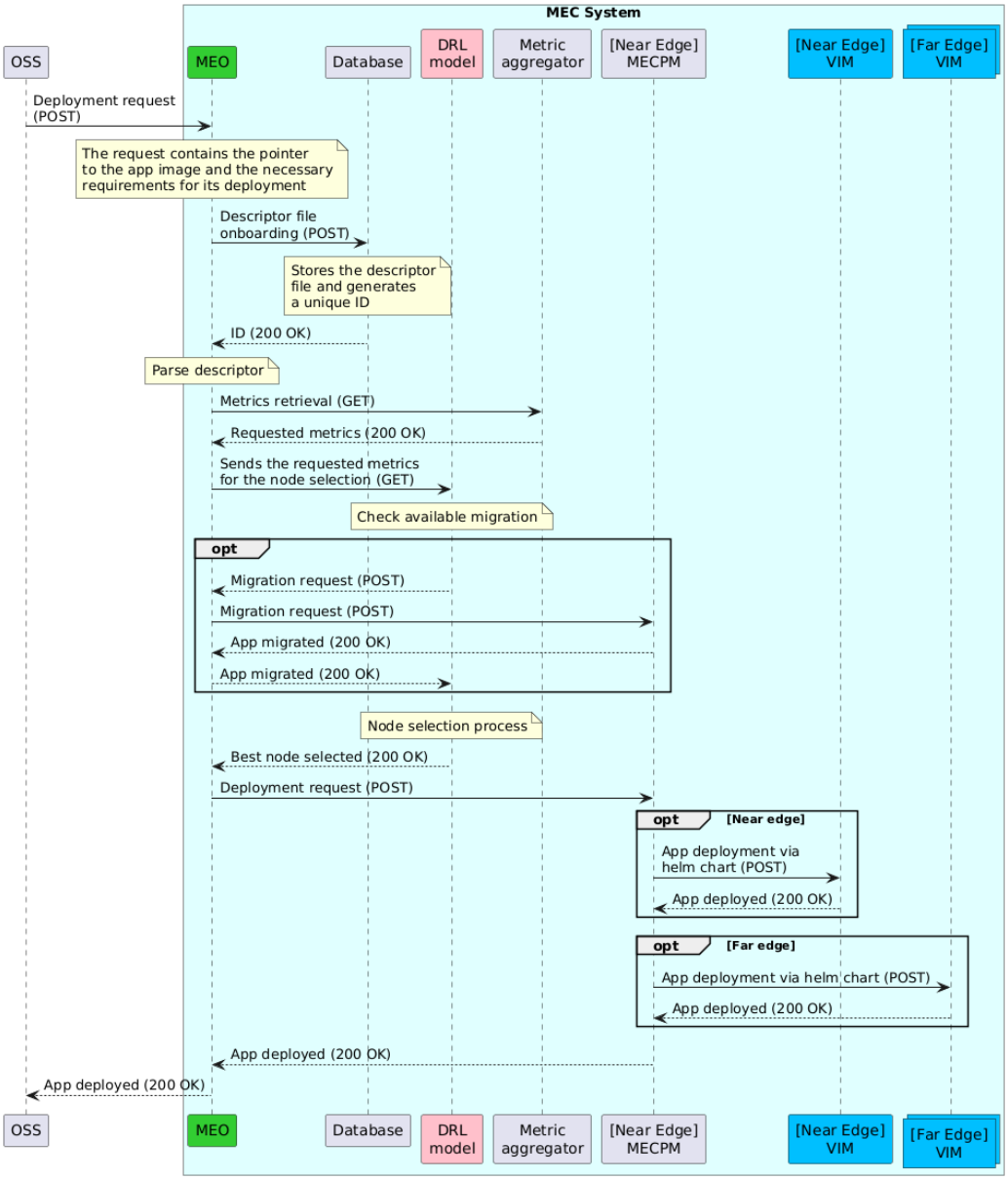


Figure 5: Application placement request workflow.

this request containing the application requirements together with the system telemetry. When acquiring this data, the REST API verifies the feasibility of migrating applications from one node to another, as described in Section 4,

to ensure a minimal active node count. Hence, the model synchronizes with the MEO to acquire the latest availability data from the nodes if migration occurs. The model employs the underlying system's status details and the application's requirements to select the optimal node that maximizes its reward function. As a result, the MEO triggers the placement request on the model's selected node. This deployment request is forwarded to the MEC Platform Manager (MECPM), leading to the effective deployment of the application either on the virtual infrastructure manager at the near or the far edge.

7. Configuration of Test Scenarios

This section outlines the configuration of test scenarios to evaluate the application placement approach described in Section 5, and it describes the methodology employed in training and deployment configuration. The test scenarios are designed to reflect realistic situations that may arise in a distributed MEC system configuration. The model is examined in situations where a MEC system, initially trained with specific types of applications, must adapt to new requirements at the deployment stage due to application migration or shifts in user preferences. For instance, incorporating XR applications to enhance the user navigation experience within the city necessitates the addition of different functionalities and capabilities, as presented in the real-world case study described in Section 5.

7.1. Methodology

The methodology for conducting the evaluation is detailed below, and is based on the architecture introduced in Section 3. The evaluation examines the performance of the application placement approach when hosting applications after the model has been trained to see how well it meets different requirements. The experiments were performed on IoT Fog, IoT Smart, and XR applications, with the aim of targeting applications as close as possible to a real-world scenario. At the start of the algorithm, all nodes are in a state of initial inactivity since no applications exist in the system. To assess the scalability and compliance with the objectives outlined, the following metrics were evaluated:

- The model's accuracy was calculated to ascertain the probability that the model predicts the action associated with the highest reward value

among the available nodes, in order to minimize the number of active nodes.

- Inference time was recorded, this representing the interval between the arrival of an application request and the model's response, which includes the instantiation at the corresponding node.

- Infrastructure information was examined to monitor node availability and the application acceptance rate within the system model. This comprehensive approach provides an overview of the architecture's performance and functionality.

- Power consumption was analyzed to determine the extent to which the proposed method can be used to save power compared with baseline approaches.

- Scalability was assessed in reference to our algorithm's performance across different scenarios. Our evaluation is focused on understanding how our algorithm adapts and performs under various conditions.

7.1.1. Vertical applications description

The applications selected for testing can be divided into three main types: IoT Smart City, IoT Fog, and XR. Our methodology presents these classes to assess the system's impact when applications present varying entry rates and requirements, representing different use cases. The corresponding requirements and application entry rates (λ_k) of applications are shown in Table 3. For each class of application, the references for the data obtained are listed: IoT Smart City [41] (IoT Smart), IoT Fog [41], and XR [42, 43]. Additionally, the applications' entry follows a Poisson distribution, with a duration time for an application ranging from $T_{min} = 150$ s to $T_{max} = 350$ s in simulation scenarios, and following a uniform distribution.

