

ARTICLE INFO

Keywords:

DQN, Edge Computing, Optimization

ABSTRACT

The fifth generation (5G) of mobile network offers a remarkable degree of flexibility to mobile operators, enabling them to provide users with effective and tailored network services. Software Defined Networking (SDN), Network Function Virtualization (NFV), and edge computing have given the operator the opportunity to easily bring computational capacity to the edge and to support latency-sensitive services. While 5G standards have defined the technological and architectural frameworks to orchestrate services, finding effective resources management and QoS optimization policies is still an open research issue.

In this paper, we propose an online orchestration methodology for a multi-user edge service. The orchestrator goal is to simultaneously maximize the QoS, and minimize the amount of resources needed.

We provide a mathematical formulation to compute an optimal offline policy and derive an online approach based on a model-free Deep Reinforcement Learning (DRL) framework. As a novel feature, the DRL agent action is modeled as a parametric combinatorial problem. A tailored multi-objective reward function leads the agent towards an effective choice of parameters for such a model. Our models are built, trained and fine-tuned by exploiting real data.

Extensive simulations in diverse scenarios show that our DRL online approach produces solutions with small gaps to the optimal offline ones, enabling the operator to both save resources and grant the users an adequate QoS level.

1. Introduction

The profound transformation of the mobile network in the last decade has unleashed the potential to effectively support a new class of mobile services [1, 2]. The fifth generation (5G) of mobile network exploits Software Defined Networking (SDN) [3] and programmable data plane [4], Network Function Virtualization (NFV) [5] technologies and edge computing architectures to bring computational capability at the edge of the network [6, 7]. The technological and architectural frameworks designed by standardization bodies 3GPP and ETSI give mobile operators the opportunity to support latency-sensitive services [8].

While standards provide all the required instruments, functionalities and technologies to manage the life-cycle of edge-based services, the decision process behind the service orchestration is still an open research issue. The edge environment is characterized by a limited amount of available resources, stringent QoS requirements and high spatio-temporal dynamics of the network conditions [9]. These issues are particularly challenging in the case of session-based services that are characterized by stateful and long-lived instances, and involve multiple users. Examples of this type of services are multi-player online games and video conferences, that require low delays, suitable bandwidth, and fair Quality of Experience (QoE) among users.


In such an environment, service orchestration requires an efficient decision mechanism, which operates in online fashion to conciliate multiple conflicting objectives. Moreover, the changing network conditions require the orchestrator

to continuously monitor previously deployed instances and potentially migrate them to enhance QoS or balance the load of the edge servers. To target this issue, many existing works, for instance [6], adopt heuristics and Mixed Integer Linear Programming (MILP) approaches. However, as shown in [10], these solutions are tightly bound to very specific systems, making their performance substantially decrease even in slightly different contexts. Moreover, most of the previous works do not consider the network dynamics over time neither in the problem formulations nor in the proposed algorithm solutions.

In this paper we propose a novel orchestration methodology for a multi-user edge service based on model-free deep reinforcement learning (DRL). DRL combines deep neural networks and reinforcement learning for making decisions in complex scenarios without prior knowledge. The orchestrator we propose has to simultaneously take two decisions: (i) when actually start servicing each user, once a request is issued, and, (ii), which edge facility has to be assigned to such a service. The assignment decisions can change over time by simply migrating services among edge facilities. The orchestrator takes into account the interaction of decisions concerning different users and operates in an *online* fashion, assuming no previous knowledge about future service requests.

The paper contributions are the following.

- We provide a mathematical formulation to compute an optimal offline policy, in terms of a generalized assignment and scheduling problem with elastic capacities (eGAP) with a multi-objective function. It aims to maximize QoS level while minimizing the amount of resources required at the edge.

 christian.quadri@unimi.it (C. Quadri); alberto.ceselli@unimi.it

(A. Ceselli); rossi@di.unimi.it (G.P. Rossi)

ORCID(s): 0000-0002-3608-8142 (C. Quadri); 0000-0002-0983-2706 (A. Ceselli); 0000-0002-4937-7744 (G.P. Rossi)

- We derive a model-free Deep Reinforcement Learning (DRL) framework. It allows to produce orchestrator decisions in a pure online environment, without assuming prior knowledge of regularities in data. As a further novel feature, the DRL agent actions do not directly produce decisions, but rather configure a parametric combinatorial problem, which is finally optimized to produce decisions. A tailored multi-objective reward function leads the agent towards an effective choice of parameters for such a model.
- We show the effectiveness of our approach by modeling a realistic multi-user edge service scenario, performing extensive simulations and hyper-parameters tuning.

The results show the effectiveness of the proposed approach in producing solutions close to the optimal ones. In particular, with respect to an offline (a posteriori) optimal solution, we are able to guarantee a high QoS level, while using only a small amount of extra resources. Finally, our DRL agent is able to adapt to different conditions in terms of system load and service demand patterns.

The remaining of this paper is organized as follows: Section 2 provides background information and related work. Section 3 provides an overview of the use case scenario and introduces the general architecture of the system. Section 4 presents the system model, while Section 5 provides the mathematical formulation of the eGAP. In Section 6 the DRL framework is presented in detail, including the modeling of the observation state, the action implementation, and the design of the reward function. In Section 8 the whole experimental setting is presented. Section 9 reports the training and evaluation results. Section 10 provides an overall discussion about the results, the limitations and strengths of our framework, and the potential improvement points of our work. Finally, Section 11 contains some brief conclusions.

2. Background and related work

Mobile edge computing (MEC) is considered a key element for supporting delay-sensitive services in mobile environment. However, the service management and orchestration at the edge is a challenging task due to the limited amount of resources available and the high dynamics of the edge environment. The authors of [6, 7, 11, 12] survey the literature concerning resource allocation approaches by considering a broad variety of use cases and showing the main challenges for managing edge services.

Edge computing is widely used for offloading the computation to edge servers rather than performing tasks directly on devices [13]. Task offloading is characterized by short-lived executions of multiple independent requests performed by single end devices, e.g. mobile phones, IoT devices, and vehicles [14]. In this work, we focus on different types of services that have the following main characteristics. First, service instances are stateful, meaning that the internal state, e.g., active connections and application-level information,

must be preserved over time and across instance migrations. Second, the life span of a single instance is generally longer than the execution time of an offloaded task, for example, a gaming session can last several minutes or even hours. During this time period, network conditions and system load significantly change [9], and new service requests may arrive. Therefore, resource allocation performed at deploying time could be considerably sub-optimal given the system dynamics and would require a tailored decision process. Finally, each instance hosts a session that groups a small set of geographically distributed users, e.g., a game session of a multiplayer online game. This latter aspect must be carefully considered during the decision process because the overall QoS level of an instance strictly depends on the QoS offered to users involved in that instance. In the following we present the literature contributions related to this work focusing on the different modeling approaches for multi-user service orchestration problem, the proposed solution for the online decision process, and the specific body of literature about eGAP.

The provisioning of the aforementioned type of services through edge computing infrastructures allows the service providers to offer the best QoE as compared to the cloud-based provisioning [15, 16]. Benamer *et al.* [17] present a solution based on genetic algorithms for the game server placement problem in Fog-based architecture. They provide a MILP formulation aimed at minimizing operational costs while guaranteeing an overall QoE for the players. Gao *et al.* [18] propose an edge computing-assisted multiplayer cloud gaming by modeling the decision process as a constrained multi-objective optimization model. In particular, their work focuses on minimizing both the maximal delay difference among players and the operation costs simultaneously. To solve their NP-hard problem, they propose a hybrid approach based on DRL and a heuristic algorithm. Wang *et al.* [19] consider a collaborative edge service placement (CESP) problem jointly taking into account multiple costs (activation, placement, proximity, and co-location). CESP is proven to be a polynomial-time reduction from the uncapacitated facility location (UFL) problem. The authors present an efficient algorithm based on iteratively solving a series of minimum s - t cut problem instances. Tsipis *et al.* [20] present a heuristic for solving the *Social Interactivity-oriented Edge Allocation (SIEA)* problem. SIEA problem is based on the widely studied capacitated facility location (CFL) problems [21]. In particular, the proposed solution explicitly accounts for the communication costs between players and edge-server and among edge servers. The resulting heuristic (SIEA- \mathcal{H}) complexity is $\mathcal{O}(|P|^2|S|^2)$, where P and S are the sets of players and edge-servers, respectively. However, all these works do not consider the temporal dynamic of network load, causing QoS variations, and service demand, i.e., new service requests and/or the termination of previously allocated instances.

In general, dealing with limited resources at the edge is a challenging task and requires tailored allocation and requests

scheduling policies to ensure the suitable QoE, while avoiding system overload. Most of the available literature relies on conventional approaches, as heuristics and MILP optimization. Solutions for joint service placement and requests routing in MEC environment based on randomized rounding [22], approximation algorithms [23, 19, 24] and game theory [25] have been proposed. While the aforementioned solutions work well in the environment they are designed for, they are not able to easily adapt to new contexts without incurring in a degradation of performances. Moreover, most of the previous works only focus on a single metric, e.g., delay [26], resources used [27], Quality of Service/Experience [28, 29], or number of served requests [30]. Clearly, single metric objectives produce high-quality solutions on a single specific target, but do not give the opportunity to the service provider to smooth the optimization, balancing with respect to different competing metrics. Some recent works have indeed considered multi-objective functions optimization [31, 32, 33]. In particular, a recent work by Hazra *et al.* [34] propose a solution for solving task allocation in Industrial Internet of Things (IIoT) scenario. The problem formulation considers a multi-objective function aiming to minimize execution time and energy consumption. Nevertheless, the formulation does not take into account task migrations due to changing network/load conditions. In line with the mentioned recent literature, our goal is to optimize a multi-objective function taking into account competing goals such as QoS, resource usage, service request waiting time, and network operational costs, which is suitable for efficiently managing dynamic scenarios like multi-user edge services.

The recent introduction of Deep Reinforcement Learning (DRL), starting from the seminal work of Mnih *et al.* [35], has given the opportunity to design new approaches for online decision processes. DRL combines the well-established reinforcement learning methods [36] with the advanced techniques of deep learning, providing a powerful framework for managing complex decision tasks. Luong *et al.* [37] provide an overview of DRL approaches applied to communications and networking. In this work, we apply DRL to a collaborative service shared among a group of users as in [18, 19], considering multiple instances of the service which are independent of one another, rather than a single one shared among all users. Schneider *et al.* [10] propose a DRL solution for joint scheduling, scaling and placement of edge services. In [38] a deep Q-network (DQN) approach is used for solving a joint optimization problem of service placement, workload scheduling, and resource allocation to minimize service response delay. Our work solves a similar problem, but we focus on the scheduling and assignment of long-lived service instances, each one serving a specific group of users.

Similarly to [39] we model our decision making process by considering a finite set of actions. However, we employ a more sophisticated algorithm for identifying and implementing the action. It combines heuristics and exact formulations as a combinatorial problem. More in detail, a core optimization problem needs to be solved. From a

modeling point of view, at each point in time, it can be seen as a variant of the elastic Generalized Assignment Problem [40] (eGAP). The eGAP is NP-Hard, but general purpose mathematical optimization solvers such as [41] are known to be effective in its numerical resolution. In our case, one instance of eGAP appears at each point in time, and decisions over time are intertwined. It is the realm of Dynamic Facility Location (DFL) problems. The issue of capacity scaling in DFL has been discussed in the literature more recently [42]. DFL problems show in fact to be considerably more difficult to solve than eGAP ones. Choosing capacity values in a discrete set helps in reducing their modeling and resolution complexity. Still, instances of realistic size can currently be solved only by heuristics like [43].

3. Scenario

We consider a scenario in which a service provider offers a latency sensitive service to a group of users. As discussed, noticeable examples can be online multiplayer gaming and video conferencing. An instance of the service is shared only among a single group of users and remains active for a certain amount of time, e.g., the entire game or conference session. We assume that the group of users is fixed and known in advance. We also assume that the location of each user is known and provided by the base stations that is serving the user. A request for a new service instance can be issued anytime and enters a queue waiting for deployment. To meet the QoS requirements, service instances are deployed on edge facilities that provide suitable, albeit limited, computing resources and lower communication delay as compared to cloud facilities. Periodically, the service orchestrator monitors the load of the facilities and selects a subset of requests for deployment. In this setting, we assume that the service provider can use the computing resources of the edge facilities according to an elastic computing paradigm. In particular, the service provider negotiates a certain level of resources with the edge network operator. If the amount of negotiated resources is not sufficient for handling the total number of requests, the service provider can ask for some extra capacity. In this case, the edge network operator will charge the service provider a cost that is proportional to the extra resource usage.

In this paper, we adopt the very general MEC-based architecture on top of which any latency-sensitive service can be deployed. In Figure 1 we report the general system architecture. The system architecture is composed of three layers. The *physical network* layer (bottom) includes the base stations (BSs), which mobile users are connected to, and the MEC facilities offering virtual computation capacity. The *orchestration layer* (middle) is responsible for managing the network and computing resources, deploying the service instances, and controlling their life-cycle. The *service layer* (top) is in charge of serving upcoming service requests and cooperates with the orchestration layer to deliver the required QoS. In Figure 1 two separate groups of users issue their service requests. Each request includes the type of

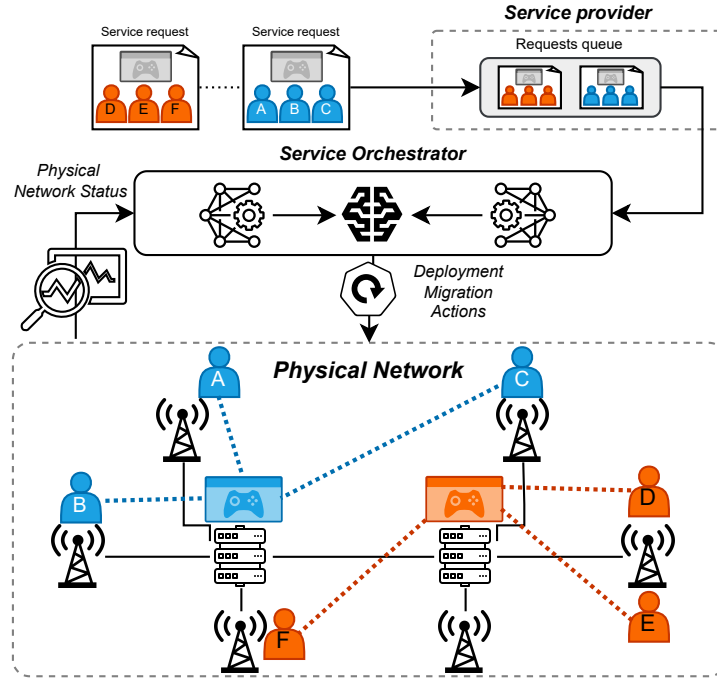


Figure 1: System architecture overview.

service, the resource and QoS requirements, the group membership, and the position of each user within the group. Each request is gathered by the service provider and placed in a waiting queue. At fixed time intervals, the orchestrator gets information about the physical network and queue status, and selects the suitable deployment setting and migration actions. The figure reports an example of the deployment of two instances. Each member of a group interacts with only one instance (dashed colored link) and QoS offered to a user strictly depends on the quality of the communication channel between the user and the MEC facility where the service instance is deployed. The service orchestrator has the responsibility of finding the most suitable deployment strategy aiming at maximizing the offered QoS and minimizing resource utilization. In the following, we present the model of the system, the mathematical formulation of the optimization problem, and the model of the DRL agent for service orchestration.

