




Value is King: The MECForge Deep Reinforcement Learning Solution for Resource Management in 5G and Beyond

Filippo Poltronieri¹ · Cesare Stefanelli¹ · Niranjani Suri^{2,3} · Mauro Tortonesi⁴ 

Received: 2 July 2021 / Revised: 13 May 2022 / Accepted: 3 July 2022 /
Published online: 27 July 2022
© The Author(s) 2022

Abstract

Multi-access edge computing (MEC) is a key enabler to fulfill the promises of a new generation of immersive and low-latency services in 5G and Beyond networks. MEC represents a defining function of 5G, offering significant computational power at a reduced latency, allowing to augment the capabilities of user equipments while preserving their battery life. However, the demands generated by a plethora of innovative and concurrent IT services requiring high quality of service and quality of experience levels will likely overwhelm the—albeit considerable—resources available in 5G and Beyond scenarios. To take full advantage of its potential, MEC needs to be paired with innovative resource management solutions capable of effectively addressing the highly dynamic aspects of the scenario and of properly considering the heterogeneous and ever-changing nature of next generation IT services, prioritizing the assignment of resources in a highly dynamic and contextual fashion. This calls for the adoption of Artificial Intelligence based tools, implementing self-* approaches capable of learning the best resource management strategy to adapt to the ever changing conditions. In this paper, we present MECForge, a novel solution based on deep reinforcement learning that considers the maximization of total value-of-information delivered to end-user as a coherent and comprehensive resource management criterion. The experimental evaluation we conducted in a simulated but realistic environment shows how the Deep Q-Network based algorithm implemented by MECForge is capable of learning effective autonomous resource management policies that allocate service components to maximize the overall value delivered to the end-users.

Keywords 5G and Beyond · Multi-access edge computing (MEC) · Value of information (VoI) · Deep reinforcement learning (DRL)

✉ Mauro Tortonesi
mauro.tortonesi@unife.it

Extended author information available on the last page of the article

1 Introduction

The deployment of 5G communications is opening up new computing scenarios that enable the next generation of immersive applications leveraging distributed computing resources in proximity to end-users [1]. To this end, applications can take advantage of the functions provided by Multi-access Edge Computing (MEC) to run software components in relatively resource rich servers that communicate with mobile devices at very low latency (1–10 msec).

In fact, together with Non Orthogonal Multiple Access (NOMA) which significantly improves network density and spectrum efficiency, MEC arguably represents the defining feature of 5G [2]. While per se, not a novel concept—previous incarnations of the data-center-at-the-edge concept, such as Cloudlets, go as far back as the early 00's—MEC represents the first widely available and commercially viable implementation of edge computing on public infrastructure. These capabilities are supposed to be pushed forward even further in the near future, with the development and deployment of so-called Beyond 5G technologies.

However, while 5G and Beyond environments will be relatively resourced rich in terms of computation, bandwidth, and storage, it is conceivable that the resource demands generated by a plethora of innovative services will saturate the available resources. 5G and Beyond applications will thus require smart, adaptive, and robust resource management solutions, capable of dealing with the highly dynamic nature of the environment and the challenging demands of a new generation of immersive, context-aware, and latency-sensitive services [3, 4].

The optimal resource management in 5G has been investigated from several perspectives, including latency and energy consumption minimization. However, most of those approaches follow an operator-centric perspective and assume relatively low dynamicity, which is not necessarily aligned to the performance experienced by end-users. Instead, we argue that optimizing resource management to maximize the utility of end-users represents a compelling avenue of research, which provides interesting opportunities to investigate and develop innovative methodologies and tools to consider the value that an IT service provides to the users from a comprehensive perspective [5].

More specifically, considering the MEC services instantiation and offloading aspects of resource management in 5G and Beyond environments, there is the need to identify which configuration can provide the highest value for the end-users. While some efforts have been proposed so far, there is still wide room for investigation to solve the remaining challenges, e.g., dynamic and adapting network slicing for real-time applications [6, 7], and so on. This requires next generation resource management solutions that provide two fundamental characteristics: they will need to be adaptive and capable to address the highly dynamic aspects of the scenario and will need to consider the heterogeneous and ever-changing nature of next generation IT services in resource assignment criteria.

These issues arguably call for the adoption of Artificial Intelligence (AI) techniques, which researchers have identified as a key enabling technology for the management of future networks [8–11]. Therefore, it is essential to identify the key

contributions that AI and Machine Learning (ML) can bring to the management of future networks to face the increasing complexity of new generation networks to find a correct configuration of resources without incurring into further complexity for tuning the optimization parameters of these models [12, 13].

Within AI, a different and potentially very promising research avenue lies in the adoption of approaches capable of learning the best strategies to adapt to the current environmental conditions, e.g., network and computing resources and users' demand [14]. In particular, reinforcement learning (RL) is a field of AI that attracts the increasing attention of the research community [15–17]. RL provides a rather simplistic but effective approach inspired by trial and error that mimics the behavior of human intelligence through reward maximization [18]. Therefore, MEC resource management solutions can leverage RL for automatically tuning the configuration parameters when the environmental conditions change to guarantee the delivery of the expected QoS and QoE.

Motivated by the promising capabilities of such techniques, this work investigates a novel methodology for resource management in 5G and Beyond and proposes the application of a deep reinforcement learning (DRL) algorithm to address the continuously changing requirements of these scenarios, such as demand variations, network and resource fluctuations, and users and devices mobility. To achieve that goal, we focus on maximization of total value-of-information (VoI) delivered to the end-users as resource management policy—and use that as a reward function for our DRL-based solution. VoI methodologies and tools aim to find an optimal configuration for the available computational and network resources [5] which prioritizes the most important data to be processed and disseminated, thus effectively addressing the data deluge of IoT applications. As a promising concept for addressing information management and prioritization in constrained environments, VoI has been recently proposed for resource management in several works [19–21].

Built upon our previous work [22], in which we formalized an optimization framework for the value-based management of fog services, this paper investigates DRL for MEC resource management by presenting a use-case scenario and its optimization. In this paper, we first discuss the application of MEC to a Smart City scenario, and then we formulate a problem description and a DRL approach called MECForge to maximize the amount of VoI that a given MEC resource allocation can provide to its users. More specifically, MECForge leverages Deep Q-Network (DQN) that given a MEC resource configuration is capable to migrate and/or activate service components to find configurations that achieve increased value-based utility for end-users.

The remainder of the paper is organized as follows. Section 2 introduces RL and discusses related efforts. Section 3 presents an overview of MEC in 5G and beyond to introduce the particular topic on which the contributions of this manuscript belong. Then, Sect. 4 illustrates the system model presenting the problem formulation and the concept of VoI for MEC resource management. Section 5 presents MECForge, an implementation of a Deep Q-Network (DQN) algorithm to solve the VoI resource allocation problem in MEC scenarios. Finally, Sect. 6 presents a comprehensive evaluation of MECForge and Sect. 7 concludes this manuscript.