7.1.2. Scenarios description

A series of scenarios were devised for both simulation and a real-world testbed, in order to assess the scalability of the application placement model in different system configurations. To achieve this, variations were performed from one configuration to another, considering the number of nodes, MEC systems, and different application types. In order to evaluate the influence of varying computational capabilities on the algorithm's performance, it is

Type	**Storage (GB)**	**RAM (GB)**	**CPU (vCPU)**	λ_k **(apps/s)**
IoT Smart	[1, 8]	{0.5, 0.7, 0.9, 1.1, 1.3, 1.5}	[1, 5]	2
IoT Fog	[1, 8]	[0.5, 2]	[1, 4]	1
XR	[0.18, 0.55]	[0.5, 2]	[0.05, 0.42]	0.06

Table 3: Applications' specifications.

necessary to consider the distinctive characteristics of the MEC systems in each scenario. The characteristics of the simulation and testbed scenarios are shown in Table 4, while Table 5 describes the objectives of these scenarios for the sake of simplicity. *Scenarios 1-3* are simulation-based, while *Scenarios 4-5* are testbed-based.

The configuration of 4 nodes and 3 MEC systems with IoT Fog applications in *Scenario 3* correlates with *Scenario 4*. The objective of this study was to assess inference time, accuracy, and availability in the same configuration across two distinct environments, testbed and simulation. This methodology enables the assessment of the impact of a real-world scenario within a simulated configuration. *Scenarios 1-3* were intended to demonstrate the scalability of the approach, and were only tested in simulation, as they are characterized by high scalability, which, in turn, necessitates the use of a considerable amount of hardware equipment.

The duration of the simulation was determined in accordance with the

Scenario	**No. Nodes**	**Applications**	**No. MECs**	**Test time (s)**	**Type of test**
1	4, 8, 12, 16, 20	IoT Fog	2	550	Simulation
2	4	IoT Fog	4, 6, 8	550	Simulation
3	4	IoT Fog, IoT Smart, XR	3	50-550	Simulation
4	4	IoT Fog	3	50	Testbed
5	4	IoT Smart	3	50	Testbed

Table 4: Simulation and Testbed Scenarios Characteristics.

Scenario	Description
1	Performance under varying number of nodes
2	Performance under varying number of MEC systems
3	Performance under different entry applications
4	Performance under IoT Fog applications on testbed
5	Performance under IoT Smart applications on testbed

Table 5: Simulation and testbed scenarios description.

entry rate of applications received for each scenario. This approach ensures comprehensive coverage of key events, including the inflow of different application requests, the end of their lifetime, the depletion of system resources, and instances of application rejection. Specifically, with regard to *Scenario 3*, the objective is to demonstrate how the accuracy and system resources vary over time for different applications. For this purpose, the first data sample point of time of 50 s was selected, as this represents a point at which applications have not yet been removed from the system and the system has not reached its maximum capacity for accommodating new applications. This is considered an appropriate starting point for demonstrating the trend in accuracy behavior.

In *Scenario 1*, we test the model's performance under varying numbers of nodes, with 2 MEC systems and 4, 8, 12, 16, and 20 nodes per MEC system. *Scenario 2* tests the influence of a higher number of MEC systems, ranging from 4 to 8, and considering 4 nodes per MEC system in each case. The evaluation aims to demonstrate the adaptability of the proposed approach to different environments and training data, regardless of the locally available data. For *Scenario 1* and *Scenario 2*, IoT Fog applications are included during training and inference with a fixed entry rate $\lambda_F = 1$ apps/s in 550 s of simulation time. The selected simulation time is strategic, as during this period, a significant number of applications have been deployed and a group of them have finished. It is also the point at which some applications have been rejected. Consequently, this timeframe captures the critical events in the simulation.

Tests in *Scenario 3* have the objective of assessing the impact of diverse applications on the system and the responsiveness of MEC systems over a 550 s simulation period. In this scenario, each of the three deployed MEC

systems handled just one of the three applications types, namely IoT Fog ($\lambda_F = 1$ apps/s), IoT Smart ($\lambda_S = 2$ apps/s), or XR ($\lambda_{XR} = 0.06$ apps/s), during the training phase. The DRL model of each MEC system was trained with a different type of application. By contrast, during the inference phase, each MEC system receives all types of applications. This approach evaluates the model's capability to accommodate unforeseen application types during the inference stage, types which were not explicitly assumed for a MEC system during the training. This experimental design facilitates an examination of the DDRL model's adaptive functionality. For simulation-based scenarios, the nodes within each MEC system are fitted with identical computing capabilities, including storage, RAM, and CPU capacity.

In *Scenario 4* and *Scenario 5*, the evaluation is carried out on a real-world testbed to determine how system configurations in a real deployment influence the model's behavior. Within the MEC system under consideration, each node possesses distinct initial capabilities regarding storage, CPU, and RAM, mirroring the typical heterogeneity found in actual MEC systems. With regards the simulation scenarios, the tests were repeated 100 times, whereas in the testbed scenarios, they were repeated ten times, including a 95% confidence interval.

7.2. Simulator Configuration

For the simulation-based scenarios, we employed Simu5G [44, 45], which is an open-source framework capable of simulating a 5G network. It uses the OMNET++ simulator as a base and introduces a 5G MEC architecture, enabling the onboarding of MEC applications. Thus, the simulation tests were performed on Simu5G with an API REST server on the host machine. This enables the exchange of information between the simulation environment and the Python libraries during the simulation and inference stages.

The MEC orchestrator decides whether to instantiate applications on MEC nodes, and the server serializes the already trained DDRL model and uses it to provide a response to the simulation environment. On the basis of this response, the MEC orchestrator in the simulation environment takes appropriate actions. Although Simu5G supports MEC scenarios with a MEC orchestrator and nodes, it lacks communication between orchestrators.

Table 6 shows the capacity values for each node per MEC system in simulation. The configurations reflect off-the-shelf edge servers from different vendors. In particular, the Intel servers were Intel NUC BKNUC9VXQNX1, Intel NUC 9 Pro 9V7QNX, and Intel NUC 11 NUC11PAHi5.