4. System model

Formally, we assume that the system operates on a discretized time frame, where $\mathcal{T} = \{1, \dots, T\}$ represents the set of all the time slots of duration τ . In real implementations, this assumption does not produce loss of generality, as τ can be chosen to be arbitrarily small.

In the following, we provide some details of the three layers along with the formal notation we will be using throughout the paper.

4.1. Physical network

The physical network consists of a set of base stations B connected to a set of MEC facilities F through backhaul

network [44]. The communication between a BS $b \in B$ and a facility $j \in F$ at time $t \in \mathcal{T}$ has a transmission cost l_{bj}^t , which is proportional to the transmission delay and varies according to the dynamics of the network load. Each facility has a finite capacity that can be used by exploiting the elastic computing paradigm. Each facility j has two levels of capacity, namely R_j and \bar{R}_j . \bar{R}_j is the maximum level of resources available at a facility j , while $R_j (\leq \bar{R}_j)$ represents the amount of computation capacity that can be used without suffering from extra costs. The gap between R_j and \bar{R}_j is the amount of extra capacity available at the facilities whenever needed, e.g. to manage peak of traffic during rush hours. We assume that the use of units of capacity above the R_j level has an extra cost o_j .

4.2. Service layer

The service layer directly supports the end-users to create an instance of the requested service. Throughout the paper, we consider that a single service instance can be shared amongst a group of potentially distributed users. This is a very common setting in many application scenarios, such as online gaming, virtual reality, for instance applied to cultural tourism, and video conferencing. We also assume that the service provider is aware of relevant data about the users' group. In particular, both the identity of users in a group $G_i \in \mathcal{G}$ and their location, i.e. the BS to which they are connected to, are available. As for a service request, we define a generic request of a new service instance i as a 4-tuple $\langle \sigma_i, \delta_i, G_i, r_i \rangle$, where σ_i is the request creation time, δ_i is the service duration time (i.e., the amount of time the service instance remains active), $G_i \in \mathcal{G}$ is the group of users requesting the new instance, and r_i represents the amount

of resources needed for the service instance. We also define $B_i \subseteq B$ to be the set of BSs servicing at least one user in G_i . The QoS level of a service instance i deployed on a facility j depends on the transmission costs, i.e., l_{bj}^t , between the facility and the base stations $b \in B_i$ that serve the group of users in G_i . Most services adjust the QoS level to offer a fair experience to all participants, even to the most critical one [45, 46]. We formally model this issue by defining the cost related to the assignment of an instance i to a facility j as

$$c_{ij}^t = \max_{b \in B_i} h(l_{bj}^t) \quad (1)$$

where $h(\cdot)$ is a generic function that maps the transmission cost onto service QoS level, e.g., frame-per-second in online games or bit-rate level for video streaming applications. Here we are assuming that the service instance adapts the QoS level to the highest transmission cost among the group's users; thus reducing transmission cost leads to an increase in the QoS level for the whole group. In [45, 46] the main goal is to maximize fairness by minimizing the difference in response delay between pairs of players. However, that can be made arbitrarily small by artificially penalizing users with high QoS. Hence, the optimization target of [45, 46] does not provide any guarantee about the overall QoS level, which instead is one of the key performance indicators for the service provider. Defining the QoS level as in Eq. (1) allows us to take into account the overall quality of service offered to the group of users, and consequently considering the global QoS of all service instances as we will discuss in Section 5. We assume that the number of QoS levels is finite.

Service requests do not need to be satisfied immediately; they can be delayed, and the longer the delay, the lower the QoE of the users. We simply define the queue waiting time for a service request i , at time t , as the difference between the current time t and the request time σ_i , i.e. $q_i^t(\sigma_i) = t - \sigma_i$, $t \geq \sigma_i$.

4.3. Orchestration layer

The orchestration layer is responsible for the allocation of physical network resources, the management of the service instances life-cycle, and the monitoring of the allocated resources, as well as the QoS of the running instances. Moreover, this layer is in charge of managing the queue of incoming service requests, denoted as \mathcal{Q} . We assume that such a queue has finite capacity and all requests exceeding the queue capacity are discarded.

Due to the fact that an instance is active for multiple time slots, network conditions could change over time leading to variations in the QoS level. Therefore, the deployment made in the previous time slots may not be suitable in light of the new network conditions. The orchestrator may decide to move a subset of the running service instances from one facility to another to improve QoS or to balance the overall system load, i.e., thus reducing the extra capacity costs. The migration of a service instance brings together extra costs, mainly associated with the rise in control traffic and with the transfer of VM-state between facilities. For

simplicity, we assume that the migration cost m_i depends on the service instance only, without considering the pair of facilities involved in the migration.

4.4. Orchestration mechanism and goal

The orchestrator is in charge of selecting the amount of resources to use at each facility, anytime. Furthermore, when each service instance $\langle \sigma_i, \delta_i, G_i, r_i \rangle$ is issued, the orchestrator decides whether to instantiate it immediately, starting its service in a suitable facility, or delaying its activation. Finally, the orchestrator may decide to migrate running instances from one facility to another. The orchestrator takes and implements its decisions at the end of each time slot, managing the set of service requests issued in the time slot as a batch.

The goal of the service orchestrator is to provide users with the best possible QoS while optimizing the resource consumption at the network layer. Clearly, offering high QoS levels and minimizing resource usage are competing goals and require careful service orchestration and life-cycle management of instances.

5. Mathematical modeling

We formalize the problem of finding optimal orchestration plans using Mixed Integer Linear Programming (MILP). Our MILP combines the structure of two families of problems from the combinatorial optimization literature. First, the main decision problem has an assignment nature: each service instance must be allocated to one facility at minimum cost. The overall load managed by each facility is limited through capacity conditions. In our case these capacities can be extended at a price. Such a structure is known in the literature as the elastic Generalized Assignment Problem (eGAP) [40], where the term 'elastic' refers to the possibility of handling capacities in a flexible way. Second, like in scheduling problems [47], activating all the service instances at their request time might be infeasible or impractical: a proper placement in time must be found for each of them, together with a suitable migration pattern. Formally, our generalized assignment and scheduling problem with elastic capacities includes the following decision variables:

- $x_{ij}^t \in \{0, 1\}$: encodes the assignment of instance i to facility j at time t ;
- $y_i^t \in \{0, 1\}$: encodes the migration of instance i at time t ;
- $s_i^t \in \{0, 1\}$: represents the deployment of instance i at time t ;
- $v_j^t \in \mathbb{N}$: indicates the units of extra resources requested on facility j at time t .

For consistency, we assume each x_{ij}^t, y_i^t, s_i^t fixed to 0 for each $t < \sigma_i$. The problem formulation is the following:

$$\min \sum_{i \in T} \sum_{i \in N} \sum_{j \in F} c_{ij}^t x_{ij}^t + \sum_{i \in T} \sum_{j \in F} o_j v_j^t + \sum_{i \in T} \sum_{i \in N} m_i y_i^t + \sum_{i \in T} \sum_{i \in N} q_i^t s_i^t \quad (2)$$

$$\begin{aligned}
 \text{s.t. } \sum_{i \geq \sigma_i} s_i^t &= 1 & \forall i \in N & \quad (3) \\
 \sum_{j \in F} x_{ij}^t &= \sum_{\bar{i}=t-\delta_i+1}^t s_{\bar{i}}^t & \forall i \in N, \forall t \in \mathcal{T} & \quad (4) \\
 x_{ij}^t - x_{ij}^{t-1} &\leq y_j^t & \forall i \in N, j \in F, t \in \mathcal{T} & \quad (5) \\
 \sum_{i \in N} r_i x_{ij}^t &\leq R_j + v_j^t & \forall j \in F, \forall t \in \mathcal{T} & \quad (6) \\
 0 \leq v_j^t &\leq \bar{R}_j - R_j & \forall j \in F, \forall t \in \mathcal{T} & \quad (7) \\
 x_{ij}^t &\in \{0, 1\} & \forall i \in N, \forall j \in F, \forall t \in \mathcal{T} & \quad (8) \\
 y_j^t, s_i^t &\in \{0, 1\} & \forall i \in N, \forall t \in \mathcal{T} & \quad (9)
 \end{aligned}$$

Our aim is to minimize the multi-objective function (2) by taking into account the assignment costs, the amount of extra resources required, the migrations costs and the queue waiting time. Unlike [40], we do not consider under-provisioning costs. We note that the assignment costs c_{ij}^t in (2) account for the QoS level and, being a minimization problem, the function $h(\cdot)$ (see Eq. (1)) must be carefully defined so the lower the assignment cost the higher the QoS level, and vice versa. In Section 8.2 we provide a definition for function $h(\cdot)$ suitable for our service scenario. The constraint (3) guarantees that each service instance is deployed once and also ensures that the deployment time is higher than or equal to the request time σ_i . The constraint (4) ensures that only one instance is active in the system for the service time interval, i.e. from its deployment and throughout the δ_i time slots up to the termination. The next constraint (5), where we formally always assume terms $x_{ij}^0 = 0$, guarantees the consistency of the assignments and migrations across consecutive time slots. Finally, the constraints (6) and (7) ensure that the capacity of the facilities is not exceeded, even considering the extra capacity requests v_j^t . Model (2)-(9) serves for three purposes: first, in terms of descriptive modeling, to formally state the problem we address; second, to formalize the subproblem to be iteratively solved online; third, when managed by general purpose solvers, to provide a benchmark for the quality of our solutions in the experimental part.

6. DQN modeling

The problem formulation presented in the previous section is not suitable for online optimization; in fact, it requires knowing in advance all the service requests over the planning horizon. This is unrealistic even within very short time frames. To overtake this limitation, we model the orchestration of the service instances as a Markov decision process (MDP) designing a solution based on DRL that uses a simplified version of the optimization model to implement the agent's actions. In Figure 2 we show the overview of the proposed orchestration framework in which the agent observes the state (see Section 6.1) of the environment and selects the next action (see Section 6.2) to perform. As a result of the interaction, the environment changes its state and provides the agent with a reward (see Section 6.4) that encodes the goodness of the performed action. Each

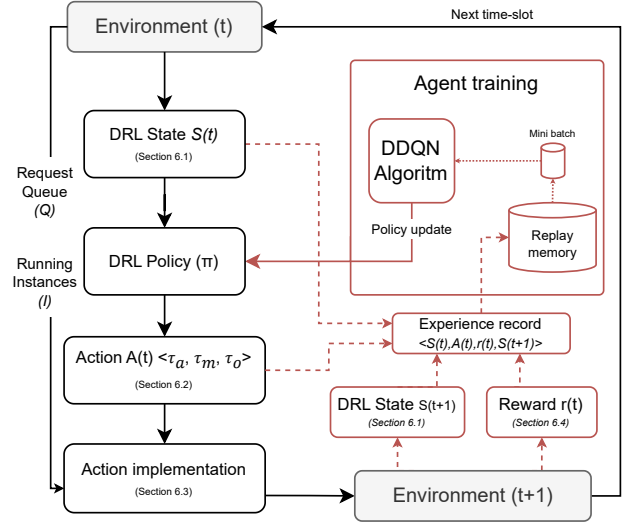


Figure 2: Overview of the DRL orchestration framework. Red arrows and boxes indicate the training process.

interaction with the environment is recorded and stored in a finite replay memory from which mini-batches are sampled to train the agent's policy [35]. Similarly to [39], we carefully model the action implementation (see Section 6.3) which produces the actual deployment plans based on the status of the environment and the selected action.

In the following, we detail the framework presenting the modeling of states and actions space, reward function, and finally we provide details about the proposed algorithm for the orchestration of service instances.

6.1. State model

The following are the observable features of the system:

1. *QoS levels*: the normalized QoS histogram of the running service instances;
2. *MEC facilities occupation*: the value representing the capacity in use in the facility $j \in F$ and normalized w.r.t. R_j ;
3. *requests queue occupation*: the percentage of the queue occupation;
4. *requests waiting time*: mean waiting time of the requests in the queue;
5. *running instances elapsed time*: the mean elapsed time of each service instance currently active in the system;
6. *running instances migration opportunity*: the percentage of running instances that could benefit from migration, for instance, to improve their QoS level.

Such a state formulation allows to model a wide set of scenarios, because it does not strongly depend on specific scenario's parameters, (e.g., the queue size, the facility capacity, the number of requests, and more). The only two relevant parameters are (i) the number of MEC facilities and (ii) the number of distinct QoS levels; they do not compromise the generality of the model. In fact, the deployed number of

facilities per network operator is constant¹ with respect to the time interval of service provisioning; moreover, the QoS discretization can be modeled in order to match a large set of QoS requirements.