2 Background and Related Work

Reinforcement Learning (RL) is an evolving field of AI, in which a software agent interacts with an environment to learn the best possible actions that maximize a reward [23]. Given the promising capabilities of RL, several approaches have been proposed to deal with problems of different sizes and complexity. Among them, deep reinforcement learning (DRL) represents a compelling set of algorithms to solve complex and large problems. DRL is a subset of RL, which involves the application of machine learning methodologies such as neural networks to avoid the memory and time limitations of standard RL approaches.

The applicability of RL and DRL tools for network management has been investigated in several works, such as the optimal placement of virtual network functions (VNF) [24, 25], energy-efficient resource allocation [26], and latency minimization [27]. Differently from supervised learning methods, that require human intervention for labeling data, RL allows to naturally train a software agent to learn an optimal policy by interacting directly with the environment. This provides a valuable tool to tame the dynamicity of these environments, which require continuous interventions to manage the available resources and meet the current applications' requirements [4, 28].

5G and Beyond networks would bring enormous capabilities from both a network and computing perspective to the end-users at the edge of the network [8]. However, such scenarios would represent dynamic and challenging environments presenting both a higher availability of resources in terms of bandwidth, computing powers, lower latency but also a higher demand from its users. To tackle this increasing dynamicity there is the need for novel solutions capable of exploiting non-static optimization methodologies, such as machine-learning techniques for the online management of resources at the edge of the cellular network.

Machine learning (ML) can be particularly useful to address network management challenges, such as traffic prediction and network monitoring. To this end, De Shepper et. al describe a traffic classification approach based on Convolution Neural Network (CNN) to recognize User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) traffic along with network bursts and data rates in [29]. Another approach exploiting CNN for network management is [30], in which the authors present a framework for spectrum management in Wi-Fi networks. Similarly, in [31] the authors present a Graph Neural-Network (GNN) approach to address interference management in Wi-Fi networks.

As for ML, RL can well suit this kind of management challenges, as demonstrated by a recent survey that analyses the application of DRL in networking [32]. Li et al. investigate the applicability of RL to network slicing and resource management in [33]. More specifically, the authors formulate a radio resource slicing problem using DRL to find a bandwidth-sharing solution that maximizes resource utilization and QoE. Then, they discuss that the same problem can be applied to priority-based scheduling of virtualized network functions (VNF). In another work [34], the authors introduce a DRL framework for dealing with network slicing under heterogeneous resource requirements and dynamic traffic

demands from network users. To achieve this objective, the authors propose to adopt DRL algorithms to maximize the overall QoS by reducing the delay to process requests. Montero et al. discuss the importance of network slicing for service management in 5G networks in [35]. Liu et al. discuss an interesting formulation to divide the resource allocation problem for network slicing into a master and slave problem to reduce its complexity in [36]. More specifically, the authors propose an algorithm called DeepSlicing to find the resource allocation policy that maximizes the utility for users. In [37], the authors present a decentralized DRL approach for network slicing to achieve optimal orchestration of network resources by proposing a detailed architecture with multiple orchestration agents.

DRL has been investigated for the resolution of service placement problems. A recent work discussing online and fault-tolerant SFC placement using DRL is [40]. Dab et al. formalize an RL problem to learn the best offloading decisions to minimize energy consumption on the devices-side under latency constraints for 5G applications in [39]. Long-Term latency minimization for fog computing using RL is also discussed in [27]. In particular, this paper proposes an RL approach combined with evolution strategies for dealing with real-time task assignments and reducing computation latency in the long-term period. Finally, Goethals et al. present a self-organizing service scheduler for fog and edge networks with hundreds of nodes in [14].

In [38], Nakanoya et al. propose an interesting and cost-effective technique for applying RL to online optimization of VNFs sizing and placement. In particular, the authors propose a two-step RL that divides the learning process with the aim of decreasing the learning exploration steps. Pujol et al. present a DRL-based approach to deal with the online management and orchestration of VNFs in [41]. More specifically, they propose an algorithm called PAT that leverages an actor-critic method to learn how to configure network resources and when to offload the execution of VNFs. In [4] Chen et al. discuss two Double Deep Q-Network (DDQN) learning algorithms for task offloading decisions in MEC in ultra-dense sliced Radio Access Network (RAN). More specifically, this paper proposes two DDQN learning algorithms to solve computational offloading under dynamic network conditions to maximize the long-term utility performance.

The authors in [43] discuss a Q-learning-based load-balancing algorithm for fog networks that enables to reduce the processing time and overload probability of networks. Another formulation leveraging DQN is the one presented in [42]. In this work, the authors address the problem of best task offloading and bandwidth allocation in MEC scenarios by proposing the Joint Task Offloading and Bandwidth Allocation (JTOBA) algorithm. Kim et al. present the application of DRL to Active Queue Management (AQM) policy to deal with the deluge of traffic generated by IoT devices in fog and edge networks. A different application for DRL is described in [44], in which a Fast Task Allocation (FTA) algorithm leveraging DRL is proposed to allocate tasks among heterogeneous UAVs. Finally, Table 1 reports a summary of related works in network and service management.

Differently from related efforts, this work addresses computing resource management in MEC scenarios using novel Value-of-Information (VoI) methodologies and tools. We believe that VoI represents an interesting criterion to deal with

Table 1 Summary of related work

Works	Objectives	Methodologies	Metrics
[29]	Traffic patterns identification for network management	CNN	Traffic classification accuracy
[30, 31]	Wi-Fi networks management	CNN and GNN	Wi-Fi performance maximization
[24, 25, 38]	Optimal placement and orchestration of VNFs	DRL algorithms adaptations and heuristics	Acceptance ratio of VNFs placement and network performance
[26, 39]	Energy efficient resource allocation	Q-Learning, SARSA, and custom Q-Learning algorithms	Energy efficiency and network performance
[27]	Real-task assignment for latency minimization	RL and Evolution Strategies	Long term latency minimization
[33–35]	QoS and QoS maximization in network slicing	Deep Q-Learning and REINFORCE	Network performance and QoE satisfaction
[36]	Resource allocation for network slicing	Problem decomposition, convex optimization, and DRL	Users utility
[40]	Fault-tolerant SFC placement	Double Deep Q-Learning (DDQL) algorithm	Service availability
[41]	Online management of VNFs	Custom DRL algorithm	Network cost and performance
[4, 42]	Task-offloading and bandwidth allocation in MEC	DQN algorithms	Long-term utility and computation offloading performance
[43]	Load-balancing in fog networks	Q-Learning algorithm	Task processing delay
[44]	Task allocation among UAV's	DQN algorithms	Task allocation performance

the processing of mission-critical information, thus enabling to prioritize the most important offloading requests in case of resource saturation, i.e., when the computing resources available at MEC servers cannot meet the users' demand. Therefore, in this work we present a system model that evaluates the performance of MEC computing configurations considering the amount of VoI they can provide to end-users. Finally, VoI can bring great benefits to the management of computing and network resources of next-generation networks, which would require novel techniques for rapidly reconfiguring resources to adapt to the requirements of the most important services.