7.3. Testbed Configuration

For production deployments, the DRL model and MEO are containerized, utilizing the communication interfaces explained in Section 6. Table 7 presents the computational capabilities of each node inside the MEC system. To enhance the functionality of the testbed, this work employs minor computational capacity servers for application placement in relation to the ones utilized for simulation. This strategic choice ensures a more realistic representation of resource utilization. Furthermore, to establish a coherent temporal mapping, we extrapolate the lifespan of applications from the simulation environment to the testbed setting to prevent servers from running out of space prematurely. For this purpose, we decrease the duration time of the applications (T_{min} = 10 s and T_{max} = 25 s). The simulation time for this case is set to 50 s; during this time, assessing the aforementioned critical system events is possible. However, it should be noted that the metrics aggregator was updated at a frequency of 30 seconds. Consequently, from the moment an application was hosted, a 30-second interval was observed before launching the other to allow the metrics aggregator to update correctly and acquire the infrastructure data. Therefore, this resulted in the duration of applications and tests having a scaling factor of 30 seconds. A single test, with $\lambda_F = 1$ apps/s and $\lambda_S = 2$ apps/s, had a duration of 30 minutes and 60 minutes, respectively. Thus, only ten tests were performed on the testbed due to the longer duration. For clarity and readability, the values in the text

# MEC	**Storage (GB)**	**RAM (GB)**	**CPU (vCPU)**	**Brand**, **Model**
1	1024	64	128	Intel NUC
2	1024	32	72	PowerEdge R250
3	500	32	32	OptiPlex XE4
4	1024	64	32	PowerEdge T550
5	1024	64	72	Intel NUC 9 Pro
6	500	8	32	Intel NUC 11
7	1024	64	72	Intel NUC 9 Pro
8	500	8	32	Intel NUC 11

Table 6: Computational capabilities of nodes per MEC system (simulation).

# Node	Storage (GB)	RAM (GB)	CPU (vCPU)	Brand, Model
1	957	64	16	Intel NUC
2	951	64	12	Intel NUC 9 Pro
3	267	32	8	VM in SuperM
4	109	32	8	VM in SuperM

Table 7: Computational capability of each node (testbed).

are presented without considering the 30-second wait.

7.4. Neural Network Configuration

The neural network from the DRL model is composed of an input layer, two hidden layers, and an output layer, with the input layer having $b \cdot N_m + |D_r(k)|$ features as input data. In other words, the model colocated with each MEO has several input features proportional to the number of MEC nodes multiplied by the number of parameters considered (i.e., in this work, RAM, CPU, and storage availability) and the application requirements under these parameters. The first and the second hidden layers have η nodes each and use *relu* as an activation function, where $\eta = 128$. The output layer contains N_m features and uses the *linear* activation function. Note that there is a difference between the number of nodes in a MEC system (N_m) and the number of nodes of a neural network layer, η.

8. Performance Results

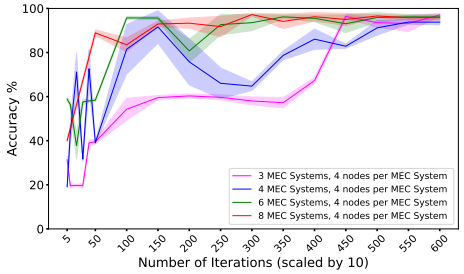
This section presents the results of the algorithm evaluation in different configuration scenarios. The objective is to draw conclusions about its performance in simulation and real-world environments. By examining its behavior under a variety of conditions, we aim to understand the model's strengths, limitations, and the trade-offs arising from minimizing the number of active nodes. The metrics presented in Section 7 are analyzed to evaluate the inference performance of the proposed approach.

Figure 6a presents the convergence rate for different number of MEC systems, Figure 6b shows the convergence rate for different number of nodes, and the training time for these configurations is depicted in Figure 6c. Figure 7a and Figure 7b illustrate the accuracy obtained for *Scenario 1*, whereas

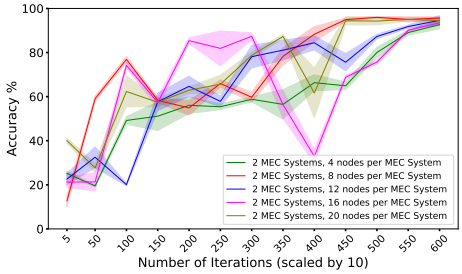
Figure 7c presents a comparison between the power consumption of DDRL-CAAP and the baseline algorithms in this scenario. The accuracy results for *Scenario 2* are shown in Figure 8. Additionally, Figure 9, Figure 10, and Figure 11 show accuracy values for IoT Fog, IoT Smart and XR applications, respectively, in *Scenario 3*. The availability and rate of accepted applications are analyzed in Figure 12. With regards to the scenarios evaluated on the testbed, Figure 13 illustrates the accuracy for *Scenario 4* and *Scenario 5*. Furthermore, Figure 14 examines the availability and the rate of accepted applications. The results are presented in detail and discussed further in this section.

The convergence rate has been analyzed for different scenarios, reflecting the accuracy over training episodes, and this allows us to determine the number of iterations that are required to achieve the global optimum. The global optimum reference determines the point of the training processes where the algorithm could be used for deployment in the different scenarios. Fig. 6a and Fig. 6b presents the convergence rate for different MEC systems and different number of nodes, respectively. The figures show that as the number of MEC systems, and consequently the number of actors in the network, increases, the accuracy reaches its global optimum in fewer iterations.

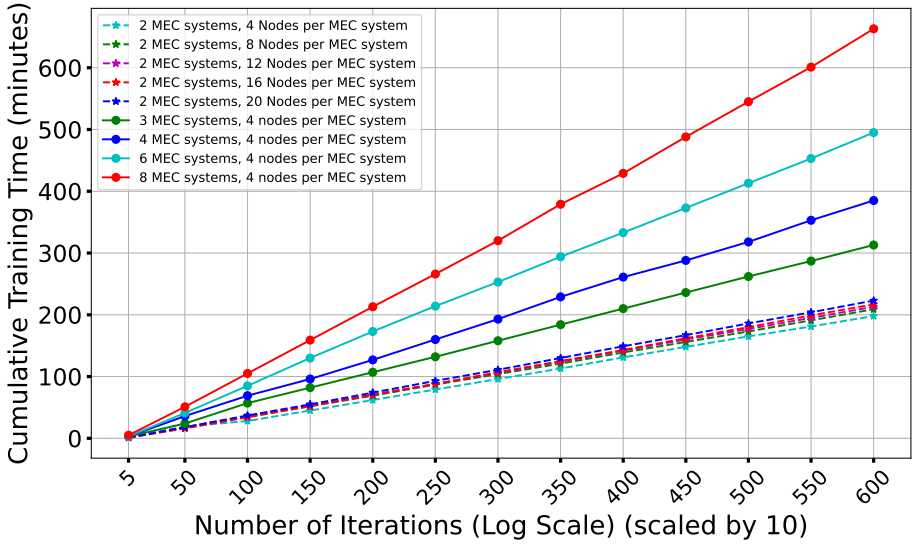
Figure 6c presents the training time in minutes for a different number of MEC systems (M) and a different number of nodes per MEC system (N_m). The time complexity presented in Section 5 indicates the dependence on these parameters. The curves in this figure reflects that the training time of the algorithm increases with the number of nodes and MEC systems. However, training time is found to increase most significantly in direct proportion to the number of MEC systems. As the number of workers increases, the demand for computational resources such as CPU, GPU, and memory also rises. The training system implemented on a single machine experiences contention, as the multiple worker processes compete for the same resources, potentially leading to slower processing times. The work in [37] presented the ApeX approach, achieving an acceptable performance for specific configurations, such as Atari games, while necessitating the use of a significant number of actors. However, it is worth studying various distributed deep reinforcement learning techniques in future research. Such a study would allow a comprehensive evaluation of training times and the accuracy of these methods, in relation to the approach presented in this paper.