6.2. Action model

The goal of the agent at each time step is to choose the best possible action given the state observation. According to the problem formulation described in Section 5, the agent has four types of decision variables: (i) assignment, (ii) migration, (iii) over-provision, and (iv) scheduling. Unlike the offline MILP formulation, which considers all time slots, the DRL agent decides at each time slot and the result affects the next time slot only. As a consequence, a few simplifications can be brought to the problem. The first of them is the elimination of the scheduling decision variables as the DRL agent can only decide whether or not to admit a queued service request. The decision making process can be further simplified by considering batch admission, which enables the agent to specify the number of requests to admit. To this aim the *admission threshold* τ_a is defined whose value represents the percentage of queued requests that are selected to be instantiated.

A similar approach can be followed to decide about migration. The agent sorts all running instances for taking a potential benefit from migration, i.e. reducing the assignment cost. The gain in terms of QoS level deriving from migrating at time t the running instance i , which was assigned to facility j at time $t-1$, can be defined as follows:

$$m_i^{gain} = c_{ij}^{t-1} - \min_j c_{ij}^t \quad (10)$$

Accordingly, a running instance may become a candidate for migration if $m_i^{gain} \geq 0$. Moreover, migration of an instance is granted even when the same QoS level is maintained; in fact, though it does not provide direct benefits to the users, it gives the mobile operator the opportunity to ensure load balancing among facilities. We note that Eq. (10) does not consider the migration cost m_i because it focuses on the potential improvement in QoS only. However, the migration cost is taken into account when the agent performs the selected action and evaluates the costs and the benefits of each potential migration (see Section 6.3). The agent chooses the *migration threshold* τ_m , which is defined as the percentage of instances whose migration gain is non-negative. M is defined as the set of running instances selected for a potential migration.

Finally, to complete the set of decisions to take, the agent decides the amount of extra capacity it requires to manage the currently running instances and those selected for being instantiated. At each time step, the agent chooses the *over-provision threshold* value τ_o , which is defined as the percentage of the available extra capacity, $\overline{R}_j - R_j$.

At each time step, the agent goal is to choose the suitable combination of values τ_a , τ_m , and τ_o , which represent the

action parameters maximizing the long term reward. In order to use a learning algorithm of the DQN family, the number of combinations must be finite, and, as a consequence, discrete values for each action parameter are imposed (see Section 9.1).

6.3. Action implementation

Given the values of action parameters, τ_a , τ_m , and τ_o , the action needs to be executed on the environment. We propose an algorithm that combines a simplified formulation of the problem in Section 5 and heuristics. The set of admitted requests A and the set of instances selected for potential migration M are found heuristically. For selecting service requests to admit, we first sort the queued requests according to waiting time q_i^t in descending order and then we select the first $k_a = \lceil \tau_a |Q| \rceil$ requests. Analogously, for selecting the running instances to migrate, we only consider instances whose migration gain is non-negative, denoted as I^{gain} , and we sort them in descending order according to migration gain value. Then, we select the first $k_m = \lceil \tau_m |I^{gain}| \rceil$ instances.

Instead, in order to assign the admitted requests to the facilities, and decide which requests, amongst the running instances, to migrate, we optimize a generalized assignment problem, whose MILP formulation is the following. We keep the decision variables $x_{ij} \in \{0, 1\}$, which encodes the assignment of instance i to facility j . The remaining decision variables are fixed by the action parameter values chosen by the agent. Therefore, the following become data:

- $\hat{j}(i) \in F$: is the facility where running instance $i \in M$ is currently deployed;
- $c_{ij(i)}$: is the assignment cost, i.e., QoS level, if running instance $i \in M$ was not migrated;
- $V_j = \tau_o(\overline{R}_j - R_j)$: is the amount of extra resources negotiated on facility j ;
- U_j : represents the amount of capacity on facility j which is used by running instances that are not selected for migration.

Our formulation is the following:

$$\min \sum_{i \in A \cup M} \sum_{j \in F} c_{ij} x_{ij} + \sum_{i \in M} m_i \cdot (1 - x_{ij(i)}) \quad (11)$$

$$s.t. \sum_{j \in F} x_{ij} = 1 \quad \forall i \in A \cup M \quad (12)$$

$$\sum_{i \in N} r_i x_{ij} \leq R_j + V_j - U_j \quad \forall j \in F \quad (13)$$

$$\sum_{j \in F} c_{ij} x_{ij} \leq \rho c_{ij(i)} \quad \forall i \in M, \rho \geq 1 \quad (14)$$

$$x_{ij} \in \{0, 1\} \quad \forall i \in A \cup M, \forall j \in F \quad (15)$$

the objective function (11) aims to (i) minimize the assignment costs, i.e., maximize the QoS, (ii) minimize the extra resources demand, and (iii) minimize the migration costs. We note that for each instance $i \in M$ selected for potential migration, the assignment variable $x_{ij(i)}$ indirectly

¹In this work we only consider a single operator, in a multi-operator scenario, the number of MEC facilities could change employing resource brokering systems, which is outside the scope of this study.

encodes the migration. If its value is 1, it means that the instance has not been migrated, on the contrary, if it is set to 0, the instance i has been moved to another facility $j \neq \hat{j}(i)$. Constraint (12) guarantees that all service instances, both admitted and selected for migration, are assigned to exactly one facility. Constraint (13) ensures that the available capacity is not violated, depending on both the resources used by instances that are not selected for migration U_j , and the extra resources being requested V_j . Constraint (14) bounds the assignment cost increasing w.r.t. the current assignment, which is controlled by the parameter ρ . This constraint proves useful to avoid too strong penalization of the running instances selected for migration in favor of new instances. In our experiments, we set $\rho = 1$, forcing the agent to maintain the same QoS level.

The overall action algorithm is summarized in Algorithm 1.

Algorithm 1 Action implementation

Input: Requests queue (Q), Running instances (I), Action ((τ_a, τ_m, τ_o))
Output: Assignment plan $\forall i \in A \cup M$

- 1: $k_a \leftarrow \lceil \tau_a |Q| \rceil$ \triangleright # admitted requests
- 2: Sort queued requests according to q_i^t in descending order
- 3: $A \leftarrow$ get firsts k_a requests in Q
- 4: **for** $i \in I$ **do**
- 5: Compute m_i^{gain}
- 6: **end for**
- 7: $I^{gain} \leftarrow \{i \in I | m_i^{gain} \leq 0\}$
- 8: $k_m \leftarrow \lceil \tau_m |I^{gain}| \rceil$ \triangleright # potential migrations
- 9: Sort I^{gain} according to m_i^{gain} in descending order
- 10: $M \leftarrow$ get firsts k_m instance in I^{gain}
- 11: Solve problem (11)-(15)
- 12: **if** Solution found **then**
- 13: **Return** $\{x_{ij} | x_{ij} = 1\}$
- 14: **else**
- 15: **Return** \emptyset
- 16: **end if**

6.4. Reward function

We design the reward function taking into account three components measured on the arrival state after the action execution: (i) the QoS level offered to the deployed instances, (ii) the amount of extra resources requested and (iii) the waiting time of the requests in the queue. Formally, we define the reward function as follows:

$$r_t = - \frac{\alpha_q \frac{\sum_{i \in N_t} \sum_{j \in F} c_{ij} x_{ij}}{|N_t|} + \alpha_o \tau_o + \alpha_t \frac{\sum_{i \in Q} \tilde{q}_i^t}{|Q|}}{\alpha_q + \alpha_o + \alpha_t} \quad (16)$$

where N_t is the set of currently deployed instances and \tilde{q}_i^t is a normalized and clipped version of waiting time q_i^t defined as:

$$\tilde{q}_i^t = \frac{\min(t - \sigma_i, \delta_i)}{\delta_i}$$

The coefficients α_q , α_o , and α_t adjust the balancing among the three components. The first part of the numerator of (16)

accounts for the average QoS level offered to the deployed service instances, the second part considers the selected over-provision threshold and the last component is the average of the waiting time of the service requests currently in the queue. The denominator normalizes the reward function. According to (16), r_t is bounded between -1 and 0, where value -1 corresponds to the worst case reward, while value 0 encodes action's optimality. In the case Algorithm 1 returns an empty set, we set $r_t = -1$ to provide the agent with strong negative feedback about infeasible actions, e.g. too many instances to deploy or insufficient negotiated capacity.

The reward function defined in Eq. (16) is derived from the multi-objective function of the MILP formulation (see Eq. (2) in Section 5), whose goal is to minimize the overall costs of the service orchestration. In particular, the utopia scenario would be that (i) all instances benefit from the best QoS (i.e., lowest assignment costs) for all duration of the session; (ii) no extra resources are required; (iii) no need of migrations; and (iv) every request is immediately served. The design of Eq. (16) follows the same principle as Eq. (2). As a matter of fact, higher rewards correspond to solutions approaching the utopia scenario more closely. Migration costs are handled implicitly by solving the problem (11)-(15)). In other terms, the proposed reward function guides the agent toward the maximization of the QoS level, the minimization of the used extra capacity, and the minimization of the request waiting time, while the action implementation selects solutions of minimum migration costs. Moreover, through the tuning of the coefficients α_q , α_o , and α_t , the network operator can adjust the importance of the different components to train agents tailored to specific service/system requirements. In Section 9 we analyze the performance of the different configurations of the reward function.

7. Theoretical analysis

One of the key points in our approach is the refined modeling of assignment, scheduling, migration and capacity scaling over time. From a theoretical point of view, such a choice comes at a price.

Observation 1. *The offline problem of finding an optimal orchestration plan is NP-Hard.*

We reduce from the Generalized Assignment Problem (GAP). In the GAP, a set of agents is given, each having a limited computing capability; a set of tasks is also given, each having a computing demand. A cost is associated to each pair task-agent. The GAP consists in assigning each task to an agent, in such a way that the sub of computing demands of tasks associated to the same agent does not exceed its capacity. The sum of assignment costs between each task and the associated agent must be minimized. The reduction works as follows. We consider a single time slot ($|T| = 1$): each task is mapped to exactly one service instance i ; each agent is mapped to exactly one facility j . Task demands are mapped to r_i values; agents capacities are mapped to

R_j values. Extra capacities are set to 0 (i.e. $\bar{R}_j = R_j$). Task-agent assignment costs are mapped to c_{ij} values. A GAP optimal solution could then be found by mapping back service instance assignments to facilities as task assignments to agents. Similarly, we observe the following.

Observation 2. *The optimization problem to be solved in the DQN action implementation is NP-Hard.*

The proof involves a similar reduction from the GAP, and is therefore omitted.

In fact, these two observations justify the use of MILP modeling and resolution methods. In particular, we rely on the following result.

Observation 3. *Our DQN action implementation resolution methods provide global optimality guarantees.*

Such a result comes directly from the use of exact algorithms for generic MILPs in our procedures. From a theoretical point of view, in fact, such a quality guarantee needs to come at the price of computing time guarantees, unless $P = NP$. We additionally report the GAP to be APX-Hard [48]; therefore there are no strong alternatives for obtaining arbitrary quality guarantees. We can finally observe the following.

Observation 4. *In the long term, our DQN method maximizes the reward function (16).*

The optimization of such a reward function would indeed come directly from the use of a Bellman equation on the action-value function, if each action were associated to the admission of a single service request. Our DQN action merges a batch of service requests and, according to Observation 3, solves the corresponding subproblem to proven optimality. The DQN action produced in this way is never worse than a set of DQN actions taken on the single service request admissions independently, since the latter provides a feasible solution for the batch of service requests, but not necessarily an optimal one.

We mention that, in order to solve the DQN action subproblem (11)-(15), exact algorithms can be fine tuned to have polynomial space complexity, but overall lead to exponential worst case-time complexity as explained above, thus defining the overall time and space complexity of our methods. At the same time, general purpose MILP solvers are known to be experimentally very effective [41]. Our tests confirm this behavior, as we could always obtain proven optimal solutions, and no optimization took more than a few seconds. We also report that ad-hoc algorithms for the GAP exist (both exact and heuristics) that are suitable for our approach, and would further reduce the computational effort for executing Algorithm 1 in large systems.

8. Environment setup

The resource orchestration method we just described has been evaluated by means of a simulation environment that models the system described in Section 4 and has been setup on the base of real data (see Appendix B). As an example

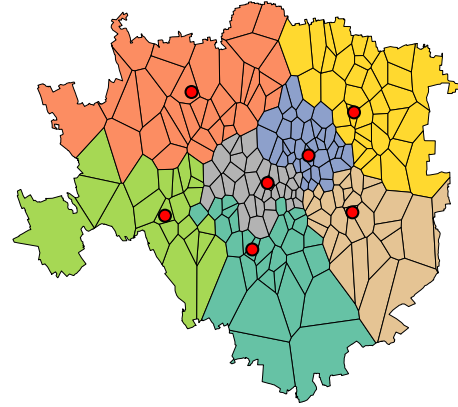


Figure 3: Physical network topology

of latency sensitive application, we consider a multi-player online game. This type of application has stringent latency requirements, while keeps the data traffic very low, in the order of 10-50 Kbps [49]. In the following, we provide an overview of how the system parameters and the players behavior have been defined.

8.1. Physical network

From a Call Detail Records (CDRs) dataset, which gathers the phone activities of about one million subscribers [50] and covers the metropolitan area of Milan, we collect a set of 224 base stations within the metropolitan area of Milan, together with their approximate GPS location, and we extract the approximate location of subscribers with a granularity of associated base station. The backhaul network and the number and location of the facilities within the physical network can only be hypothesized. To this aim, we assume that the backhaul is organized in hierarchical rings, as in [44]. In particular, each base station is connected to a M1 node and each M1 node can handle up to 6 base stations. M1 nodes are organized in access-rings which contain up to 6 M1 nodes each and are connected to the rest of the backhaul network and to the core network via a M2 node which handles up to 4 access-rings. We assume that the network operator has deployed one MEC facility in each access-ring; thus 7 MEC facilities are required to manage 224 base stations. To properly reconstruct the backhaul topology we apply a modified version of the k-medoids algorithm, where we impose a maximum number of points in a cluster. We first associate base stations to M1 nodes and subsequently we create the access-rings. The resulting mediods represent the locations of the MEC facilities. Figure 3 shows the resulting physical network topology, where each area represent a base station obtained from the Voronoi tessellation, the red dots are the facilities, and the color indicates the access ring the a base station belongs to.