3 MEC for 5G and Beyond

5G and Beyond scenarios will enable a new generation of immersive and context-aware applications with low-latency requirements that cannot rely on cloud computing approaches but require computing capabilities located in the proximity of users and devices [45]. To this end, MEC represents a very interesting solution to fulfill this requirement. More specifically, MEC is a standard proposed by the European Telecommunications Standards Institute (ETSI) to bring computing capabilities in cellular networks [46].

MEC allows users to offload the processing of computational tasks to servers (or other computing equipment) installed by MEC providers (telco operators, cloud providers, municipality) at the edge of the cellular network, e.g., in proximity to base stations [2] as shown in Fig. 1.

More specifically, Fig. 1 shows an example deployment of a MEC scenario in a smart city environment with multiple software components (colored blocks in Fig. 1) deployed at MEC servers or in cloud facilities. Users can leverage the available software components through their User Equipments (UEs) by requesting their offloading (solid lines in Fig. 1). Furthermore, it is conceivable that existing software components can be migrated to MEC servers (dashed line in Fig. 1), which are managed by a MEC orchestrator.

As it finally promises to be a widely available and commercially viable solution for enabling low-latency computation and high levels of Quality-of-Experience (QoE) and Quality-of-Service (QoS) [47], MEC is poised to play an important role in 5G and Beyond applications, and for that reason, it is receiving an increasing attention in scientific literature.

It is expected that communication latency between UEs and MEC servers, located within the same gNodeB, will likely be very low (1–10 ms) [8, 48], thus enabling the aggressive offloading of computational tasks from UEs and the cloud to MEC servers and preserving the battery life of UEs. In addition, 5G and Beyond would overcome the limitations of previous-generation networks by providing high-data rates and ultra-low latency for a new range of services such as Augmented Reality (AR), Virtual Reality (VR), and Industry 4.0 [42].

As illustrated in Fig. 1, it is conceivable that many adaptive and dynamic IT services will run on the architecture described above, leveraging a plethora of software components executing concurrently on MEC servers with different

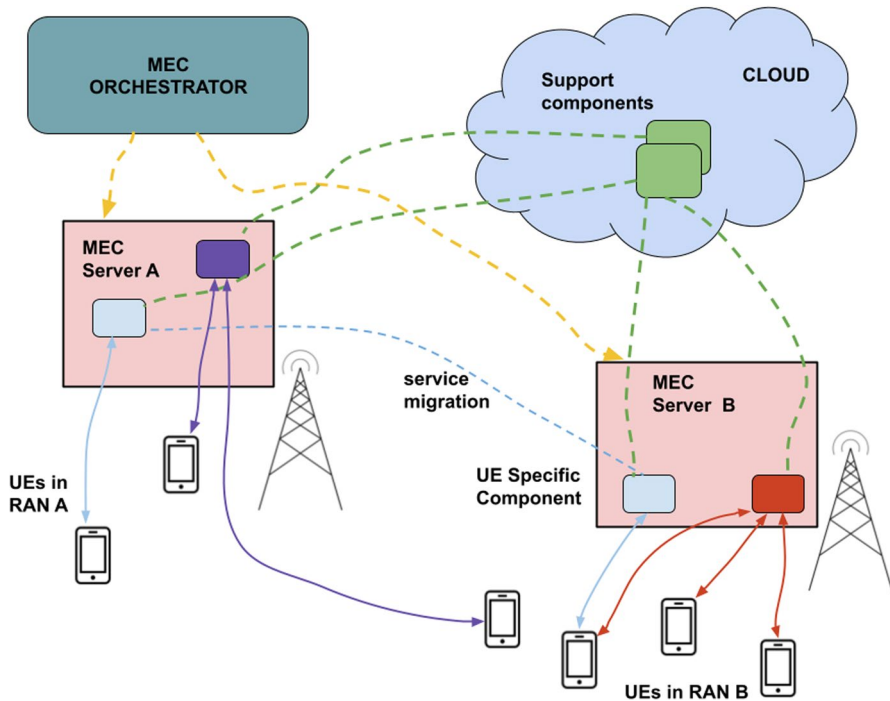


Fig. 1 An illustration of a MEC Scenario in a Smart City environment. Users can offload the processing of service components (colored blocks) to MEC servers within the RAN or other MEC servers connected to the 5G network

priorities, requirements, outreach, and mobility management policies. Some services, e.g., crowdsourcing/crowdsensing or widely available IT services, might leverage components that serve multiple UEs and are typically activated/deactivated in a given MEC server as UEs enter/leave the corresponding cell. Other services, e.g., autonomous driving or surveillance, need to leverage components that are guaranteed to receive the required share of resources and to run in the closest possible MEC server, possibly migrating between MEC servers as the UEs roam between cells [49].

Note that it is highly likely that only a relatively small share of these services will have a mission-critical nature that requires the static and preemptive assignment of the required resources in the MEC servers nearest to the corresponding UEs. Most services are instead very likely to provide a good QoE/QoS even when provided with a slightly suboptimal resource assignment, with, e.g., software components running on a near MEC server (but not on the nearest one), possibly supported by additional components running in the cloud.

In this situation, resource management solutions based on static and/or fixed priority resource assignment policies are likely to lead to suboptimal and wasteful resource allocations. Instead, there is the need for smart and adaptive AI-based

resource management solutions that explicitly consider contextual information and aggressively explore trade-offs in resource assignment in order to make the best use of available resources at the entire system level, avoiding processing bottlenecks, waste of resources, and maximizing the overall utility at the end-user level.

More specifically, the dynamic resource management and service orchestration for 5G and Beyond environments can particularly benefit from the adoption of self-* approaches based on Deep Reinforcement Learning (DRL) that are capable of learning the best resource management strategy to adapt to the ever-changing conditions. DRL is a relatively recent branch of reinforcement learning (RL) that has proved to be quite effective in dynamic and complex domains where the space of possible states is large and high dimensional [50]. As a result, it can well suit dynamic resource management problems such as MEC.

However, a DRL solution needs an accurate, coherent, and comprehensive criteria to evaluate the performance of IT service configurations. In the case of applications dealing with a significant number of users with heterogeneous interests, a large number of devices, and applications with different QoS and QoE requirements, this represents quite a challenge. In this context, *Value of Information (VoI)* maximization, i.e., the maximization of the utility that the information contained in service responses deliver to end recipients, represents an interesting subjective criterion [5]. In fact, VoI-based optimization would allow to naturally and seamlessly prioritize the assignment of resources to services that are providing the highest value to their end-users – either because they are serving a considerable amount of users or because they are providing highly valuable information.