(a) DDRL-CAAP convergence rate for different number of MEC systems.



(b) DDRL-CAAP convergence rate for different number of nodes.

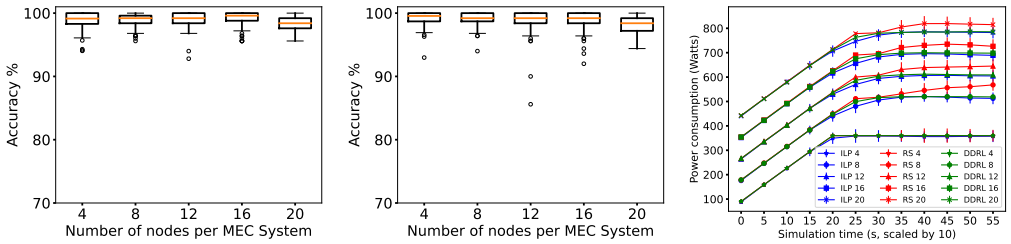


(c) Training time of DDRL-CAAP algorithm.

Figure 6: DDRL-CAAP convergence rate and training time for the different scenarios.

8.1. Scenario 1: Performance with Different Number of Nodes

The results for *Scenario 1* regarding the model's scalability in terms of different numbers of nodes per MEC system are presented in Figure 7. In particular, we provide a comparative analysis of the accuracy metrics for two different MEC systems at the 250 s mark. This timestamp has been chosen to present the accuracy results because, up to this point, the critical events of application ingress, egress, and, therefore, application migration have occurred.



(a) DDRL-CAAP test in MEC system 1.

(b) DDRL-CAAP test in MEC system 2.

(c) Power consumption compared with benchmark algorithms.

Figure 7: Inference accuracy and power consumption performance for different number of nodes (*Scenario 1*).

During training, the model was incorporated into an architecture with two MEC systems, and we demonstrated its performance in both MEC systems separately during inference. As can be seen, the accuracy is not significantly affected by the increase in the number of nodes. The model supports 20 nodes while maintaining an average accuracy above 98.1% across all cases. In certain configurations, it can achieve up to 99.1% accuracy. Figure 7a and Figure 7b indicate that the model performs similarly in both MEC systems.

Additionally, the power consumption for all nodes is measured across a simulation time of 550 s and compared with random selection (RS) and the iterative ILP (One-Step-CAAP) algorithms in Figure 7c. To facilitate a finer appreciation of the graphs in this figure and in the *Scenario 3* graphs, the x-axis has been scaled by a factor of 10.

The Random Selection algorithm randomly selects between the nodes that satisfy the constraints described in Section 4. The DDRL-CAAP algorithm outlines the anticipated behaviour of the MEC orchestrator in the absence of an integrated application placement algorithm. Power consumption in relation to CPU is selected as a baseline to determine the relation between CPU and power consumption from the Scaphandre metrics as per the model in work [7]. We aim to demonstrate how our proposed DDRL-CAAP algorithm can reduce the nodes' overall power consumption compared with existing baseline methods during simulation time. In this case, the ILP algorithm outperforms the DDRL approach with a mean consumption reduction of 2.41%. The DDRL approach, in turn, outperforms the Random Selection algorithm with an average consumption reduction of 4.35%.

It can be stated that, for 4 nodes, the algorithms do not differ with regards to power consumption, except for the case of 200 s. During this simulation time, the ILP approach outperforms DDRL. The observed performance with 4 nodes is consistent across different algorithms when the CPU consumption reaches 70% of total capacity or higher. At these levels of utilization, the nodes are rapidly approaching their maximum capacity, resulting in a similar overall power consumption regardless of the algorithm employed. After approximately 200 s, the system reaches its saturation point, and power consumption levels off as a result.

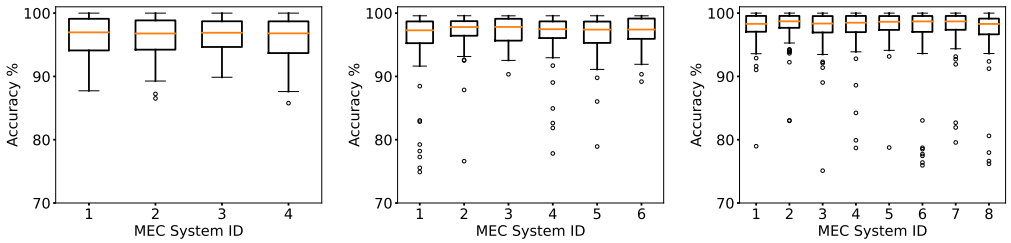
From the eight-node configuration upwards, there is a clear difference between ILP and DDRL regarding Random Selection. The Random Selection algorithm increases power consumption by randomly selecting nodes, which leads to a higher probability of activating more nodes. In contrast, the DDRL algorithm effectively minimizes the number of active nodes, thereby reducing overall power consumption.

8.2. Scenario 2: Performance with Different Number of MEC systems

Conversely, Figure 8 shows the scalability regarding the number of MEC systems for training, corresponding to *Scenario 2*. The experiments were conducted using varying numbers of MEC systems, ranging from 4 to 8, and considered the inference results for each training configuration. This test was performed using IoT Fog applications, with a training configuration of 4 nodes per MEC system and a 550 s simulation time. The accuracy results are presented at the 250 s mark, as in the previous scenario. Figure 8a depicts the results for $M = 4$, achieving an average accuracy of over 96.1% in all cases. The analysis of 6 MEC systems (Figure 8b) yielded an accuracy above 95.7%, and in the case of $M = 8$ (Figure 8c) the accuracy results reached 96.9%. Nonetheless, the lower whisker limits increase from 4 to 8 MEC systems, indicating a higher data dispersion. As the number of MEC systems increases, so does the ability of the actors in the DDRL model to synchronize their actions, and the sharing of model weights reduces the variance.