To model the communication costs between base stations and facilities, we assume that the variation of delays is caused by background traffic of other services using the same

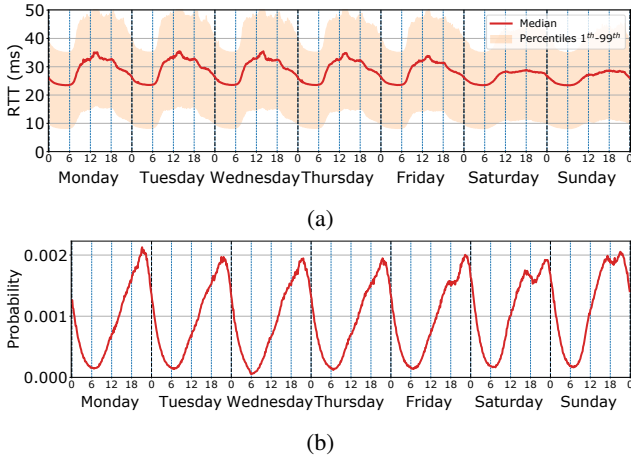


Figure 4: RTT over week (a). Service requests generation probability over a week (b)

physical network. To model the background traffic demand from each base station to facilities we combine the CDRs with another dataset, provided by the Italian operator Telecom Italia Mobile (TIM), as part of their Big Data Challenge initiative [51]. We assume that (i) all the traffic generated from a base station is managed by the closest facility, (ii) the experienced delay is a function of the amount of the traffic at both base station and facility. Applying STL time-series decomposition [52] and the results of time-shared systems analysis in [53], we create an instance of RTT distribution over a week as the one shown in Figure 4a. For more details about background traffic reconstruction, see A.

8.2. Service

To model the user base we reconstruct the interactions graph of subscribers considering call and text messages activities from CDRs dataset. For each subscriber in the graph we identify his/her home base station by applying the home/work detection algorithm in [54] on the mobility trace reconstructed from the subscriber’s phone activities. We filter out all subscribers whose home base station is not detected, and based on the filtered graph we extract all maximal cliques of 4 subscribers, which constitute the set of groups of users \mathcal{G} .

As for the temporal dynamics of the service requests, we consider the popular Multiplayer Online Battle Arena (MOBA) League of Legends (LoL). Through the public APIs service provided by the Riot Games developers [55], we extracted the starting time of all the matches played in 2019, and we aggregate them over 10-minute time intervals. Likewise our approach for the background traffic, we apply STL time-series decomposition to extract the weekly service request pattern and we normalize the resulting time-series to get the probability of generating a service request in a specific time interval. We assume that all service requests are drawn from the same distribution. In Figure 4b we report the probability of generating a service request over a week time period.

Finally, we model the cost related to the assignment of a facility instance, i.e. c_{ij}^t in Eq. (1), by considering that the QoS level affects the users’ Quality of Experience (QoE). We assume 5 levels of QoS, from 1 to 5, while level 0 encodes the “no-service” case as a result of observing excessive network delays. Therefore, we specify function $h()$ of (1), mapping transmission costs to service QoS level, as follows:

$$h(l_{bj}^t) = \frac{\min\left(5, \left\lfloor \frac{l_{bj}^t}{\Delta} \right\rfloor\right)}{5} \quad (17)$$

The term Δ is a simple normalization factor that is tuned according to the range of variation of the network delay. In our settings we use $\Delta = 10ms$ which allows us to cover the entire range of network delays (see Figure 4a). With this formulation the lower the RTT the lower the assignment costs, for example, an RTT below 10 ms leads to $c_{ij}^t = 0$, while an RTT above 50 ms is mapped to 1, i.e., the highest assignment cost. For sake of completeness, in this work we use a straightforward function to map network delay to QoS, but (17) can be easily modified to suit more sophisticated mappings.

8.3. Agent

In order to use a DQN learning algorithm, the set of possible actions must be finite. We discretize the set of values of the action parameters, i.e. τ_a , τ_m , and τ_o , by considering five possible values (see Table 1). Among all possible combinations, we remove those with both τ_a , and τ_m equal to 0, thus leading to 120 possible actions.

As for the neural network architecture, we use a multilayer perceptron (MLP) having input layer composed of 17 units (the dimension of the state representation), output layer of 120 units (all possible actions). We experimented different MLP structures by varying the number of hidden layers (2,3,4) and the number of units within each hidden layer (16, 32 and 64), but we observed similar results for all configurations. Based on these results we use 2 hidden layers of 64 units each.

In Table 1 we summarize the main parameters of our simulation environment. We remark that our computational analysis is focused on evaluating the feasibility and the effectiveness of the proposed DRL orchestration approach; therefore the simulation framework we used in our experiments provides a high-level abstraction of the whole system, without simulating in full engineering detail real world protocols and network interactions.

9. Results

In this section we present the detailed results of our experiments.

9.1. Learning phase

We implemented our system using the *OpenAi Gym* framework [56] and we use *Tianshou* library [57] for training the agent. As a training algorithm, we use a variant

Table 1
Simulation environment parameters

<i>Physical network and service parameters</i>	
Number of BSs	224
Number of facilities ($ F $)	7
Number of groups ($ G $)	~ 2200
Service duration time (δ_i)	6
Service requests per week (day)	~ 6000 (~ 860)
Queue capacity ($\max\{Q\}$)	100
Service deployment cost (r_i)	1
Service instance migration cost (m_i)	1
Over-provisioning cost (o_i)	1
Base facility capacity (R_j)	8
Maximum facility capacity (R_j)	12
<i>Agent actions</i>	
Admission threshold (τ_a)	0%, 25%, 50%, 75%, 100%
Migration threshold (τ_m)	0%, 25%, 50%, 75%, 100%
Over-provisioning (τ_o)	0%, 25%, 50%, 75%, 100%

of the DQN algorithm called Double-DQN[58] with Prioritized Experience Replay (PER) memory [59]. DDQN helps to mitigate the over-estimation problem that affects DQN, while PER allows for better sampling of the replay memory. In particular, with PER memory each element is sampled with a probability that depends on the temporal difference (TD) error, rather than drawn from a uniform distribution. The key idea behind PER is to give more importance to those samples where the difference between the expected reward and the received one is large. In other words, the goal is to train the agent with more examples of bad past decisions. Moreover, PER introduces a bias correction weight, namely *importance-sampling*, which is tuned using the hyper-parameter $\beta \in [0, 1]$. β is linearly annealed from its initial value to 1 during the learning phase, allowing an almost uniform sampling at the beginning of training, where the policy is highly unstable, in contrast to a more biased selection of examples in the final training stages. More details about PER are available in [59]. To balance exploration and exploitation during the learning phase, we adopt a simple ϵ -greedy policy, imposing $\epsilon = 0.1$. In Table 2 we report the main hyper-parameters used in the learning phase and in the following we discuss which combination led to best results in terms of mean expected reward.

In Figure 5 we show the results of the learning phase using different combinations of hyper-parameters. From Figure 5a we can observe the effect of two hyper-parameters: the discount factor γ and the learning rate. The results clearly show the impact of the discount factor. In particular, if $\gamma = 0.99$ the learning algorithm convergence is poor, probably because the current Q-function estimation discount is too optimistic. As for the learning rate, the tested values do not affect the performance significantly. We therefore fix the learning rate to 0.0005 in the subsequent experiments. The impact of the mini-batch size is shown in Figure 5b. The learning phase benefits from larger mini-batches which allow to obtain a finer update and consequently faster convergence. For these reasons, we choose random mini-batches of 64 samples.

Table 2
Parameters for the learning state, in bold the configurations leading to better results.

<i>DDQN parameters</i>	
Episode duration (steps)	1008 (1 week)
Learning phase duration (episodes)	1040 (20 years) - 1560 (30 years)
Target network update interval	2000
Learning rate	{0.0001, 0.0005 , 0.001}
Discount factor (γ in)	{0.99, 0.9 }
Training interval steps	4
Mini-batch size	{32, 64 }
Exploration rate (ϵ)	0.1
<i>Replay memory</i>	
Memory size	100,000 (FIFO)
Prioritization exponent (α in [59])	0.6
Important sampling weight exponent (β in [59])	0.2
<i>Reward function parameters</i>	
QoS weight (α_q)	{0.25, 0.5, 1}
Over-provisioning weight (α_o)	
Waiting time weight (α_t)	
<i>MLP architecture</i>	
Hidden layer units	64
Number of layers	2
Activation function	Hidden layers: Relu Output layer: Linear

The multi-objective formulation of the reward function (see Eq. (16)) provides the service provider with the opportunity to finely balance reward components. We perform a grid-search considering three values for each weight α_q , α_o , and α_t , i.e. 0.25, 0.5 and 1. The Figure 5c shows the evolution of the mean reward per episode over a learning phase of more than 1000 episodes on a representative subset of combinations. As we can observe, the choice of reward components' weights has a strong impact on the maximum accumulated reward. In particular, the weight associated to the waiting time component α_t is a critical factor, whose reduction leads the agent to converge to low reward values. As for the other two components, the results show a marginal sensitiveness to the QoS weight α_q and poor performance when the value of the over-provisioning weight α_o is greater than or equal to α_t . We further investigate the impact of the weights of the reward function. In Figure 6 we report the evolution of each single reward component considering the four combinations of weights shown in Figure 5c. These figures confirm what we observed in the combined reward function, that is the strong impact of the value of α_t (see Figure 6a), in particular, the higher relative importance of QoS components leads to converging to lower values of the reward component. These results are a direct consequence of the action implementation described in Section 6.3. The actual QoS level of the deployed instances is the result of the solution of the eGAP problem (11)-(15), on which the agent has control only indirectly, through the choice of migration threshold (τ_m) value. Moreover, the achievable QoS level mainly depends on the conditions of the underlying physical network, which are not under the agent's control. On the contrary, the agent has direct control of both the queue and the negotiated resources; thus the choice of the values of τ_a and τ_o directly impacts the reward function. As we can observe in Figure 6b and Figure 6c, the greater the relative importance of the component, the higher the reward component value. This is particularly evident for the waiting

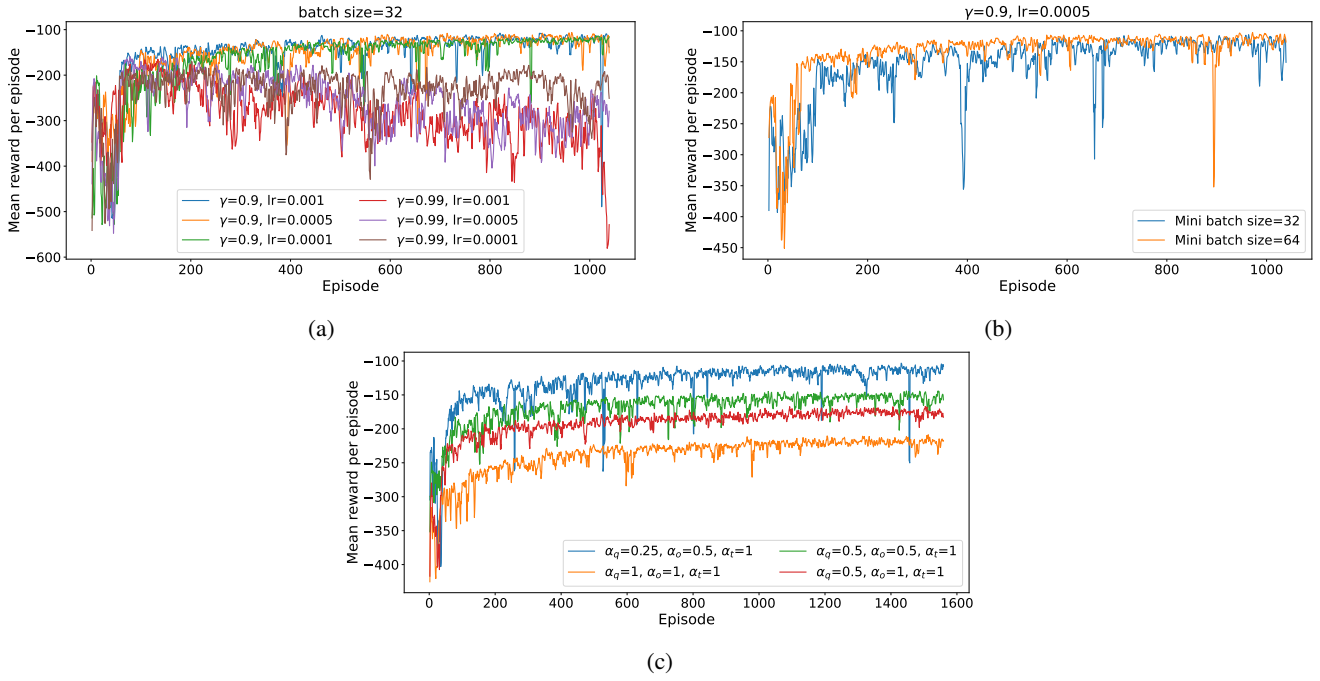


Figure 5: Evolution of mean reward per episode during the learning phase varying hyper-parameters settings.