4 System Model

In this work, we assume a 5G and Beyond scenario with multiple Base Stations (BS) each one provided with a MEC server S_i as shown in Fig. 2. At each BS a Radio Access Network (RAN) provides communication capabilities to User Equipments (UEs) and MEC servers. We also assume that MEC servers are federated using the 5G core network, thus users can offload the processing of service components not only to the closest MEC server but also to the other servers available in a geographical area, e.g. a Smart City.

With regard to the offloading of service components in MEC, we assume that a user can offload the computation of a service component to a MEC server instead of processing it locally. In fact, some service components could not be processed on the UE because they require dedicated resources (large neural networks, GPU, etc.). In general, there are several variables that contribute to the offloading decision such as remaining battery life, reduced computing latency, QoS and QoE requirements, and so on. This work assumes that all service components required by users residing at the edge of the network are offloaded to MEC servers for processing.

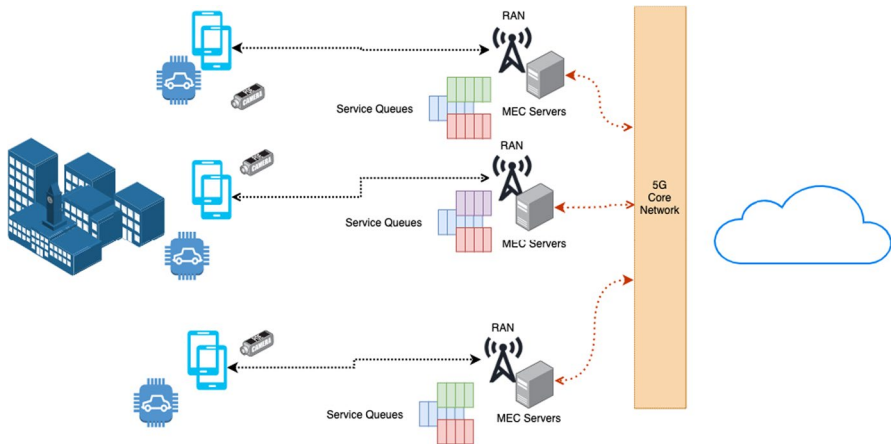


Fig. 2 Problem model. MEC servers have associate service queue that store service components to be processed. The different colors are to indicate distinct service component types

4.1 Problem Description

To define our MEC resource allocation problem we need:

- a set of service components C
- a set of offloading requests R
- a set of users U
- a set of MEC servers S

We suppose that users can offload several service components corresponding to different types of applications or micro-services. More specifically, $C = \{c_1, c_2, \dots, c_n\}$ is the set of service components that users can offload to MEC servers. Furthermore, each service component type c_i requires $c_i(res)$ amount of resources for processing. Without lack of generality, we assume that res represents the number of CPU cores assigned to the specific service component for processing.

Then, $S = \{s_1, s_2, \dots, s_n\}$ is the set of MEC servers available to compute the users' offloading requests. We consider that all MEC servers are accessible within a low latency range but we also assume that users in the close proximity of a MEC server can benefit of very limited latency, e.g., within 5ms range if they are covered by the same gNodeB [8]. Finally, MEC servers have associated computing resources for processing, thus enabling to model different types of servers and their computing capabilities. More specifically, a MEC server $s_j \in S$ is assigned with $s_{j,res}$ resources where res represents the number of CPU cores.

Therefore, at a given time t the amount of resources allocated for the processing of service components on a single MEC server must not exceed its capacity:

$$\begin{aligned} \sum_{i \in C} c_i(res) \times x_{i,j} &\leq s_{j,res} \quad \forall s_j \in S \\ \sum_{j \in S} x_{i,j} &\geq 1 \quad \forall c_i \in C \end{aligned} \quad (1)$$

where $x_{i,j}$ equals to 1 when a service component c_i is allocated for processing offloading requests on a MEC server s_j and to 0 otherwise.

Let us note that MEC resources are configured for a given time-window, which size depends on the specific scenario. In this work, we consider resource management as the act of allocating computing resources for a set of service components on each MEC server. More specifically, we consider that at a given time t each MEC server $s_j \in S$ is associated with a subset of service components $c_i \in C$ for which users can request the offloading, i.e., a MEC server can process only the offloading requests regarding the subset of allocated service components. Therefore, the resource management operations define a set of MEC servers on which it is possible to execute a certain type of service component $c_i \in C$. Finally, let us specify that a service component $c_i \in C$ would be allocated for processing on at least one MEC server—as specified in the second constraint of Eq. (1).

4.2 Value of Information (VoI)

Value-of-Information (VoI) is an interesting criterion for resource management optimization that we explored in previous works [5, 22]. VoI enables to assign a value to every single piece of information to quantify the utility that it can bring to its consumers. The concept of VoI was originally born from the seminal research by Howard in 1966 [51], which tried to extend Shannon's information theory to economics and decision sciences.

We believe that scenarios for 5G and Beyond applications can benefit from the adoption of VoI methodologies and tools to prioritize the processing of most important and mission-critical service components (from a MEC user perspective) by distributing the set of computing resources accordingly. More specifically, with VoI-based policies, we want to propose a set of methodologies and tools that would maximize the utility that a set of computing and network resources can provide to its users when these resources cannot entirely satisfy the users' requests. Therefore, VoI methodologies and tools can be beneficial in all those situations involving environments characterized by a limited amount of computing, network, and storage resources where there is the need to prioritize important and mission-critical services. Interested readers can refer to the work in [5] for a numerical example of the application of VoI methodologies and tools that shows how compelling VoI can be for addressing the computation of important information in resource constrained scenarios.

To contribute to MEC resource management, we express the VoI optimization problem as an objective function that measures the VoI generated by the processing

of offloaded requests $r \in R$ in a given time-window with a specific resources configuration.¹ We have:

$$f_n = \text{TOTAL}_{\text{Vol}}(t_n, t_{n+1}) = \sum_{r \in R(t_n, t_{n+1})} \text{Vol}_{\Theta}(r, c). \quad (2)$$

where R is set of offloading requests sent by users that need MEC computing capabilities for processing service components and c the service component associated to r . The processing of r is performed by MEC server $s_i \in S$ and generates a response $m_{r,c}$, which is returned to the user that requested the offloading.

To measure the $\text{Vol}_{\Theta}(r, c)$ we need to express different components: the initial VoI of a message, decay functions, and a utility function. First, we express the initial VoI of a message using the notation $\text{Vol}_0(r, c)$. We suppose that $\text{Vol}_0(r, c)$ is service component specific, e.g., a priority factor, and it is to measure the initial VoI of a request r .