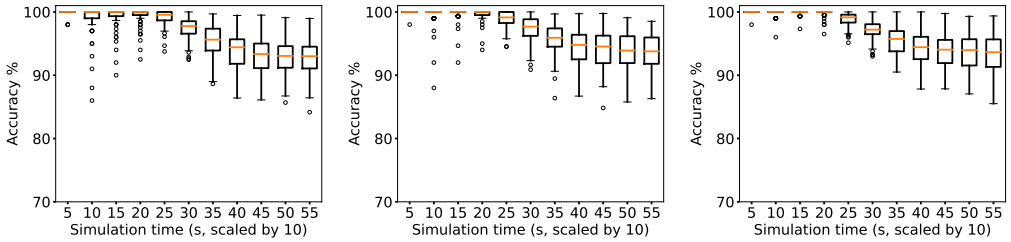
8.3. Scenario 3: Performance with Different Applications

The test in *Scenario 3* aims to demonstrate the model's performance when encountering different applications during inference. Three MEC systems were provided with data related to IoT Fog, IoT Smart, and XR applications during training.



(a) Test with 4 MEC systems. (b) Test with 6 MEC systems. (c) Test with 8 MEC systems.

Figure 8: DDRL-CAAP inference accuracy for different number of MEC systems, 4 nodes per MEC system (*Scenario 2*).

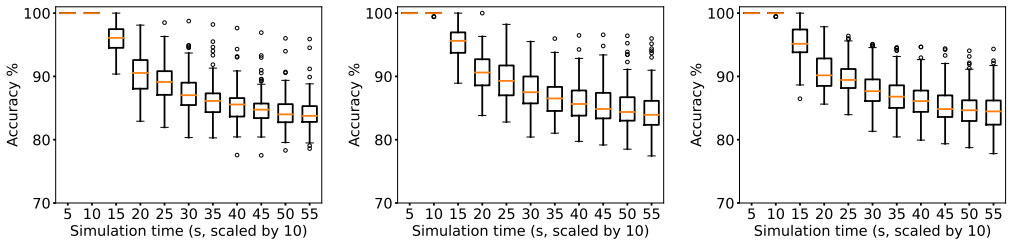


(a) Test in MEC system 1. (b) Test in MEC system 2. (c) Test in MEC system 3.

Figure 9: DDRL-CAAP inference accuracy for different MEC systems, 4 nodes per MEC system with IoT Fog (*Scenario 3*).

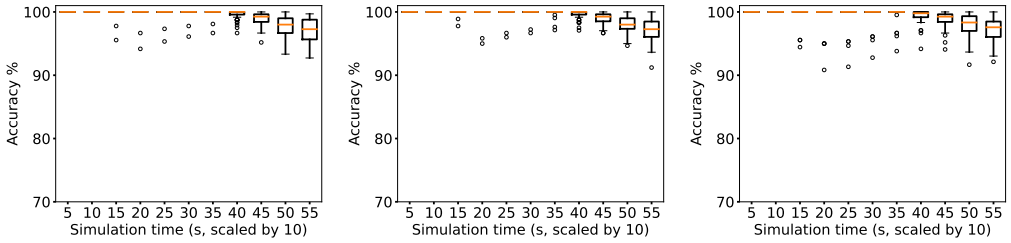
A comparison between Figure 9 and Figure 10 reveals that as IoT Smart applications exhibit a faster entry rate, compared with IoT Fog, the infrastructure saturates earlier. For the case of Figure 11, as applications enter the system with a low arrival rate, the applications are maintained with an acceptable accuracy performance during the simulation phases. Consequently, the accuracy results for IoT Smart applications declines more rapidly than for IoT Fog applications. This observation underscores the model's acceptable performance under a high volume of applications. For these tests, it can be noticed that the values remain relatively stable at a certain point in the simulation because the system starts to saturate. New applications are incorporated onto the available resources left by those that have exited the system, and the model operates accurately.

In addition to analyzing the accuracy of each simulated scenario, this work examines the behavior of system resource availability as applications



(a) Test in MEC system 1. (b) Test in MEC system 2. (c) Test in MEC system 3.

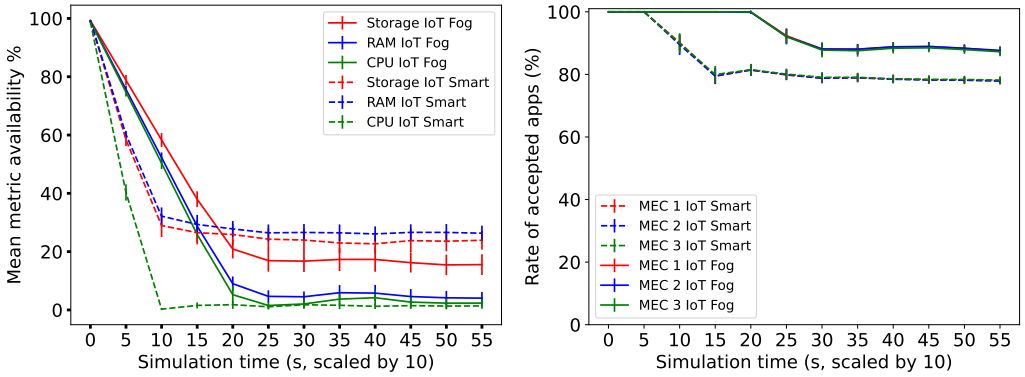
Figure 10: DDRL-CAAP inference accuracy for different MEC systems, 4 nodes per MEC system with IoT Smart (*Scenario 3*).



(a) Test in MEC system 1. (b) Test in MEC system 2. (c) Test in MEC system 3.

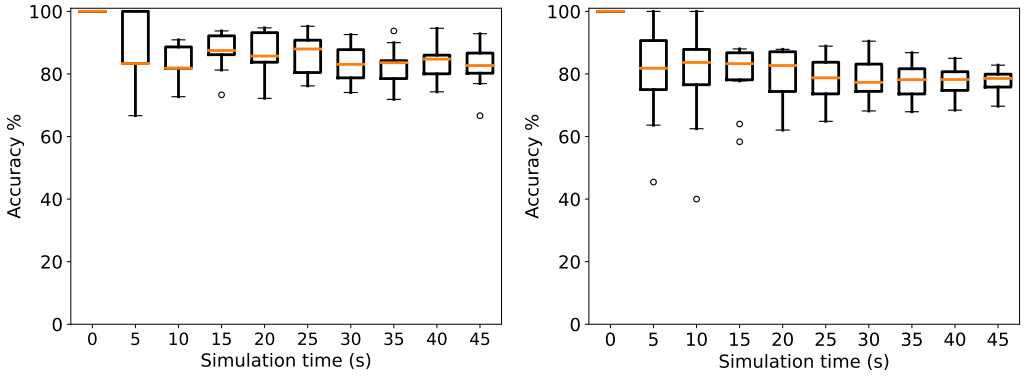
Figure 11: DDRL-CAAP inference accuracy for different MEC systems, 4 nodes per MEC system with XR (*Scenario 3*).

are introduced into the system. In this regard, Figure 12a presents the infrastructure resource availability over the whole simulation time of MEC system 1 for IoT Fog and IoT Smart applications regarding storage, CPU, and RAM metrics, corresponding to *Scenario 3*. The availability ends in the case of IoT Smart applications because CPU capacity is exhausted before RAM and storage when the simulation time reaches 100 s. However, in the case of IoT Fog, both RAM and CPU reach the minimum almost simultaneously after 200 s. These results align with the accepted application rate shown in Figure 12b. IoT Smart applications start being rejected by the system after 100 s, while IoT Fog applications start after 200 s, confirming the results shown in Figure 9 and Figure 10.