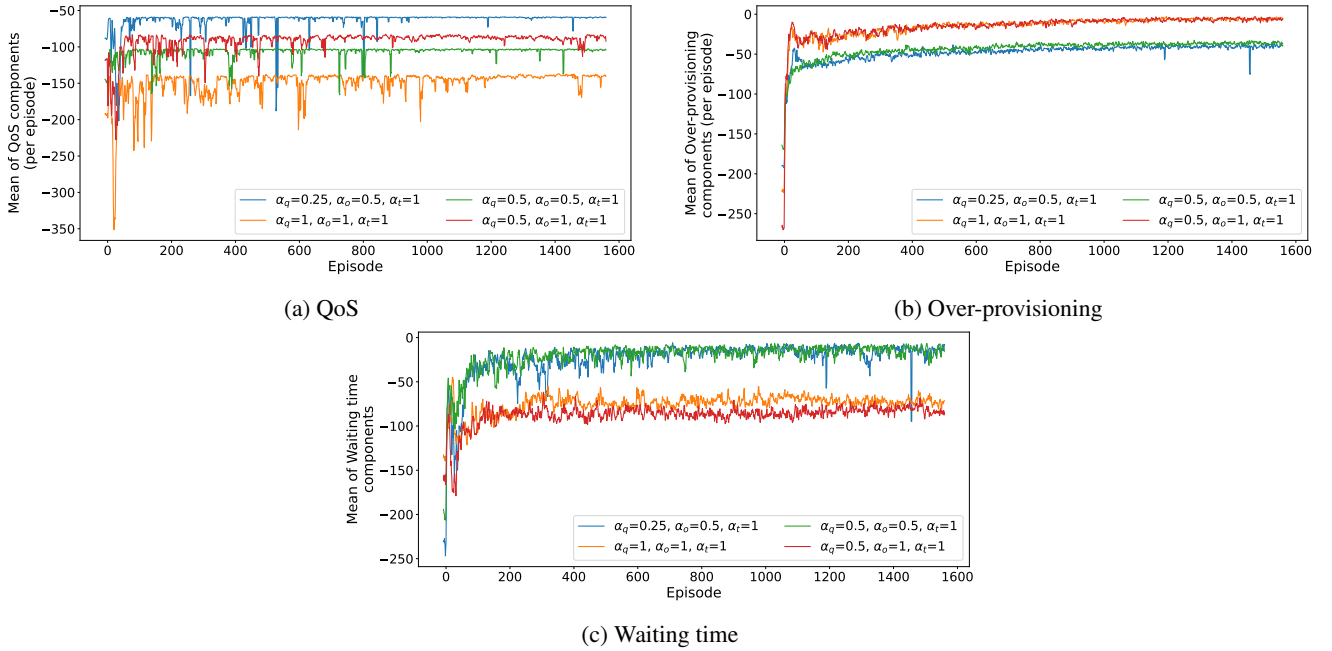


Figure 6: Evolution of mean reward per episode during the learning phase of the different components of the reward function considering different weights combinations.

time component, with a difference of almost 100 units of reward collected during a training epoch.

Overall, the results of the training phase show a fast convergence of the reward function followed by a long fine-tuning phase of the DRL policy. Moreover, we underline the sensitivity of the policy to the hyper-parameter values, which requires careful tuning according to the operation goals of the service orchestrator.

9.2. Policy evaluation

In this section the learning policy is evaluated by comparing the decisions of the DRL agent with those obtained by means of the full offline problem model (see Section 5). From the model (2)-(9) we obtain a lower bound guarantee on the best value that the DRL agent can achieve. In fact, while the DRL agent works online, (2)-(9) works offline,

requires all data in advance and generates a globally optimal solution.

Due to the long computational time required to solve the problem (2)-(9) over a single epoch (1 week), the optimization proceeds by disaggregating it into one independent subproblem working over a single day (144 time slots). In the remainder we refer to such a solution as the *optimal offline policy*.

As a benchmark of our approach, we compare it with three different heuristics. The first is a simple greedy best-fit scheduling heuristics [60] that takes the service requests one at a time, in the arrival order, and selects for each of them the facility to assign and the start time minimizing the objective (2). The orchestrator is then committed to start processing the request accordingly, without interruptions and without migrations. The partial solutions produced for the single requests iteratively stack up in a full solution. The time horizon of best-fit has been limited to 6 time slots²: its solution can be produced very quickly by simple inspection loops.

The other two heuristics are “online” counterparts of the model (2)–(9). They consider only the service requests in a single batch, and only the time slots in the horizon of 6 time slots, thereby producing an instance whose size is very small. Their global optimal solution is obtained by means of GUROBI. From a combinatorial viewpoint, this is the best solution that can be produced without learning mechanisms. As for best-fit, the orchestrator is committed to performing this partial solution, and partial solutions for the single batches iteratively stack up into full solutions. Each GUROBI run requires about 0.05 s (+/- 0.3 s). Two versions of this MIP online algorithm have been included in our comparison: one setting all objective function costs to those of the time slot in which the algorithm is invoked, and one setting the costs to a forecast value coming from data. We refer to the latter as the ‘cost prediction’ version.

We run experiments on 100 different sets of randomly generated service requests (see 4b) and we evaluate the performance by means of the following metrics:

1. *Optimality gap* - it measures the goodness of the agent decision, comparing the value of the objective function (2) on its solution w.r.t the one of the optimal offline policy. The four objectives are aggregated by sum, using equal weights. The optimality gap metric is defined as $|obj^a - obj^*|/|obj^*|$ where obj^a and obj^* are the values of the objective function evaluated considering the DRL agent and optimal offline policy, respectively;
2. *Average QoS level* - it accounts for the average level of QoS offered to users over the entire running time of a service instance;
3. *Average waiting time* - it measures the average time a service request has to wait in the queue before being chosen for deploying;

4. *Over-provisioning overall cost* - it accounts for the total amount of extra resources required over the whole time period;
5. *Number of migrations* - it counts the number of migrations performed over the entire time period.

In the following, we show the results obtained by comparing the DRL policies against the three above mentioned competing policies as well as the optimal offline policy. For sake of readability, we only report the results of the top-three DRL policies based on the median optimality gap value.

In Figure 7 we report the boxplot of the distribution for each metric over all 100 experiments. We can observe that the DRL policies achieve an optimality gap always below 5% with a median value around 2.5%. Conversely, the other non-DRL policies achieve only an optimality gap of around 12-14%, on median. Considering the four components of the objective function (2) separately, we observe significant differences among policies. By looking at the QoS component we note that this component is the least sensitive to different policies. We note that all DRL policies are able to guarantee near optimal QoS level, while the cost prediction version of the MIP online policy achieves the best result benefiting from the information about future assignment costs. On the contrary, waiting time and over-provisioning components are highly sensitive to different policies. We can observe that DRL policies are the most balanced by performing a good trade-off between the two components. In particular, they are able to guarantee a waiting time close to the optimal one without asking for too many extra resources. Both MIP online policies show extremely conservative behavior, asking for a limited amount of extra resources by strongly penalizing the waiting time. The best-fit heuristic performs similarly to DRL policies by looking at over-provisioning component, but is unable to offer short waiting time. As for the migration component, DRL agents perform a number of migrations in line with the optimal offline policy, as opposite to MIP online policies which perform up to three times the optimal number of migrations. The best-fit policy does not perform any migration by design.

In Figure 8 we consider two other metrics which are not directly part of the objective function (2), namely the mean of the load of the facilities and the largest load difference over time. In the figure, we report all six online policies and the optimal offline one. For each policy, the top figure shows the facilities load over time, while the bottom one reports the largest load difference metric. For sake of readability, we only show the mean value and the standard deviation of the mean across 100 runs. As observed in Figure 7, both MIP online policies tend to use slightly fewer extra resources, in particular during the peak load (between 18 and 24). In general, the best-fit policy shows the highest over-provisioning cost (see Figure 7), however, it fails to use all the resources available at the facility during peak hours. This is more evident by observing the difference in facility load, where it emerges that best-fit deployments are more unbalanced w.r.t. other policies during mid-load hours and saturate in the peak hours. DRL policies exhibit a behavior

²We observed that in some challenging scenarios 6 time slots were not enough. In these cases we allowed a time horizon of 12 time slots.

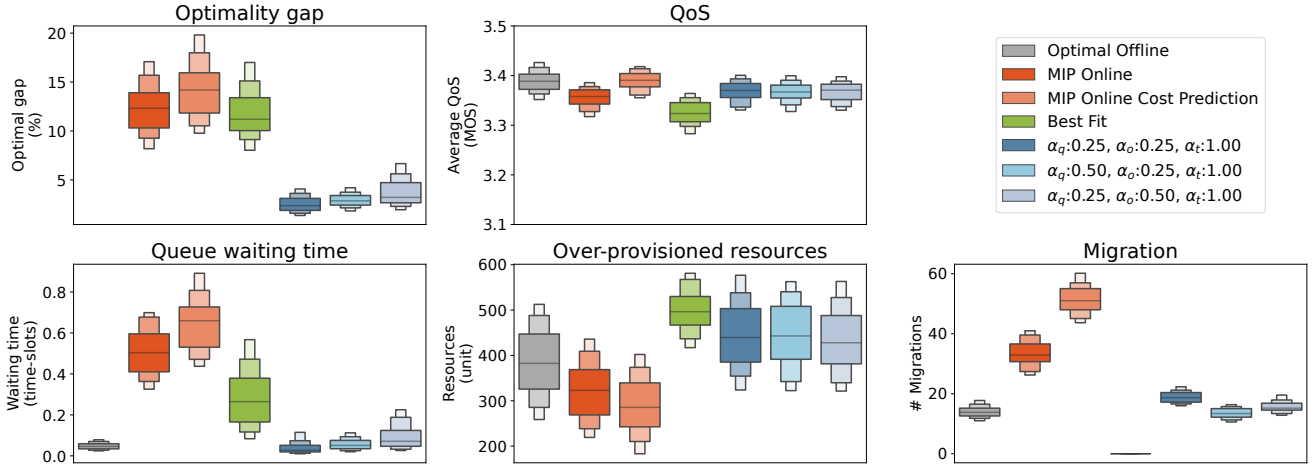


Figure 7: Evaluation on base scenario: boxplot of the distribution of the evaluation metrics considering the top-three DRL policies, the optimal offline policy and the heuristics.

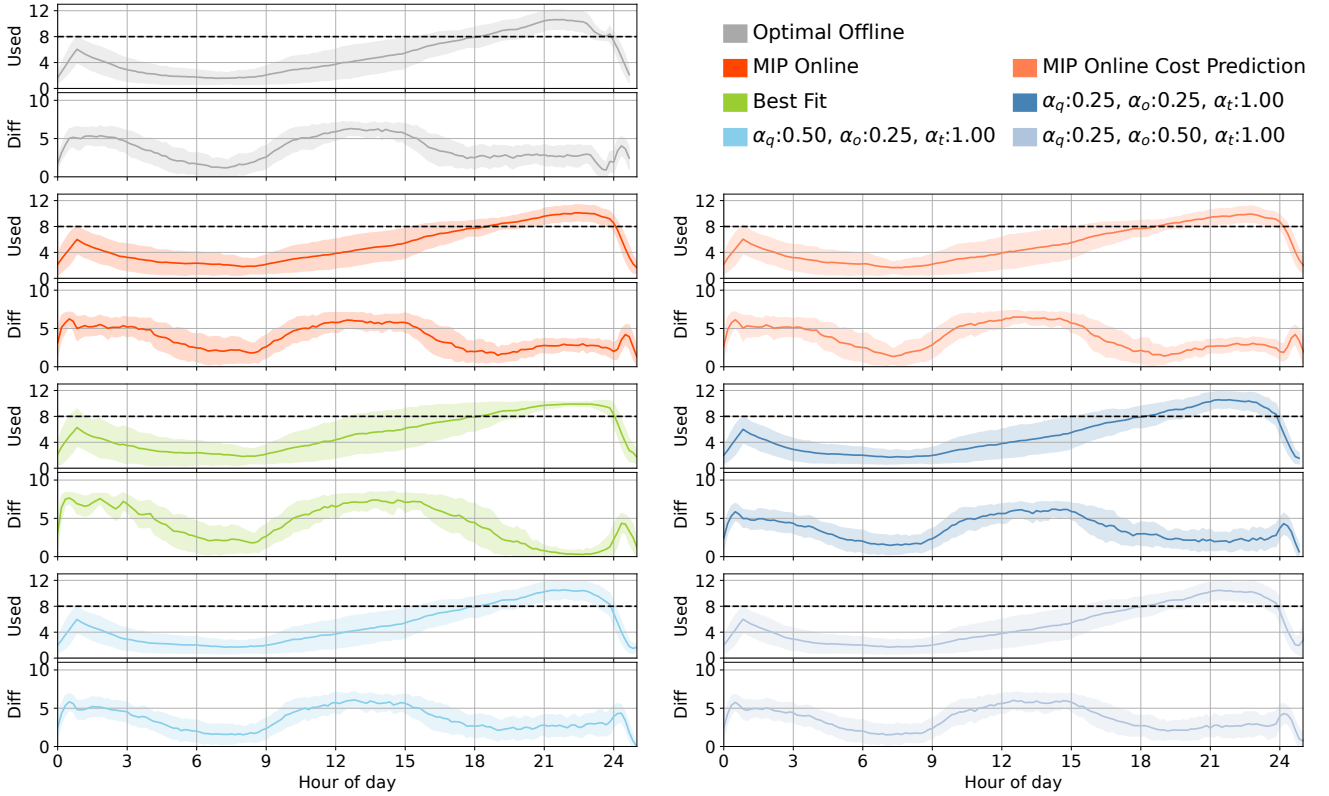


Figure 8: For each policy, the mean of the load of the facilities (top) and the largest load difference (bottom) over time in the base scenario. The horizontal dashed line indicates the limit of the available capacity without incurring extra costs (R_i).

close to Optimal policy, with the exception of slightly more usage of extra resources during peak hours, in line with the results shown in Figure 7 highlighting the overall higher over-provisioning cost.

The results observed in this set of experiments highlight the ability of the DRL agents to orchestrate the service by balancing the different components of the objective function. Moreover, we can observe that the reward function configurations for the selected DRL policies are all unbalanced

towards the waiting time component, in line with what we have observed in the learning phase.