Second, we consider a Proximity Relevance Decay (PRD) and a Time Relevance Decay (TRD) functions to take into account the decay a request is subjected to from its originating (location/time) to its processing (location/time) and back to its originating (location/time). Let us note that it is possible to specify multiple TRD definitions for modeling latency-sensitive and -tolerant service components [5, 52, 53]. For example, a latency-sensitive service component would likely have a strong VoI decay if not processed within a useful time, while the VoI of a latency-tolerant service component is less subjected to time-decay. In this work, we define a TRD and PRD function for each service component $c_i \in C$ by modeling these functions as decay multipliers with values in $[0, 1] \in \mathbb{R}$.

Then, we model the function $U(u_r, m_{r,c})$ to define the utility for a MEC user of type u_r in receiving a message (successful execution of the offloaded request) $m_{r,c}$, where c is the service component type.

We have that the VoI of a processed request can be calculated as follows:

$$\text{Vol}_{\Theta}(r, c) = \text{Vol}_0(r, c) \times \text{TRD}(m_{r,c}) \times \text{PRD}(m_{r,c}) \times U(u_r, m_{r,c}) \quad (3)$$

where $\text{Vol}_0(r, c)$ is the initial VoI of a request r considering a service component $c \in C$, r is the user's offloading request, and $m_{r,c}$ represents the result of the service component processing.

Let us specify that the VoI of a message is calculated in several steps, during which the VoI value associated with a message changes. For instance, when a user requires the offloading of a service component, the underlying framework generates a message with associated VoI metadata that describes the end-user utility, the generation time, the originating location, the initial VoI value, and the current VoI value. Then, when the offloading request arrives at the MEC server for processing, the associated VoI will change according to the definition of the PRD and TRD functions, which model the decay from the originating time/location to the delivery

¹ Since a request r is uniquely related to a service component, to simplify the notation we do not consider subscripts for service components in Sect. 4.2.

time/location. Afterward, the service component processing would add increased value to the message, thus further changing its current VoI value. When the user receives the service response, it will perform the last calculation step to include the additional decay from the processing time/location to the delivery time/location.

4.3 Computing and Communication Models

As for the computing model, we consider that offloaded requests are processed sequentially by modeling service queues that buffer incoming requests. At a given time t , $sq_s^i(t)$ defines the number of queued requests, where i represents the offloaded service component type, and s the MEC server associated with the service queue. We also assume that a service queue sq^i has an associated maximum capacity sq_{\max}^i that is equal for each service queue. In addition, we consider that service queues will start dropping offloaded requests as soon as $sq_s^i(t)$ reaches the maximum capacity. Finally, let us note that each service queue $sq_s^i(t)$, allocated on a MEC server, is associated with an amount of allocated res (CPU cores). Therefore, we assume that the processing of the different service queues can proceed in parallel.

With regard to the buffer processing, we assume that different policies can be adopted such as first-in first-out (FIFO) and VoI based policies. In the case of a FIFO policy, queued requests will wait for a time that is proportional to the service component processing time and the number of queued requests:

$$Q_{time}(r, c_i, t) = \text{Service_time}(c_i) \times sq_s^i(t) \quad (4)$$

where r is an offloading request, $\text{service_time}(c_i)$ the processing time for service component c_i , $sq_s^i(t)$ is the number of queued requests in the service queue processing the service component c_i on the MEC server $s_j \in S$.

Instead, using VoI queue management policies we have that the Q_{time} for an offloading request r' depends not only on the VoI value of the request ($\text{VoI}_0(r')$) but also from the VoI value of the other queued requests. Therefore, in the case of a VoI queue management, we model the Q_{time} for a request r' as follows:

$$Q_{time}(r', c_i, t) = \text{Service_time}(c_i) \times \sum_{r|\text{VoI}(r) \geq \text{VoI}(r')} sq_s^i(t) \quad (5)$$

where we suppose to have a sorted service queue, in which service components with higher VoI would be the first to be processed. Therefore, such priority-based mechanisms require the implementation of load-balancing and sorting dispatching at the MEC server level. Let us also note that service components with high VoI values will likely be prioritized if using (5) and they will have a shorter Q_{time} . On the other hand, this mechanism penalizes service components with low VoI values, thus causing possible starvation for low-priority service components.

Then, we model the links between users and MEC servers using a simplistic assumption for the MEC scenario illustrated in Fig. 1. More specifically, we consider that users can communicate with the MEC server in the proximity of the Base Station using at a very reduced latency by exploiting the RAN cell coverage, e.g.

a ultra-Reliable Low Latency Communications (uRLLC) network slice of the 5G network. On the other hand, users can also request the offloading of a service component to other MEC servers attached to the core network at the expense of greater communication latency. We express the communication time between users and MEC servers using the simplified notation:

$$T_{\text{comm}}(u, s) = \begin{cases} T^{\text{RAN}} & \text{if user } u \text{ resides within RAN with MEC server } s \\ T^{\text{RAN}} + T^{\text{CORE}} & \text{if user } u \text{ communicate with MEC server } s \text{ using the core network} \end{cases} \quad (6)$$

where T^{RAN} represents the communication time to access computing resources within the RAN where the user u resides, and T^{CORE} the communication time to access computing resources attached to the core network. Let us note that for off-loading a request to a MEC server in a different location, a user spend both T^{RAN} and T^{CORE} to access the MEC server $s_j \in S$ at a different gNodeB.

It is worth noting that in the case of successful execution of an offloaded request for the service component c_i , the user receives a message m containing the response of service component processing in time:

$$T_{\text{req}}(r, c_i, s_i) = 2 \times T_{\text{comm}} + Q_{\text{time}}(r, c_i, t) + \text{Service_time}(c_i) \quad (7)$$

where we assume $2 \times T_{\text{comm}}$ to send the request and to receive the response, the queue time to be calculated using either (4) or (5), plus the time for processing the request itself.

Let us further specify that $Q_{\text{time}}(r, c_i, t)$ in (7) depends on the buffer processing policy. More specifically, if the system processes the requests according to the VoI model presented in (5), it is expected that requests carrying higher VoI values would have lower queuing times, and consequently lower T_{req} times. Finally, to give readers a comprehensive summary of the system model notation, we present Table 2.

5 MECForge for Resource Management in MEC

Finding a MEC resource configuration that maximizes the value for (3) is a challenging task because it requires solving the resource management problem for different values of t , i.e. for consequent time-windows, thus transforming a static problem into a time-variant and dynamic problem. Let us note that if static optimization methods can be useful to maximize (3) for a specific time-window, they could be less effective when dealing with a dynamic environment, which presents significant differences between time-windows.