(a) Availability of nodes in MEC system 1. (b) Rate of applications accepted for placement.

Figure 12: Inference-availability of nodes and rate of applications accepted for placement (*Scenario 3*).

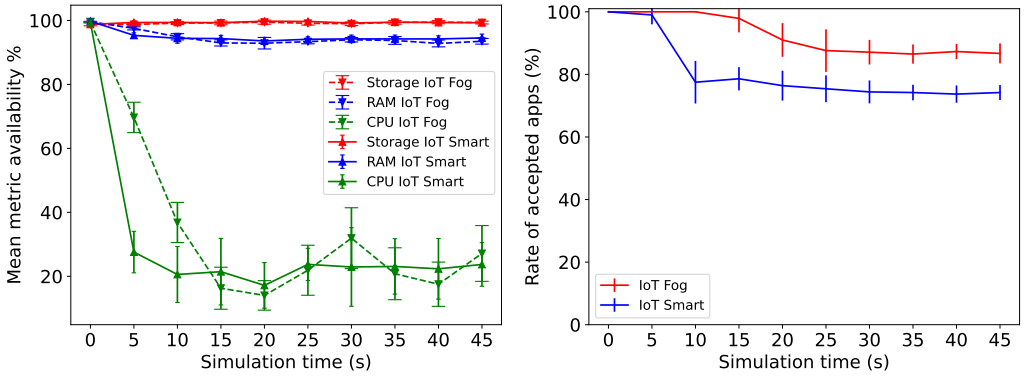


(a) IoT Fog applications *(Scenario 4)*. (b) IoT Smart applications *(Scenario 5)*.

Figure 13: Inference accuracy for IoT Fog and Smart applications (*Scenario 4* and *Scenario 5*), respectively.

8.4. Scenario 4 and Scenario 5: Performance for IoT Fog and IoT Smart Applications (Testbed)

As we show in Table 4, to experiment with the model in a real-world environment, we consider two testbed scenarios, namely *Scenario 4* and *Scenario 5*. Each one has the same configuration, but they are differentiated by the types of applications: *Scenario 4* considers IoT Fog applications, while



(a) Availability of nodes.

(b) Rate of applications accepted for placement.

Figure 14: DDRL-CAAP inference-availability of nodes and rate of applications accepted for placement (*Scenario 4* and *Scenario 5*).

Scenario 5 IoT Smart applications.

For these two scenarios, we evaluate the accuracy results in Figure 13a and Figure 13b. The tests for *Scenario 4* and *Scenario 5* demonstrate an underperformance compared with the simulation. The model follows an equal tendency to that in simulation, and when the simulation time of applications corresponds to the time when the system is almost full of applications, the model's performance starts to decrease. According to Figure 14b, the applications start to be rejected in the MEC system at 10 s for IoT Smart applications and, in the case of IoT Fog, at 20 s.

The above behavior is corroborated by the availability curve shown in Figure 14a, which demonstrates that in the context of IoT Smart and IoT Fog applications, the nodes reach full occupancy at 10 s and 20 s, respectively. These graphs show a trend that is similar to that in *Scenario 3*, but with lower accuracy due to reduced capacity and quicker saturation. When a node is almost reaching its total capacity, the model could choose another available node to perform the placement. As a result, in cases where a new application could have been accommodated on the previous node with available resources but is instead placed on the next node, it is considered a case with an accuracy equal to zero. The ApeX algorithm, based on priority distribution, does not typically interpret this situation as a priority, leading to a degradation when this event occurs.

#	No. Nodes	Apps	No. MECs	Inf. Time	CI Inf. Time	Inf. Time ILP
1	4	IoT Fog	2	0.442	(0.435, 0.450)	14.306
	8	IoT Fog	2	0.435	(0.426, 0.443)	22.925
	12	IoT Fog	2	0.461	(0.453, 0,467)	25.602
	16	IoT Fog	2	0.426	(0.419, 0.432)	30.062
	20	IoT Fog	2	0.440	(0.435, 0.445)	31.260
2	4	IoT Fog	4	0.323	(0.316, 0.330)	23.206
	4	IoT Fog	6	0.443	(0.436, 0.451)	27.170
	4	IoT Fog	8	0.342	(0.332, 0.352)	31.010
3	4	IoT Fog	3	0.333	(0.322, 0.344)	21.545
	4	IoT Fog	3	0.378	(0.367, 0.390)	21.350
	4	IoT Fog	3	0.368	(0.355, 0.380)	22.109
	4	IoT Smart	3	0.323	(0.315, 0.331)	22.293
	4	IoT Smart	3	0.299	(0.289, 0.310)	22.321
	4	IoT Smart	3	0.333	(0.324, 0.343)	21.837
	4	XR	3	0.349	(0.339, 0.360)	20.292
	4	XR	3	0.333	(0.326, 0.340)	20.229
	4	XR	3	0.368	(0.360, 0.377)	20.223
4	4	IoT Fog	3	0.630	(0.573, 0.688)	-
5	4	IoT Smart	3	0.672	(0.603, 0.728)	-

Table 8: DDRL-CAAP inference time and ILP computation time (ms) for different scenarios (average and confidence interval).

8.5. General Performance Discussions

Table 8 presents the results obtained for inference time for the different scenarios described. The mean and confidence interval (CI) of 100 tests were calculated for each application's arrival rate to determine the inference time for each scenario. The same approach was used for the testbed scenarios, with ten tests conducted for each scenario. The evaluation of the performance of the iterative ILP (One-Step-CAAP) reflects that the accuracy is always 100%. However, the average computing time to place one application is significantly higher, as shown in Table 9 compared with Table 8, for all the simulated scenarios presented in Table 4. The average inference time reduction of DDRL over ILP is 98.3%. Given the extended inference time required for completion as evidenced in the simulation, ILP tests were not conducted on the real-world

Conf.	Inf. Time	CI Inf. Time	Conf.	Inf. Time	CI Inf. Time
No. Nodes	**Scenario 1**		**Apps**	**Scenario 3**	
4	14.306	(14.285, 14.328)	IoT Fog	21.545	(21.516, 21.573)
8	22.925	(22.897, 22.954)	IoT Fog	21.350	(21.323, 21.376)
12	25.602	(25.573, 25.631)	IoT Fog	22.109	(22.080, 22.138)
16	30.062	(30.033, 30.090)	IoT Smart	22.293	(22.265, 22.322)
20	31.260	(31.227, 31.293)	IoT Smart	22.321	(22.293, 22.349)
No. MECs	**Scenario 2**		IoT Smart	21.837	(21.809, 21.864)
			XR	20.292	(20.275, 20.309)
4	23.206	(23.182, 23.229)	XR	20.229	(20.215, 20.243)
6	27.170	(27.138, 27.200)	XR	20.223	(20.210, 20.236)
8	31.010	(30.980, 31.041)			

Table 9: ILP computation time (ms) (average and confidence interval) for the same scenarios as in Table 8.