9.3. Evaluation on higher system load

In a second set of experiments, we test the performances of the DRL policies when used on systems having a load that is different from that used for learning. In particular, in our tests, the overall number of service requests is 10% higher than the one used during learning and previous evaluation

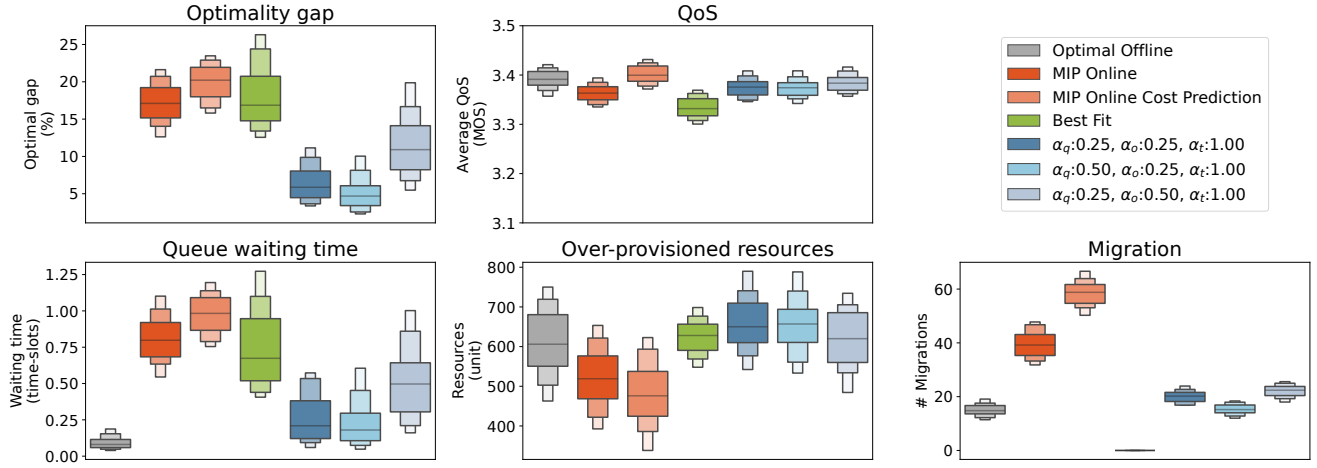


Figure 9: Evaluation on higher system load: Boxplot of the distribution of the evaluation metrics considering the top-three DRL policies, the optimal offline policy and the heuristics.

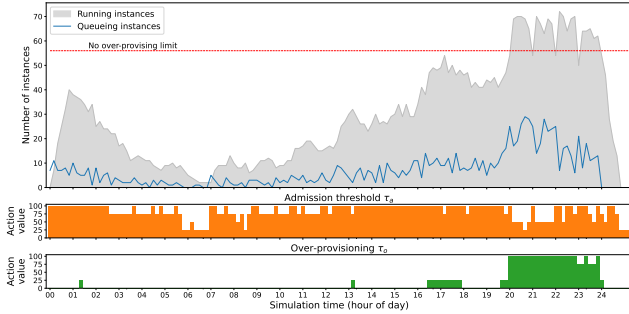


Figure 10: Example of DRL agent decisions in a more loaded scenario.

phase. For consistency, we consider the same set of DRL policies used above (Section 9.2). Figure 9 shows the boxplot of the distribution for each metric over all 100 experiments. As expected, the performance worsens, due to the more challenging environment. Also in this scenario, DRL policies outperform the competing non-DRL policies and exhibit a higher capability to properly balance all the components. Both MIP online policies show the same behavior as the one observed previously in Figure 7, i.e., a conservative usage of capacity at the expense of a long waiting time. The best-fit heuristic has a similar behavior to DRL and optimal policies in terms of extra resource usage, but yields the lowest QoS level and has poor waiting time.

By focusing on the differences of the performance of the DRL agent w.r.t. previous scenario, we observe that the agent performs worse than the optimal policy by looking at the queue waiting time component. This is caused by the myopic view of DRL agent which eagerly admits service requests to reduce waiting time which leads to rapid saturation of the facilities thus preventing the agent from deploying all requests in the queue, in particular during peak hours. In Figure 10 we report an example to clarify this observation. As we can see, starting from 7 P.M., the system is getting more and more loaded due to the DRL agent decision to

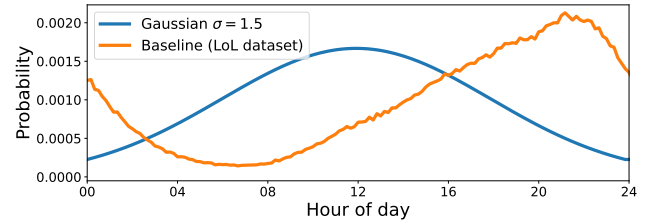


Figure 11: Probability of generating a service request during the day. Baseline (from LoL dataset) and gaussian distribution.

admit almost all requests. At a certain point (around 8 P.M.) this action becomes infeasible even by using all the extra capacity, thus inducing the agent to reduce the admission threshold. As a consequence, a non-negligible amount of requests have to wait for long in the queue. As for the load of the facilities over time, we observed similar results as in the previous scenario (see Figure 8).

Overall, the results obtained with this set of experiments highlight the goodness of the DRL policies which are able to compete with the optimal offline policy in terms of both achieved QoS and required resources. They outperform other online policies at the cost of a marginal increase in queuing time.

9.4. Evaluation on different temporal dynamics

In a last set of experiments, we consider a further challenging environment, in which the DRL agent's is asked to take decisions on a system having a different temporal pattern of service requests (see Figure 11). This task is more difficult, due to a longer period of high service demand causing faster saturation of the facilities.

In Figure 12, we show an example of the DRL agent decisions in this case. As we can observe, the number of requests in the queue during the peak hours is almost double the one in Figure 10, although the agent asks for the maximum available capacity. In Figure 13 we summarize the results of these experiments which highlight the strong contribution of

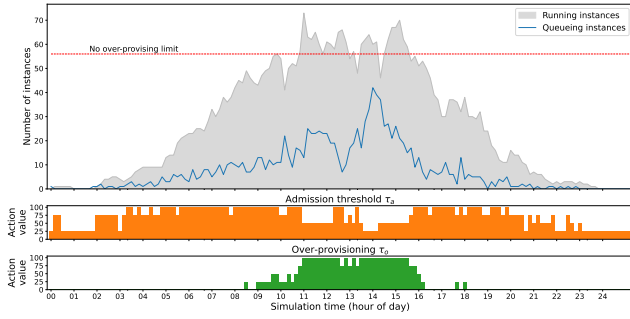


Figure 12: Example of DRL agent decisions in the gaussian requests generation scenario.

the waiting time component and the differences among the various reward function settings. In particular, the third DRL policy, i.e. ($\alpha_q = 0.25, \alpha_o = 0.5, \alpha_t = 1$), which tends to be more conservative in asking for extra resources, performs significantly worse than the others. On the contrary, the other two DRL policies achieve again resource usage levels that can compete with the optimal offline ones and perform better than non-DRL competitors. Besides, DRL policies also preserve an overall good performance regarding the waiting time, which keeps around one time slot on average. Non-DRL policies exhibit the same behavior observed in the other two scenarios, with the only exception of the best-fit heuristic which becomes more conservative in asking for extra resources.

As for the resource usage at the facility level, in Figure 14 we report the load of each facility and the largest load difference over time. Compared to the base scenario (see Figure 8) we observe that all the online policies use more extra resources for a longer time than the optimal one during the peak hours. The only exception is the last DRL policy, i.e., ($\alpha_q = 0.25, \alpha_o = 0.5, \alpha_t = 1$), which, however, exhibits poor performance admitting less requests than the other DRL policies, causing longer waiting time (see Figure 13). As highlighted in the analysis of the base scenario (see Figure 8), the best-fit policy is unable to exploit all the available capacity and suffers from poor load balancing among facilities. Another interesting general observation about the behavior of online policies is that they produce more unbalanced allocation strategies than optimal offline in peak hours. However, this drawback is partially compensated by the fact that facilities are not completely full, on average, as with the optimal policy, leaving some free resources to cope with some extra service requests.

9.5. Computational efficiency

The last aspect we analyze is the computational efficiency of the online decision process. To perform this analysis, we measure the time the agent requires to perform a decision over a single time slot. In order to keep track of the policies requiring to solve an optimization problem, we also include the preprocessing time to generate a problem instance. All the measurements were conducted on a Ubuntu 20.04 server equipped with two Intel Xeon Silver 4216

(2.10 GHz) capable of managing 32 parallel threads each. In the experiment, we execute 60 scenario runs in parallel, simulating the case of multiple orchestration tasks on a single physical machine.

In Table 3, we report the descriptive statistics (i.e., mean, standard deviation of the mean, and median) of the time taken by each online policy in three scenarios presented above to perform the decision process. As expected the best-fit policy is the quickest taking 2-3 ms, on average, independent of the type of scenario. On the contrary, MIP online policies are the least computationally efficient (~ 150 -270 ms on average), and processing time is strongly affected by the difficulty of the problem instance. DRL policies show a good performance with processing time between 14 and 17 ms, which is a remarkable result if we consider their complexity as compared to the best-fit heuristic. Finally, in Figure 15 we report the complementary cumulative distribution function (CCDF) of the decision time considering all three scenarios. This analysis gives an indication of the suitable duration of a time slot to be able to use a specific online policy³. From the figures, we can see that best-fit can be easily employed for near-real-time or even real-time decision processes below 20-30 ms. On the contrary, MIP online policies can not be used in near-real-time settings, because a non-negligible fraction of decisions takes more than 500 ms or even 1 s in case of more challenging scenarios (see Figure 15c). DRL policies are able to perform decisions within 100 ms in most cases making them suitable for near-real-time decision processes.

10. Discussion

The proposed online approach based on DRL has shown good performance in all the evaluation scenarios. In general, our DRL framework is able to match the QoS of an optimal offline policy while keeping low the waiting times, at the only cost of using a marginal amount of extra resources. Such a result is far from obvious, as our DRL agent has no information about potential future requests, which could lead to sub-optimal decisions especially when facilities are overloaded. Therefore, we expect that further improvements can be achieved by estimating future service demand, thus assisting the agent to take better decisions. To this aim, we could improve our framework by embedding AI-based prediction models, such as [61], into the DRL agent.

A further key issue to consider is the tuning of the reward function which depends on both the model hyper-parameters and the weights balancing the different objectives. While the first set can be chosen through simulations, the best set of objective weights to select in a real deployment mainly depends on the specific characteristics of the service, and on the relative importance among the different components of the objective function. Ultimately, such a relative importance needs to be chosen by the decision maker. However, we have shown that some combinations of weights can be dominated

³Here we assume that the order of magnitude of service requests per time slot is the same as the one used in our experiments.

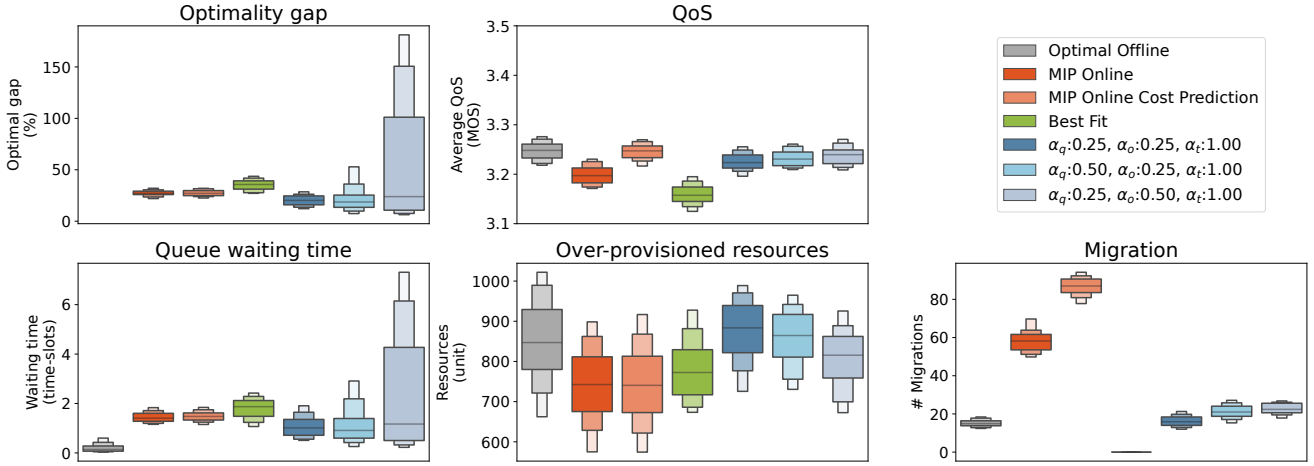


Figure 13: Evaluation on different temporal dynamics: Boxplot of the distribution of the evaluation metrics considering the top-two DRL policies, the optimal offline policy and the heuristics.

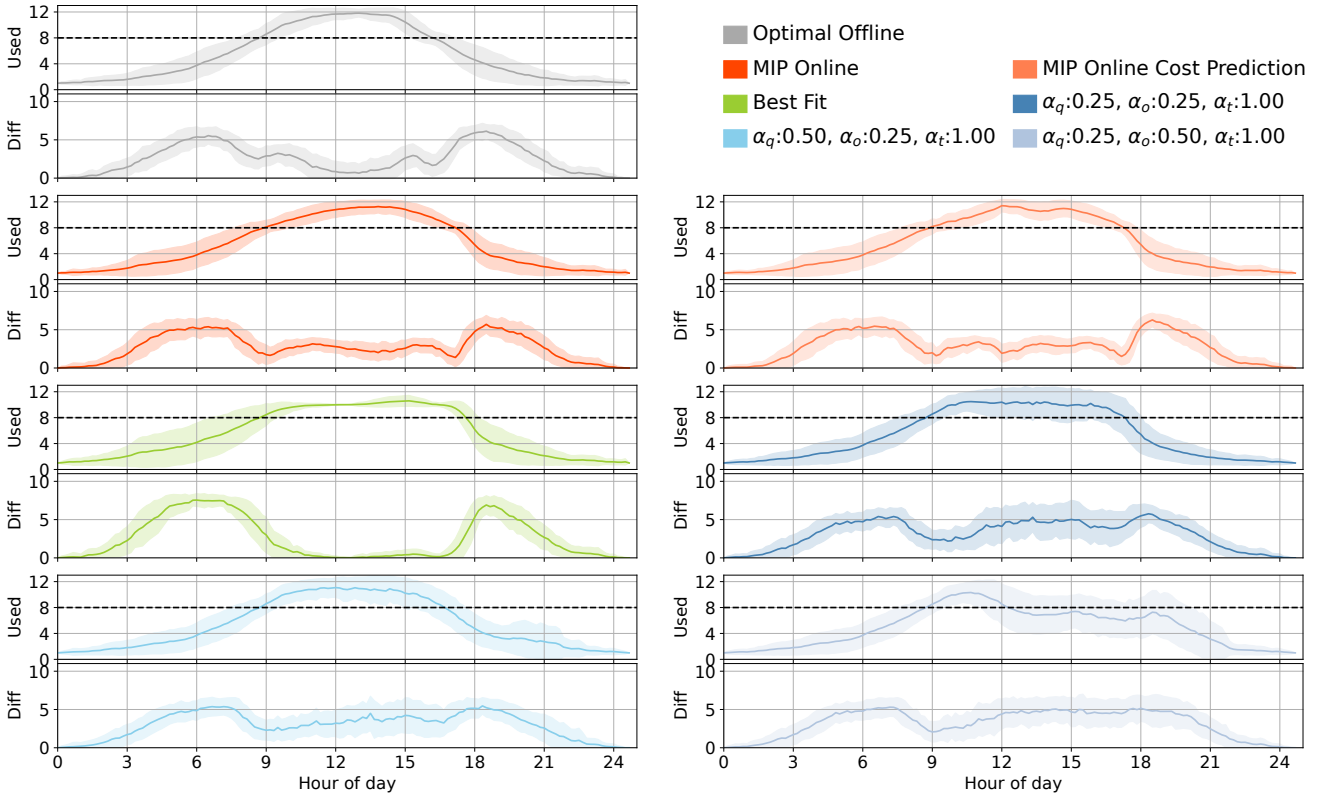


Figure 14: For each policy, the mean of the load of the facilities (top) and the largest load difference (bottom) over time in the higher system load scenario. The horizontal dashed line indicates the limit of the available capacity without incurring extra costs (R_j).

by others by means of experiments, thus guiding the decision maker towards smart weight choices. In particular, a fair balance among the different components looks suitable for a generic multi-user service tolerating a non-negligible setup time. Moreover, the results obtained by using the DRL policies are consistent across all the tested environments, thus showing the ability of the agent to adapt to similar, albeit new contexts.