To overcome the limitations of static optimization methodologies, we investigate the adoption of an RL approach for MEC resource management. More specifically, we propose the adoption of DRL to train a software agent (resource orchestrator) through reward-maximization [18]. To exploit DRL for our specific problem, we choose the Deep Q-Network (DQN) algorithm as it demonstrated to be well applicable to a wide range of optimization problems in the related literature [4, 18, 28, 42].

Table 2 Summary of the used notation

Term	Meaning
C	Set of service components hosted in MEC servers
$c \in C$	A service component
r	A service request to a service component
$R(t_n, t_{n+1})$	Set of offloading requests within time window (t_n, t_{n+1})
U	Set of users
S	Set of MEC servers
$s_j \in S$	A MEC server s_j
$x_{i,j}$	The allocation of service component c_i on MEC server s_j
$m_{r,c}$	The result returned in response to the processing of request r by service component c
Θ	A system configuration
$VoI_{\Theta}(r, c)$	The VoI delivered by request r to service component c with configuration Θ
$VoI_0(r, c)$	The initial VoI of a service request r for service component c
u_i	A user type
PRD	Proximity Relevance Decay function
$PRD(m_{r,c})$	Ratio of VoI lost by response $m_{r,c}$ at its receipt due to proximity relevance decay
TRD	Timeliness Relevance Decay function
$TRD(m_{r,c})$	Ratio of VoI lost by response $m_{r,c}$ at its receipt due to timeliness relevance decay
$U(u_i, m_{r,c})$	Differential utility of $m_{r,c}$ for a user of type u_i
sq_s^i	Service queue at MEC Server s for processing offloaded service component of type i
$sq_s^i(t)$	Number of queued requests at time t
sq_{max}^i	Service queue maximum capacity
$Q_{time}(r, c_i, t)$	Queue time for a request r
$service_time(c_i)$	Service time for c_i
$T_{comm}(u, s)$	Communication time between user u and MEC server s
T^{RAN}	Communication Time to access the RAN network
T^{CORE}	Communication Time to access the core network
$T_{req}(c_i, s_j)$	Time to receive the response

This section provides a brief background on DQN and then describes the Markov Decision Process (MDP) for maximizing VoI in MEC resource management.

5.1 DQN

Classical RL algorithms require the exploration of the state’s space to learn a sequence of actions capable to maximize the agent’s reward. This requires evaluating a considerable amount of actions and to store the corresponding rewards in memory using tabular encoding functions [23].

A representative example of a simple RL approach is Q-Learning, which adopts a tabular encoding, called Q-Table, to store Q-value state action transactions. More specifically, we have that the learning process updates the Q-value function in an iterative way using the Bellman equation as follows [23]:

$$Q(S, a) = Q(S, a) + lr[R(S, a) + \gamma \max(Q'(S', a') - Q(S, a))]. \quad (8)$$

where lr is the learning rate, $R(S, a)$ the immediate reward for performing action a under state S , γ the discount rate for taking into account the expected future reward given by state S' and all consequent possible actions.

To overcome the limitations of classic RL approaches, DRL has emerged as a promising solution to deal with problems with large-scale states and action spaces [54]. Differently from RL, DRL exploits other machine learning methodologies such as neural networks to approximate the state, action transition values. More specifically, instead of using a tabular encoding to store transitions and corresponding rewards, DRL-based algorithms model the transition function using deep neural networks whose parameters and sizes depend on the complexity and size of the problem.

Within DRL, a widely used Q-learning algorithm is DQN, in which the prefix Deep is to introduce the adoption of neural-networks to learn a parameterized estimation of the Q-value function $Q(s, a, w)$ [54]. More specifically, DQN algorithms substitute the Q-table with a deep neural network for mapping states into action values. The goal of DQN algorithms is to minimize a loss function defined as the squared difference from the target and predicted value:

$$Loss = (R(S, a) + \gamma \max_{a'} Q(S', a'; w') - Q(S, a; w))^2 \quad (9)$$

where $R(S, a)$ the immediate reward for performing action a under state S , w and w' represent weight parameters (for the Q-network), $Q(S', a'; w')$ indicates the target value for the state S' , and $Q(s, a; w)$ is the current predicted value. During the training iterations, the DQN algorithm minimizes the Loss function using the gradient descent method for updating the weights parameters w :

$$w_{t+1} = w_t + lr[R(S, a) + \gamma \max(Q'(S', a'; w') - Q(S, a; w)) \nabla_w Q(S, a; w)]. \quad (10)$$

Moreover, to learn the parameterized estimation of the Q-value function, DQN exploits two different Q-value networks: a local Q-network and a target Q-network. During this process, the target network is updated after a configurable amount of iterations with the weights from the local Q-network to stabilize the training process. Finally, most recent DQN versions rely on a replay memory to store some state transitions for training. The algorithm randomly samples mini-batch transitions from the replay memory to update the neural network, thus reducing the correlation and enabling experience replay.

5.2 MDP Description

To solve the MEC resource management problem using an RL approach we define a Markov Decision Process (MDP) for the presented system model [55]. A MDP is a general framework (particularly suited for RL) to define decision making

problems [23]. Let us note that using proper modeling of states and actions is essential to solve a particular problem using RL. In fact, RL in general requires a good knowledge of the whole problem and a proper reward definition in order to allow the training process to converge.

The MDP for VoI allocation defines a set ST of states $\{S, S', S'', \dots\}$, a set A of actions that allow an agent to move from a state S to another state S' , and a Reward function R that defines the reward given by an action that moves the environment into a different state. In particular, $R_a(S, S')$ is the immediate reward for performing action $a \in A$ under state $S \in ST$. Finally, the goal of the MDP is to find the optimal policy $a = \pi(S)$ that gives the best action $a \in A$ under state $S \in ST$ that allows maximizing the Q-function $Q(S, a)$ for each state action pair.

With regard to a state $S \in ST$, we define it as a two-elements tuple:

$$S = (\theta_{RATIO}, CONF)$$

where θ_{RATIO} describes the processing ratio, i.e., the ratio of offloaded requests processed using a particular MEC resource configuration, and $CONF$ describes the configuration of MEC servers.

To model $CONF$ service configuration we adopt an array-like service configuration with binary values:

$$CONF = \{X_{c_1, s_1}, X_{c_2, s_1}, X_{c_3, s_1}, \dots, X_{c_{k-1}, s_{n-1}}, \dots, X_{c_k, s_n}\}$$

where the value of the element X_{c_i, s_j} describes if on MEC server s_j are allocated computing resources for processing service of type c_i , n is the number of MEC servers and k the $|C|$. For instance, $X_{c_1, s_3} = 1$ is an example of a resource configuration in $CONF$ that allocates on MEC server $s_3 \in S$ the computational resources c_1^r requested for processing offloading requests of type c_1 .