Note: Conf: Configuration of the scenarios, please refer to Table 8 for the remaining configuration parameters of each scenario.

testbed of *Scenario 4* and *Scenario 5*.

In the simulation scenarios, the time data is very similar across all the MEC systems and even between the different types of applications. Nevertheless, in the context of testbed configurations, a slight increase in inference time is discernible compared with the scenarios in the simulation. The involvement of all components from the moment the request enters the system until the decision node places the application, results in an increase in inference time that is higher than those observed in simulated scenarios. Nevertheless, despite the increase in inference time, the values remain within an acceptable range in order for the model to be deployed in a real-world production environment.

9. Conclusions

We have proposed a MEC application placement approach considering vertical applications with random lifecycle time in distributed MEC system environments. The main goal was to minimize the number of active nodes and maximize the model's accuracy. To solve the application placement problem, we proposed Integer Linear Programming and a distributed deep

reinforcement learning algorithm, and the approaches were tested in a simulation environment. We containerized the model as this provides better performance and less inference time to work in a real environment considering scenarios with 4 nodes. The results demonstrated the model's adaptability to environments with varying initial resource availability on the nodes, different number of nodes and MEC systems, achieving highly accurate application placement. The DDRL-CAAP algorithm improves power consumption in comparison with the Random Selection algorithm by an average of 4.35%. By analyzing the power consumption results and using the Random Selection as a baseline, the value obtained represents how much energy consumption is reduced in average when using DDRL-CAAP algorithm. Furthermore, considering the ILP algorithm as a baseline, the DDRL-CAAP algorithm presents an average reduction in inference time of 98.3%. Future research will explore the influence on the algorithm of volatile infrastructures, taking into account MEC node disconnection during training and inference.

Acknowledgements

This work has been performed in the framework of the European Union's H2020 project AI@EDGE, co-funded by the EU under grant agreement No 101015922. The authors would like to acknowledge CERCA Programme/ Generalitat de Catalunya for sponsoring part of this work. This work has also been supported by the EU "NextGenerationEU/PRTR", MCIN, by AEI (Spain) under project IJC2020-043058-I, by MCIN/AEI/10.13039/501100011033 (FEDER "a way of making Europe") under grant PID2022-142332OA-I00, and by the Spanish Ministry of Economic Affairs and Digital Transformation and the European Union – NextGeneration EU, in the framework of the Recovery Plan, Transformation and Resilience (PRTR) (Call UNICO I+D 5G 2021, ref. number TSI-063000-2021-9-6GSMART-ICC). This work was also supported by the grant ONOFRE-3 PID2020-112675RB-C43, funded by MCIN/AEI/10.13039/501100011033.

References

[1] ETSI, MEC federation: deployment considerations, White Paper ETSI White Paper No. 49, European Telecommunications Standards Institute (Jun. 2022).

[2] B. Mao, F. Tang, Y. Kawamoto, N. Kato, AI Models for Green Communications Towards 6G, IEEE Commun. Surv. Tut. 24 (1) (2022) 210–247.

[3] A. Sufyan, K. B. Khan, O. A. Khashan, T. Mir, U. Mir, From 5G to beyond 5G: A comprehensive survey of wireless network evolution, challenges, and promising technologies, Electronics 12 (10) (2023) 2200.

[4] Y. Shi, Y. Yang, C. Yi, B. Chen, J. Cai, Toward online reliability-enhanced microservice deployment with layer sharing in edge computing, IEEE Internet of Things Journal 11 (13) (2024) 23370–23383.

[5] C. Ying, Z. Zhao, C. Yi, Y. Shi, J. Cai, An AoTI-driven joint sampling frequency and access selection optimization for industrial wireless sensor networks, IEEE Transactions on Vehicular Technology 72 (9) (2023) 12311–12325.

[6] Y. Shi, C. Yi, R. Wang, Q. Wu, B. Chen, J. Cai, Service migration or task rerouting: A two-timescale online resource optimization for MEC, IEEE Transactions on Wireless Communications 23 (2) (2024) 1503–1519.

[7] C. Centofanti, J. Santos, V. Gudepu, K. Kondepu, Impact of power consumption in containerized clouds: A comprehensive analysis of open-source power measurement tools, Comput. Netw. 245 (2024) 110371.

[8] S. Radhika, P. Rangarajan, Fuzzy based sleep scheduling algorithm with machine learning techniques to enhance energy efficiency in wireless sensor networks, Wireless Pers Commun 118 (4) 3025–3044.

[9] D. Tchuani Tchakonté, E. Simeu, M. Tchuente, Lifetime optimization of wireless sensor networks with sleep mode energy consumption of sensor nodes, Wireless Netw 26 (1) (2020) 91–100.

[10] S. Tang, B.-S. Lee, B. He, Towards economic fairness for big data processing in pay-as-you-go cloud computing, in: 2014 IEEE 6th International Conference on Cloud Computing Technology and Science, 2024, pp. 638–643.

[11] H. Badri, T. Bahreini, D. Grosu, K. Yang, Energy-aware application placement in mobile edge computing: A stochastic optimization approach, IEEE Trans. Parall. Distr. 31 (4) (2020) 909–922.

[12] G. Perin, M. Berno, T. Erseghe, M. Rossi, Towards sustainable edge computing through renewable energy resources and online, distributed and predictive scheduling, IEEE Trans. Netw. Serv. Manag. 19 (1) (2022) 306–321.

[13] G. Premsankar, B. Ghaddar, Energy-efficient service placement for latency-sensitive applications in edge computing, IEEE Internet things 9 (18) (2022) 17926–17937.

[14] C. Torres-Pérez, E. Coronado, C. Cervelló-Pastor, , M. S. Siddiqui, Distributed learning for application placement at the edge minimizing active nodes, in: Proc. of 6GNet, Paris, France, 2023, pp. 1–4.

[15] ETSI, Multi-access Edge Computing (MEC); Framework and Reference Architecture, Group Specification (GS) MEC 003, European Telecommunications Standards Institute, version 3.2.1 (Apr. 2024).