We remark that we do not provide a monolithic solution for the online orchestration of edge services. Rather, we propose a flexible and modular integration of DRL and MILP components. Each component can be modified to address the specific requirements of a service, thus allowing a finer tuning. Moreover, specific deployment goals or efficiency requirements may affect the choice of a specific DRL algorithm and even the actual action implementation. As an

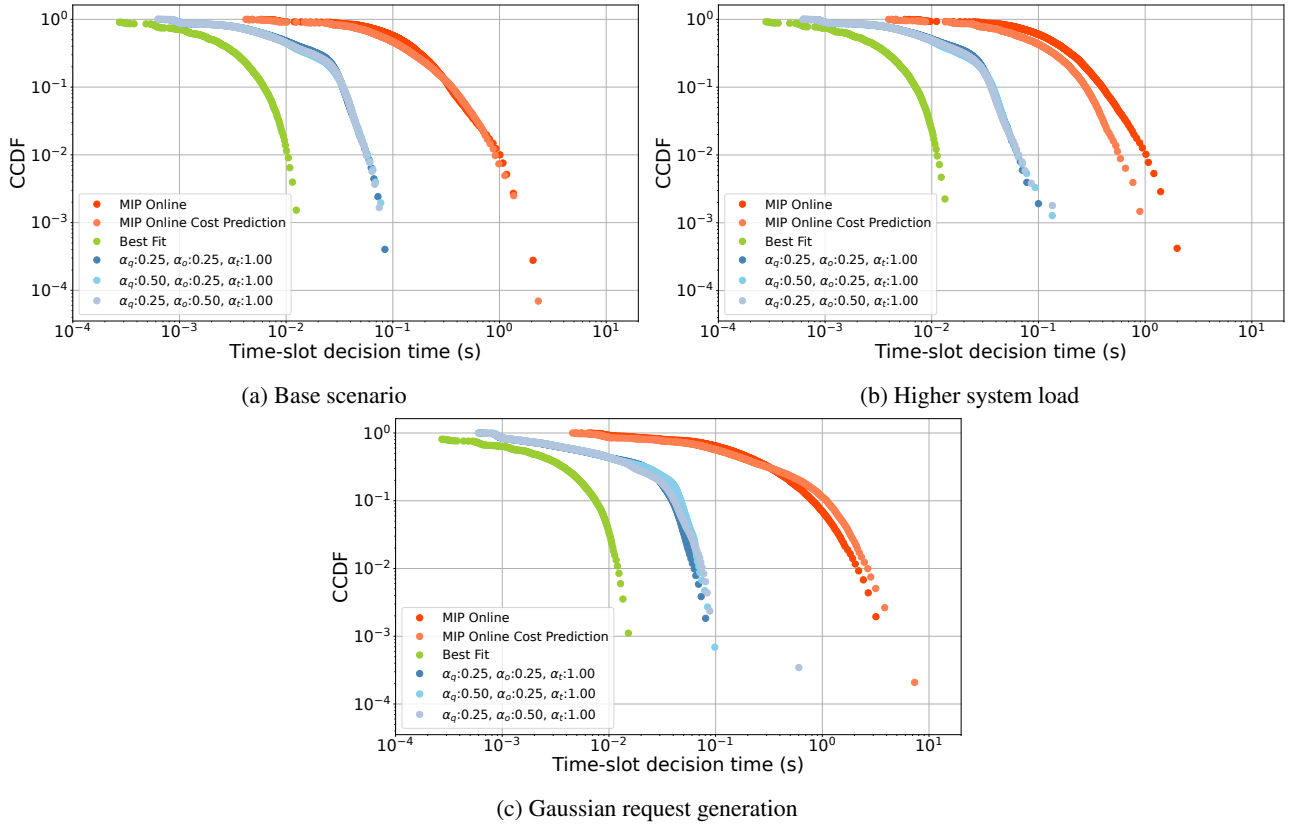


Figure 15: Complementary cumulative distribution function (CCDF) of the decision process time (in seconds) for all online policies in all considered scenarios (log scale).

Table 3

Descriptive statistics of the decision process time (in milliseconds) for all online policies.

Policy	Base			Higher load			Gaussian pattern		
	Mean	Std.	Med.	Mean	Std.	Med.	Mean	Std.	Med.
Best Fit	2.69	2.44	2.03	3.07	2.73	2.31	2.91	3.05	1.88
$\alpha_q:0.25, \alpha_o:0.25, \alpha_i:1.00$	14.73	17.98	9.19	16.36	20.75	10.38	15.16	21.1	6.94
$\alpha_q:0.50, \alpha_o:0.25, \alpha_i:1.00$	14.01	19.06	8.59	15.63	22.09	9.47	17.14	24.87	7.04
$\alpha_q:0.25, \alpha_o:0.50, \alpha_i:1.00$	14.15	18.09	8.47	15.72	17.79	9.88	15.36	29.29	7.42
MIP Online Cost Prediction	152.34	186.69	91.0	120.92	126.64	80.01	371.05	584.95	130.67
MIP Online	167.98	184.56	119.04	185.76	199.36	128.32	321.87	473.21	164.95

example, the sorting steps of Algorithm 1 can be customized to consider different operating goals.

As for the choice of DRL algorithm, we selected DDQN which is designed for finite action space, like the other learning algorithms of the DQN family. In order to deal with potential explosion of action space dimension, we chose sensible values (see Table 1) which are a trade-off between a manageable space dimension and meaningful agent decisions. Our experiments have shown that the DRL agent decisions produce results close to those obtained by an optimal offline policy, which operates in a much larger action space. Whenever we need to deploy new services with specific or finer threshold requirements, we have two options to follow. The first is to train the DRL agent by considering a larger and service-tailored action space. However, this straightforward option may lead to intractable action spaces

and over-fitting problems. The alternative option is to change DRL algorithm by exploiting the modular architecture of our orchestration framework. For example, DRL algorithms belonging to Deep Deterministic Policy Gradient (DDPG) family, such as Twin Delayed Deep Deterministic Policy Gradients (TD3) [62] and Proximal Policy Optimization (PPO) [63], are suitable candidates for handling continuous action space, although they require longer learning phases.

Finally, we observe that in this paper we focus on a family of services, rather than a specific one. This leads to considering a high-level model of the physical and service layers with the aim of describing a wide range of network delays (see Section 8). This approach enables to achieve an overall understanding of the performance of a DRL-based orchestration framework in a very general setting, and it can

be easily deployed for managing services with similar characteristics. Of course, in case we need to orchestrate services with very specific requirements and dynamics, that cannot be captured by a general network and service model, further modeling of the environment and agent's observation space is required. This could lead, for example, to consider specific communication protocols [64] and standard orchestration framework architectures. Moreover, the DRL agent can be easily integrated with more sophisticated simulation tools, such as OMNeT++⁴ and ns-3⁵, in a holistic co-simulation framework.

11. Conclusions

In this paper, we presented an online orchestration algorithm based on DRL to manage the life-cycle of session-based services, such as multi-player online games and video conferences. This class of services is characterized by stateful, long-lived, and shared instances which require flexible modeling and algorithmic solutions. We presented a mathematical formulation of the problem based on generalized assignment and scheduling problem with elastic capacities (eGAP) with a multi-objective function accounting for the dynamic of network conditions. We designed an orchestration framework based on a combination of DRL and a parametric combinatorial model, which is suitable for online service orchestration.

Our computational results show that our online algorithm based on DRL leads to solutions whose gap with respect to a theoretical offline optimal policy remains very limited. Moreover, the learned policy is able to guarantee a suitable QoS level even in high load scenarios and unexpected load patterns, at the cost of using a limited amount of additional resources. Moreover, our policy is able to maintain a fair load balancing among facilities, close to the optimal one. Finally, the obtained figures of computational time show that our solution is suitable for addressing near-real-time decision processes. As an overall intuition, we argue that the main factor yielding our method to achieve better performance than existing ones is its combination of agent decision and action implementation. As a matter of fact, our agent is allowed to implicitly explore a combinatorial set of options related to each action, thereby implicitly taking into account a very large set of optimization possibilities.

As future directions of this research, in addition to adopting DRL algorithms belonging to DDPG family to deal with continuous action space, we are considering alternative models of the observable state eliminating the dependency on a fixed set of facilities. The goal is to obtain an even more general model of the edge computing system allowing for dynamic edge network topology, as in the case of multi-provider/multi-tenant scenarios.

⁴<https://omnetpp.org/>

⁵<https://www.nsnam.org/>

Acknowledgments

This work was partially supported by project SERICS (PE00000014) under the MUR NRRP funded by the EU-NextGenerationEU

References

- [1] J. Navarro-Ortiz, P. Romero-Diaz, S. Sendra, P. Ameigeiras, J. J. Ramos-Munoz, J. M. Lopez-Soler, A survey on 5g usage scenarios and traffic models, *IEEE Communications Surveys & Tutorials* 22 (2) (2020) 905–929. doi:10.1109/COMST.2020.2971781.
- [2] M. Agiwal, A. Roy, N. Saxena, Next generation 5g wireless networks: A comprehensive survey, *IEEE Communications Surveys & Tutorials* 18 (3) (2016) 1617–1655. doi:10.1109/COMST.2016.2532458.
- [3] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, T. Turletti, A survey of software-defined networking: Past, present, and future of programmable networks, *IEEE Communications Surveys & Tutorials* 16 (3) (2014) 1617–1634. doi:10.1109/SURV.2014.012214.00180.
- [4] S. Kianpisheh, T. Taleb, A survey on in-network computing: Programmable data plane and technology specific applications, *IEEE Communications Surveys & Tutorials* (2022) 1–1. doi:10.1109/COMST.2022.3213237.
- [5] B. Yi, X. Wang, K. Li, S. k. Das, M. Huang, A comprehensive survey of network function virtualization, *Computer Networks* 133 (2018) 212–262. doi:10.1016/j.comnet.2018.01.021.
- [6] C.-H. Hong, B. Varghese, Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms, *ACM Comput. Surv.* 52 (5) (sep 2019). doi:10.1145/3326066.
- [7] F. Spinelli, V. Mancuso, Toward enabled industrial verticals in 5g: A survey on mec-based approaches to provisioning and flexibility, *IEEE Communications Surveys & Tutorials* 23 (1) (2021) 596–630. doi:10.1109/COMST.2020.3037674.
- [8] N. Hassan, K.-L. A. Yau, C. Wu, Edge computing in 5g: A review, *IEEE Access* 7 (2019) 127276–127289.
- [9] A. Furno, M. Fiore, R. Stanica, C. Ziemlicki, Z. Smoreda, A tale of ten cities: Characterizing signatures of mobile traffic in urban areas, *IEEE Transactions on Mobile Computing* 16 (10) (2017) 2682–2696. doi:10.1109/TMC.2016.2637901.
- [10] S. Schneider, R. Khalili, A. Manzoor, H. Qarawlus, R. Schellenberg, H. Karl, A. Hecker, Self-learning multi-objective service coordination using deep reinforcement learning, *IEEE Transactions on Network and Service Management* 18 (3) (2021) 3829–3842. doi:10.1109/TNSM.2021.3076503.
- [11] J. Gil Herrera, J. F. Botero, Resource allocation in nfv: A comprehensive survey, *IEEE Transactions on Network and Service Management* 13 (3) (2016) 518–532. doi:10.1109/TNSM.2016.2598420.
- [12] Q. Luo, S. Hu, C. Li, G. Li, W. Shi, Resource scheduling in edge computing: A survey, *IEEE Communications Surveys Tutorials* 23 (4) (2021) 2131–2165. doi:10.1109/COMST.2021.3106401.
- [13] P. Mach, Z. Becvar, Mobile edge computing: A survey on architecture and computation offloading, *IEEE Communications Surveys Tutorials* 19 (3) (2017) 1628–1656. doi:10.1109/COMST.2017.2682318.
- [14] F. Dressler, C. F. Chiasserini, F. H. Fitzek, H. Karl, R. L. Cigno, A. Capone, C. Casetti, F. Malandrino, V. Mancuso, F. Klingler, G. Rizzo, V-edge: Virtual edge computing as an enabler for novel microservices and cooperative computing, *IEEE Network* 36 (3) (2022) 24–31. doi:10.1109/MNET.001.2100491.
- [15] A. Tsipis, K. Oikonomou, Player assignment in mec gaming for social interactivity and server provisioning optimization, in: *2021 IEEE Symposium on Computers and Communications (ISCC)*, 2021, pp. 1–7. doi:10.1109/ISCC53001.2021.9631480.
- [16] X. Zhang, H. Chen, Y. Zhao, Z. Ma, Y. Xu, H. Huang, H. Yin, D. O. Wu, Improving cloud gaming experience through mobile edge computing, *IEEE Wireless Communications* 26 (4) (2019) 178–183. doi:10.1109/MWC.2019.1800440.