On the other hand, we define an action $a \in A$ as the binary decision 0, 1 to allocate or not computing resources for processing the service component $c_i \in C$ on MEC server $s_j \in S$, e.g. the setting of variable X_{c_i, s_j} . More specifically, at each step l the agent analyzes a X_{c_i, s_j} variable that describes the $CONF[l]$ element of a particular state $S \in ST$ and decides whether to allocate or not the computing resources for the service component c_i on MEC server s_j . It is worth noting that a particular action could result in an infeasible allocation that violates the resource capacity constraints defined in Eq. (1). When this happens, the action is invalidated by not allocating resources on the saturated MEC server.

With regard to the problem resolution, we model the MDP as a sequence of n discrete time steps $l = 0, 1, 2, \dots, i, \dots, n$, where $n = |S| \times |C|$, $|S|$ is the number of MEC servers, and $|C|$ represents the number of service component types. During each step, l the software agent analyzes the $CONF[l]$ element (where the l index corresponds to the l time step) and decides whatever or not allocates computing resources for the service component on the MEC server indicated by $CONF[l]$ value.

Table 3 Summary of the MDP notation

Term	Meaning
ST	Set of states
$S \in ST$	A particular state
A	Set of actions
$a \in A$	An action a in A
R	Reward
$R_a(S, S')$	Reward from performing action a under state S
$a = \pi(S)$	Optimal policy that selects action a under state S
$Q(S, a)$	Q-function for a state S and action a pair
Θ_{RATIO}	The processing ratio
$CONF$	MEC servers configuration
X_{c_i, s_j}	Allocation of service component c_i on MEC server s_j
l	A step l
$CONF[l]$	The l -th $CONF$ element
$sc_k^{rr}(t)$	Amount of resources requested by k -th service component at time t

Every time the agents performs an action $a \in A$ under state $S \in ST$, a new state S' is reached, and the agent gets an immediate reward $R_a(S, S')$. In this work, we model rewards using the following policy:

$$R_a(S, S') = \begin{cases} 1 + \Theta_{RATIO} & \text{if } S' \text{ improves the value of (2)} \\ 0.5 & \text{if } S' \text{ improves the } \Theta_{RATIO} \text{ but not the value of (2)} \\ 0 & \text{otherwise} \end{cases} \quad (11)$$

where we assign $1 + \Theta_{RATIO}$ to those actions that improve the generated VoI, 0.5 to those actions that do not improve the value for (2) but can increase the Θ_{RATIO} of the system, and 0 to all actions that do not improve the MEC resource configuration described at the previous step.

We designed this reward model to let MECForge learn how to allocate resources according to 2 criteria: total VoI maximization and processing ratio maximization. Towards that goal, Eq. 11 assigns higher rewards to the actions capable of improving the total amount of VoI delivered to users (according to Eq. 2), thus prioritizing the actions capable of maximizing the amount of VoI delivered to the end-users. On the other hand, our model also rewards those actions that can increase Θ_{RATIO} (with a limited reward value) and thus teaches MECForge to prefer configurations that deliver a high total VoI but also lead to a high number of requests served on MEC, with the result of higher utilization of MEC. Finally, readers can find a summary of the used notation and their meaning in Table 3.

5.3 MECForge Allocation Algorithm

To solve the described MDP, we present the MECForge algorithm, which implements a DQN based approach. MECForge aims to find the VoI optimal allocation for service components by learning the optimal policy $a = \pi(S)$ that selects the best actions to take under a particular state $S \in ST$ that lead to a VoI optimal MEC resource configuration in a finite number of steps N .

Algorithm 1 MECForge pseudo-algorithm and DQN

```

procedure DQN( $\epsilon, \gamma, \text{update\_step}$ )
     $Q_0(S, a) = 0$  initialize action-value function to 0
    initialize target action-value Q-network for Q estimation
    initialize replay memory  $RM$ 
    initialize a random allocation  $S_1$  to be used as initialization state
    calculate number of steps  $N$  as  $|C| \times |S|$ 
    for  $i$  in  $1, \text{max\_episodes}$  do
        initialize state  $S_1 = S_s$ 
        initialize  $er = 0$  episode reward
        initialize  $Vol(t) = 0$ 
        initialize  $\Theta_{RATIO}(t) = 0$ 
        for  $t$  in  $1, N$  do
            if  $\text{rand}() \geq \epsilon$  then
                select action  $a_t = \text{argmax}\{Q(S_t, a)\}$ 
            else
                select a random action  $a_t \in 0, 1$ 
            end if
            take action  $a_t$  and generate the next state  $S_{t+1}$ 
             $S_t, \text{reward}, Vol(t), \Theta_{RATIO}(t) = \text{MECFORGE}(CONF, S_{t+1}, Vol_t, \Theta_{RATIO}(t))$ 
            set  $S_t$  as the next state  $S_{t+1}$ 
            store transition in replay memory  $RM$ 
            update  $er$ 
            if  $t$  is an update step and there are enough transitions in memory then
                sample a random subset of transitions from  $RM$  and learn
            end if
            calculate discounted reward using  $\gamma$ 
        end for
        decrease  $\epsilon$  and make it decay
    end for
    return  $\pi$  a policy for mapping states into actions
end procedure

function MECFORGE( $CONF, S_{t+1}, Vol(t), \Theta_{RATIO}(t)$ )
     $Vol_{t+1}, \Theta_{RATIO}(t+1) = \text{simulate}(CONF, S_{t+1})$ 
    if  $Vol_{t+1} \geq Vol_t$  then
         $\text{reward} = 1 + \Theta_{RATIO}(t+1)$ 
    else if  $\Theta_{RATIO}(t+1) \geq \Theta_{RATIO}(t)$  then
         $\text{reward} = 0.5$ 
    else
         $\text{reward} = 0$ 
    end if
    return  $S_{t+1}, \text{reward}, Vol(t+1), \Theta_{RATIO}(t+1)$ 
end function

```

▷ apply (2) to calculate VoI and Θ_{RATIO}
 ▷ assign the reward r_t according to (11)

Algorithm 1 is to give readers a simplified illustration of the MECForge, which exploits a DQN algorithm with experience replay. Algorithm 1 takes as input the values for ϵ (the exploration rate), γ (the discount rate), and the update_step parameters used during the training of the Q-network. The replay memory RM is used to store the transitions and for sampling a mini-batch during the training with experience replay [50].

As depicted in Algorithm 1, we define the allocation problem as an episodic task with a maximum number of episodes $max_episode$. Let us specify that MECForge defines an episode as a finite number of steps N , after which the episode is considered finished. At the beginning of a new episode, the initial state S_1 is set to the random state S_* generated in the initialization phase of the algorithm. This is a common practice when defining a DRL training process, however, different policies can be chosen. In addition, we set the values of $\Theta_{RATIO}(t)$ and $VoI(t)$ to 0, where t indicates the current step. This is to implement the reward assignment policy described in (11).