[16] R. Aghazadeh, A. Shahidinejad, M. Ghobaei-Arani, Proactive content caching in edge computing environment: A review, Software: Practice and Experience 53 (3) (2023) 811–855.

[17] M. Ghorbian, M. Ghobaei-Arani, L. Esmaeili, A survey on the scheduling mechanisms in serverless computing: a taxonomy, challenges, and trends, Cluster Computing 27 (5) (2024) 5571–5610.

[18] M. Tari, M. Ghobaei-Arani, J. Pouramini, M. Ghorbian, Auto-scaling mechanisms in serverless computing: A comprehensive review, Computer Science Review 53 (2024) 100650.

[19] R. Mahmud, S. N. Srirama, K. Ramamohanarao, R. Buyya, Quality of experience (QoE)-aware placement of applications in fog computing environments, J. Parallel Distr. Com. 132 (2019) 190–203.

[20] E. Badidi, Qos-aware placement of tasks on a fog cluster in an edge computing environment, J. Ubiquitous Syst. Pervasive Netw. 13 (1) (2020) 11–19.

[21] K. Kaur, F. Guillemin, V. Q. Rodriguez, F. Sailhan, Latency and network aware placement for cloud-native 5G/6G services, in: Proc. of IEEE CCNC, Las Vegas, USA, 2022, pp. 114–119.

[22] M. Salimian, M. Ghobaei-Arani, A. Shahidinejad, An evolutionary multi-objective optimization technique to deploy the IoT services in fog-enabled networks: An autonomous approach, Applied Artificial Intelligence 36 (1) (2022) 2008149.

[23] C. Yi, S. Huang, J. Cai, An incentive mechanism integrating joint power, channel and link management for social-aware D2D content sharing and proactive caching, IEEE Transactions on Mobile Computing 17 (4) (2018) 789–802.

[24] C. Yi, S. Huang, J. Cai, Joint resource allocation for device-to-device communication assisted fog computing, IEEE Transactions on Mobile Computing 20 (3) (2021) 1076–1091.

[25] Y. Yang, Y. Shi, C. Yi, J. Cai, J. Kang, D. Niyato, X. Shen, Dynamic human digital twin deployment at the edge for task execution: A two-timescale accuracy-aware online optimization, IEEE Transactions on Mobile Computing (12) 1–16.

[26] H. Sami, H. Otrok, J. Bentahar, A. Mourad, AI-based resource provisioning of IoE services in 6G: A deep reinforcement learning approach, IEEE Trans. Netw. Serv. Man. 18 (3) (2021) 3527–3540.

[27] J. S. Camargo, E. Coronado, C. Torres-Pérez, J. Palomares, M. S. Siddiqui, DQN-based intelligent application placement with delay-priority in multi MEC systems, in: Proc. of EuCNC/6G Summit, Gothenburg, Sweden, 2023, pp. 460–465.

[28] M. Goudarzi, M. Palaniswami, R. Buyya, A distributed deep reinforcement learning technique for application placement in edge and fog computing environments, IEEE T. Mobile Comput. 22 (5) (2023) 2491–2505.

[29] D. Wei, J. Ma, L. Luo, Y. Wang, L. He, X. Li, Computation offloading over multi-UAV MEC network: A distributed deep reinforcement learning approach, Computer Networks 199 (2021) 108439.

[30] B. Hu, Z. Cao, M. Zhou, Scheduling real-time parallel applications in cloud to minimize energy consumption, IEEE Trans. Cloud Comput. 10 (1) (2022) 662–674.

[31] F. Jazayeri, A. Shahidinejad, M. Ghobaei-Arani, A latency-aware and energy-efficient computation offloading in mobile fog computing: a hidden markov model-based approach, The Journal of Supercomputing 77 (5) (2021) 4887–4916.

[32] A. Shahidinejad, F. Farahbakhsh, M. Ghobaei-Arani, M. H. Malik, T. Anwar, Context-aware multi-user offloading in mobile edge computing: a federated learning-based approach, Journal of Grid Computing 19 (2) (2021) 18.

[33] C. Centofanti, W. Tiberti, A. Marotta, F. Graziosi, D. Cassioli, Taming latency at the edge: A user-aware service placement approach, Computer Networks 247 (2024) 110444.

[34] J. Santos, C. Wang, T. Wauters, F. De Turck, Diktyo: Network-aware scheduling in container-based clouds, IEEE Trans. on Netw. and Serv. Manag. 20 (4) (2023) 4461–4477.

[35] ETSI, Multi-access Edge Computing (MEC); Federation enablement APIs, Group Specification (GS) MEC 040, European Telecommunications Standards Institute, version 3.2.1 (Mar. 2024).

[36] Y. Li, Deep reinforcement learning: An overview, arxiv preprint arXiv:1701.07274 (Nov. 2018).

[37] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, D. Silver, Distributed prioritized experience replay, arxiv preprint arXiv:1803.00933 (Mar. 2018).

[38] M. R. Samsami, H. Alimadad, Distributed deep reinforcement learning: An overview, arxiv preprint arXiv:2011.11012 (Nov. 2020).

[39] R. Chen, C. Yi, K. Zhu, B. Chen, J. Cai, M. Guizani, A three-party hierarchical game for physical layer security aware wireless communications with dynamic trilateral coalitions, IEEE Transactions on Wireless Communications 23 (5) (2024) 4815–4829.

[40] AI@EDGE, D2.3 consolidated system architecture, interfaces specifications, and techno-economic analysis, accessed Aug 2024 (2023).
URL `https://shorturl.at/0KKmf`

[41] A. S. Ibrahim, K. Y. Youssef, H. Kamel, M. Abouelatta, Traffic modelling of smart city internet of things architecture, IET Commun. 14 (8) (2020) 1275–1284.

[42] R. Fantacci, B. Picano, Edge-based virtual reality over 6G terahertz channels, IEEE Network 35 (5) (2021) 28–33.

[43] T. Taleb, A. Boudi, L. Rosa, L. Cordeiro, T. Theodoropoulos, K. Tserpes, P. Dazzi, A. I. Protopsaltis, R. Li, Toward supporting XR services: Architecture and enablers, IEEE Internet Things 10 (4) (2023) 3567–3586.

[44] G. Nardini, D. Sabella, G. Stea, P. Thakkar, A. Virdis, Simu5G–an OMNeT++ library for end-to-end performance evaluation of 5G networks, IEEE Access 8 (2020) 181176–181191. doi:10.1109/ACCESS.2020.3028550.

[45] A. Noferi, G. Nardini, G. Stea, A. Virdis, Deployment and configuration of MEC apps with simu5G, arxiv preprint arXiv:2109.12048 (Sep. 2021).