- [17] A. R. Benamer, K. Boussetta, N. B. Hadj-Alouane, A genetic algorithm for the placement of latency-sensitive multiplayer game servers in the fog, in: 2021 IEEE Global Communications Conference (GLOBECOM), 2021, pp. 1–6. doi:10.1109/GLOBECOM46510.2021.9685952.
- [18] Y. Gao, C. Zhang, Z. Xie, Z. Qi, J. Zhou, Cost-efficient and quality-of-experience-aware player request scheduling and rendering server allocation for edge-computing-assisted multiplayer cloud gaming, IEEE Internet of Things Journal 9 (14) (2022) 12029–12040. doi:10.1109/JIOT.2021.3132849.
- [19] L. Wang, L. Jiao, T. He, J. Li, H. Bal, Service placement for collaborative edge applications, IEEE/ACM Transactions on Networking 29 (1) (2021) 34–47. doi:10.1109/TNET.2020.3025985.
- [20] A. Tsipis, K. Oikonomou, Joint optimization of social interactivity and server provisioning for interactive games in edge computing, Computer Networks 212 (2022) 109028. doi:10.1016/j.comnet.2022.109028.
- [21] P. B. Mirchandani, R. L. Francis, Discrete Location Theory, Wiley, 1990.
- [22] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, L. Tassiulas, Joint service placement and request routing in multi-cell mobile edge computing networks, in: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, 2019, pp. 10–18. doi:10.1109/INFOCOM.2019.8737385.
- [23] V. Farhadi, F. Mehmeti, T. He, T. L. Porta, H. Khamfroush, S. Wang, K. S. Chan, Service placement and request scheduling for data-intensive applications in edge clouds, in: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, 2019, pp. 1279–1287. doi:10.1109/INFOCOM.2019.8737368.
- [24] S. Pasteris, S. Wang, M. Herbster, T. He, Service placement with provable guarantees in heterogeneous edge computing systems, in: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, 2019, pp. 514–522. doi:10.1109/INFOCOM.2019.8737449.
- [25] P. Lai, Q. He, G. Cui, F. Chen, M. Abdelrazek, J. Grundy, J. Hosking, Y. Yang, Quality of experience-aware user allocation in edge computing systems: A potential game, in: 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS), 2020, pp. 223–233. doi:10.1109/ICDCS47774.2020.00036.
- [26] F. Chiti, R. Fantacci, F. Paganelli, B. Picano, Virtual functions placement with time constraints in fog computing: A matching theory perspective, IEEE Transactions on Network and Service Management 16 (3) (2019) 980–989. doi:10.1109/TNSM.2019.2918637.
- [27] F. Santos, R. Immich, E. R. Madeira, Multimedia services placement algorithm for cloud-fog hierarchical environments, Computer Communications 191 (2022) 78–91. doi:10.1016/j.comcom.2022.04.009.
- [28] R. Mahmud, S. N. Srirama, K. Ramamohanarao, R. Buyya, Quality of experience (qoe)-aware placement of applications in fog computing environments, Journal of Parallel and Distributed Computing 132 (2019) 190–203. doi:10.1016/j.jpdc.2018.03.004.
- [29] H. Badri, T. Bahreini, D. Grosu, K. Yang, Energy-aware application placement in mobile edge computing: A stochastic optimization approach, IEEE Transactions on Parallel and Distributed Systems 31 (4) (2020) 909–922. doi:10.1109/TPDS.2019.2950937.
- [30] Y. Li, W. Liang, J. Li, Profit Maximization for Service Placement and Request Assignment in Edge Computing via Deep Reinforcement Learning, Association for Computing Machinery, New York, NY, USA, 2021, p. 51–55. doi:10.1145/3479239.3485673.
- [31] Z. Zhou, Q. Wu, X. Chen, Online orchestration of cross-edge service function chaining for cost-efficient edge computing, IEEE Journal on Selected Areas in Communications 37 (8) (2019) 1866–1880. doi:10.1109/JSAC.2019.2927070.
- [32] M. Bagaa, T. Taleb, J. Bernal Bernabe, A. Skarmeta, Qos and resource-aware security orchestration and life cycle management, IEEE Transactions on Mobile Computing (2020) 1–1. doi:10.1109/TMC.2020.3046968.
- [33] Z. Lin, S. Bi, Y.-J. A. Zhang, Optimizing ai service placement and resource allocation in mobile edge intelligence systems, IEEE Transactions on Wireless Communications 20 (11) (2021) 7257–7271. doi:10.1109/TWC.2021.3081991.
- [34] A. Hazra, P. K. Donta, T. Amgoth, S. Dustdar, Cooperative transmission scheduling and computation offloading with collaboration of fog and cloud for industrial iot applications, IEEE Internet of Things Journal (2022) 1–1. doi:10.1109/JIOT.2022.3150070.
- [35] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533. doi:10.1038/nature14236.
- [36] R. S. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, A Bradford Book, Cambridge, MA, USA, 2018.
- [37] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, D. I. Kim, Applications of deep reinforcement learning in communications and networking: A survey, IEEE Communications Surveys Tutorials 21 (4) (2019) 3133–3174. doi:10.1109/COMST.2019.2916583.
- [38] Y. Hao, M. Chen, H. Gharavi, Y. Zhang, K. Hwang, Deep reinforcement learning for edge service placement in software-defined industrial cyber-physical system, IEEE Transactions on Industrial Informatics 17 (8) (2021) 5552–5561. doi:10.1109/TII.2020.3041713.
- [39] J. Pei, P. Hong, M. Pan, J. Liu, J. Zhou, Optimal vnf placement via deep reinforcement learning in sdn/nfv-enabled networks, IEEE Journal on Selected Areas in Communications 38 (2) (2020) 263–278. doi:10.1109/JSAC.2019.2959181.
- [40] R. M. Nauss, The elastic generalized assignment problem, Journal of the Operational Research Society 55 (12) (2004) 1333–1341. arXiv:10.1057/palgrave.jors.2601806, doi:10.1057/palgrave.jors.2601806.
- [41] Gurobi Optimization, LLC, Gurobi Optimizer Reference Manual (2021). URL <https://www.gurobi.com>
- [42] S. D. Jena, J.-F. Cordeau, B. Gendron, Dynamic facility location with generalized modular capacities, Transportation Science 49 (3) (2015) 484–499. arXiv:10.1287/trsc.2014.0575, doi:10.1287/trsc.2014.0575.
- [43] A. Silva, D. Aloise, L. C. Coelho, C. Rocha, Heuristics for the dynamic facility location problem with modular capacities, European Journal of Operational Research 290 (2) (2021) 435–452. doi:10.1016/j.ejor.2020.08.018.
- [44] J. Martín-Pérez, L. Cominardi, C. J. Bernardos, A. de la Oliva, A. Azcorra, Modeling mobile edge computing deployments for low latency multimedia services, IEEE Transactions on Broadcasting 65 (2) (2019) 464–474. doi:10.1109/TBC.2019.2901406.
- [45] Y. Dang, H. Cheng, F. Li, S. Yang, Research on fairness algorithm of user allocation problem in moba edge gaming, in: 2022 IEEE 96th Vehicular Technology Conference (VTC2022-Fall), 2022, pp. 1–5. doi:10.1109/VTC2022-Fall157202.2022.10012913.
- [46] Y. Chen, J. Liu, Y. Cui, Inter-player delay optimization in multiplayer cloud gaming, in: 2016 IEEE 9th International Conference on Cloud Computing (CLOUD), 2016, pp. 702–709. doi:10.1109/CLOUD.2016.0098.
- [47] P. Brucker, Scheduling Algorithms, Springer, 2007.
- [48] S. K. C. Chekuri, A ptas for the multiple knapsack problem, SIAM Journal on Computing 35 (2006) 713–728.
- [49] M. Suznjevic, O. Dobrijevic, M. Matijasevic, Mmorpg player actions: Network performance, session patterns and latency requirements analysis, Multimedia Tools and Applications 45 (1) (2009) 191–214. doi:10.1007/s11042-009-0300-1.
- [50] C. Quadri, M. Zignani, L. Capra, S. Gaito, G. P. Rossi, Multidimensional human dynamics in mobile phone communications, PLOS ONE 9 (7) (2014) 1–12. doi:10.1371/journal.pone.0103183.
- [51] Telecom Italia big data challenge (2016). [link]. URL <http://www.telecomitalia.com/bigdatachallenge>
- [52] R. Cleveland, W. S. Cleveland, J. E. McRae, I. J. Terpenning, Stl: A seasonal-trend decomposition procedure based on loess (with discussion), 1990.

- [53] L. Kleinrock, Time-shared systems: A theoretical treatment, *Journal of the ACM (JACM)* 14 (2) (1967) 242–261.
- [54] M. Papandrea, K. K. Jahromi, M. Zignani, S. Gaito, S. Giordano, G. P. Rossi, On the properties of human mobility, *Computer Communications* 87 (2016) 19–36. doi:10.1016/j.comcom.2016.03.022.
- [55] Riot Games (2021). [link]. URL <https://developer.riotgames.com/>
- [56] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, Openai gym (2016). arXiv:1606.01540.
- [57] J. Weng, H. Chen, D. Yan, K. You, A. Duburcq, M. Zhang, H. Su, J. Zhu, Tianshou: A highly modularized deep reinforcement learning library, arXiv preprint arXiv:2107.14171 (2021).
- [58] H. van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, arXiv preprint arXiv:1509.06461 (2015). arXiv:1509.06461.
- [59] T. Schaul, J. Quan, I. Antonoglou, D. Silver, Prioritized experience replay, arXiv preprint arXiv:1511.05952 (2016). arXiv:1511.05952.
- [60] Q. Luo, S. Hu, C. Li, G. Li, W. Shi, Resource scheduling in edge computing: A survey, *IEEE Communications Surveys Tutorials* 23 (4) (2021) 2131–2165. doi:10.1109/COMST.2021.3106401.
- [61] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, X. Costa-Perez, Deepcog: Cognitive network management in sliced 5g networks with deep learning, in: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 280–288. doi:10.1109/INFOCOM.2019.8737488.
- [62] S. Fujimoto, H. van Hoof, D. Meger, Addressing function approximation error in actor-critic methods, in: J. Dy, A. Krause (Eds.), *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80 of *Proceedings of Machine Learning Research*, PMLR, 2018, pp. 1587–1596.
- [63] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms (2017). doi:10.48550/ARXIV.1707.06347.
- [64] P. K. Donta, S. N. Srirama, T. Amgoth, C. S. R. Annavarapu, Survey on recent advances in iot application layer protocols and machine learning scope for research directions, *Digital Communications and Networks* 8 (5) (2022) 727–744. doi:10.1016/j.dcan.2021.10.004.

A. Background traffic reconstruction

The network delay depends on the time for traversing the base station, the delay on backhaul links and processing time at the facility. We assume that the delay experienced by users is entirely caused by the background traffic and that: (i) each base station is properly equipped with enough capacity to process all the amount of traffic demand generated according to its weekly pattern; (ii) the background traffic from each base station is routed to the closest facility; (iii) the backhaul delay is negligible; (iv) the processing time at facility is a function of the background traffic to be managed. To map the amount of traffic to the network delay, we apply the results in [53] where for time-shared systems the processing time grows exponentially. In particular, we use the following equation:

$$d = \frac{d_{min}}{1 - \frac{l}{\eta L_{max}}} \quad (18)$$

where d_{min} is the minimum time when no traffic is present, l is the current level of traffic, L_{max} is the maximum level of traffic that can be handled and $\eta > 1$ (we use $\eta = 1.1$) controls the maximum increment of delay. In our experiments, we model the BS delay assuming $d_{min} = 1ms$ and L_{max} different for each base station according to assumption (i).

This configuration leads to a maximum delay for traversing BS of 10 ms. As for the facility processing time, we consider $d_{min} = 3ms$, while L_{max} is equal for all facilities and is equal to the peak of background traffic uniformly distributed over all facilities. Based on these considerations, $RTT = 2d_{BS} + d_f$ is bounded between 5ms and 50ms.

B. Datasets

B.1. Call Detail Records (CDRs) dataset

Call Detail Records (CDRs) dataset gathers the phone activities of about one million subscribers to one of the largest Italian mobile operators [50]. This dataset covers the metropolitan area of Milan for a time period of 9 weeks and provides the approximate location of subscribers with a granularity of associated base stations. The dataset contains about 100 million calls, 52 million text messages, and 61 million Internet records. Each record is provided with the anonymized subscriber identity, the timestamp, and the identification number together with the toponymic of the base station where the subscriber was registered when he/she performed the activity. For privacy and security reasons, the location information of each record does not include the GPS location of the base station. Therefore, we employed the web service LocationAPI offered by UnwiredLabs⁶ to retrieve the approximated GPS location.

B.2. TIM BigData Challenge

This dataset has been provided by Telecom Italia Mobile (TIM), as part of their Big Data Challenge initiative [51], which contains mobile Internet traffic information over a time period of 2 months aggregated over 10-minute time intervals, according to a regular-cell ($235 \times 235 m^2$) spatial tessellation of the city of Milan. For privacy and security reasons the information about the mobile operator released the dataset by obfuscating the actual amount of traffic using a custom function that preserves the proportionality.

B.3. MOBA matches dataset

We collect information about match creation requests of the popular Multiplayer Online Battle Arena (MOBA) League of Legends (LoL) through the public APIs service provided by the Riot Games developers [55]. We collected more than 7 million matches played for a time period of almost 2 years. We started from a small seed of users whose ID is publicly available. and we perform a breadth-first exploration. For each user, we collect all the matches played within a specific period by the user along with all the other players involved in the matches. Due to the limited amount of requests per hour allowed by free account APIs, we were not able to exhaustively sample all the match requests, so we stopped the crawling task as soon as we had collected sufficient data for reconstructing the pattern of service request generation.

⁶UnwiredLabs available at <https://unwiredlabs.com/>