During each step t , the agent interacts with the environment by selecting to allocate or not resources for the given service component on the given MEC server identified by $CONF[t]$, thus generating a new state S_{t+1} . Then, MECForge simulates the new configuration S_{t+1} to calculate the total amount of delivered (VoI($t + 1$)), using (2), and the $\Theta_{RATIO}(t + 1)$ value which are required for the reward calculation. During these steps, the goal of the agent is to maximize the cumulative reward that it collects over the episode in a given amount of iterations.

5.4 Continuous Optimization Framework

To illustrate to readers how we intend to realize continuous reconfiguration of MEC resources we present a proof-of-concept framework leveraging MECForge and Digital Twin methods. A Digital Twin is a virtual replica of a real-world object on which experimenting changes and (re)configurations, without altering the real object. Digital Twin approaches are becoming more popular even in the networking area to tame the configuration complexity of new-generation networks and they represent an interesting trade-off for evaluating resource management strategies without risking misconfiguration and faults of systems running in a production environment [56, 57]. Similar approaches have been studied in related literature, such as the one in [58], which exploits a simulation approach to validate a ML generated configuration before applying it to the real network.

Figure 3 shows the envisioned framework's architecture for allowing continuous optimization of MEC resources running in a 5G and Beyond network. More specifically, the framework includes two main blocks: an optimizer and a Resource Monitoring block.

First, the Resource Monitoring block is responsible for collecting metrics and information regarding the behavior of the observed system, including users' interests/utility, available MEC servers, and network conditions (latency, bandwidth, and so on). The Resource Monitoring block uses the collected metrics to update the configuration of the Digital Twin of the system. We implemented the Digital Twin using a discrete-event simulator for MEC scenarios that we built by leveraging the experience of the Phileas research project [59]. The MEC simulator enables reenacting the execution of offloading requests in a MEC scenario using a particular MEC resource configuration, thus allowing us to estimate the amount of delivered VoI. Let us note that the simulator requires an input configuration that describes the scenario-specific settings such as the number of devices, service component types, user distribution,

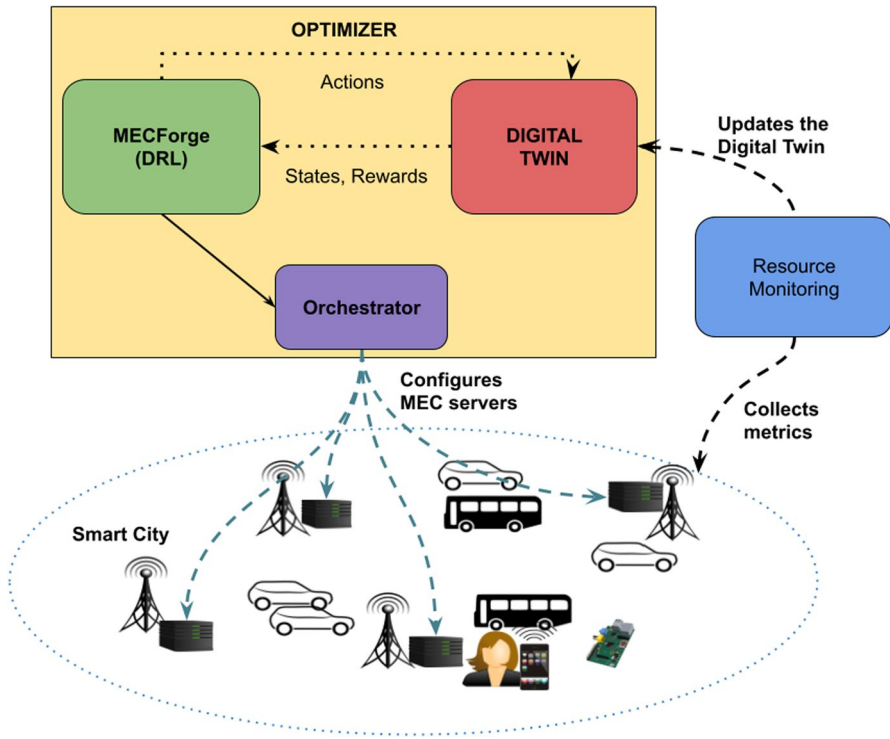


Fig. 3 An overview of the management framework for MEC in 5G and Beyond that exploits a Digital Twin of a real-world scenario (a Smart City) on which MECForge is executed to find the best configurations for MEC resources

and so on. This configuration can be defined by the simulator's users or created in an automated matter as described above.

Second, the Optimizer is responsible to learn the best configuration for MEC resources by interacting with the Digital Twin. More specifically, MECForge interacts with the Digital Twin to find the MEC resource configuration that maximizes the value of (2). To this end, the Optimizer runs MECForge for a finite number of episodes during which it measures the quality of different configurations and it keeps track of the one that maximizes the value of (2). When the Optimizer finds a suitable allocation, the orchestrator component configures the resources at the edge of the network by interacting directly with the MEC servers.

To enable continuous optimization we envision this loop to run continuously, thus ensuring MECForge to exploit both the experience built by interacting with the Digital Twin during previous time-windows but also to keep experiencing with different environmental conditions, which may require a re-configuration of resources (mobility of users, system's load, etc...). We assume that the number of re-configurations depends

on the specific scenario, e.g., two or three re-configurations for relatively static scenarios. Finally, let us specify that the optimizer component can run on a dedicated edge computing node or on cloud computing resources.

6 Evaluation

To verify the capabilities of the proposed framework, we first define an initial scenario for Smart City applications on which we evaluate the DRL convergence, i.e. the capability of MECForge to learn a good rewarding function that solves the MEC resource management problem. Furthermore, we evaluate the VoI management policies under different configurations and we compare it with greedy approaches.

Then, we devised a second scenario in which we changed the distribution of users among the locations of interest, and investigated if MECForge was still capable of distributing offloading requests to MEC servers in a way that maximizes the total VoI without having to re-train the agent.

6.1 MEC Scenario for Smart Cities

To verify the capabilities of the proposed solution we envision a Smart City of the near future where citizens are connected to a 5G and Beyond network that provides communication and computing capabilities at the edge of the network. In this scenario, the citizen can offload the processing of service components to MEC servers nearby to save the battery life of their UEs and to benefit from the reduced processed latency that MEC servers can provide. We envision that MEC servers are installed by one or multiple providers at Base Stations (BS) or other equipment in the proximity of the edge cellular network, in a configuration similar to the one illustrated in Fig. 1.

More specifically, the scenario contains the description of 12 MEC Servers and 4 service components for which users can request the offloading. We model communication latency between UEs and MEC servers according to (6) to specify different latency values for users to access MEC servers inside and outside their RAN. In the illustrated scenario, users can communicate with the MEC server in their RAN with a reduced latency range, while other MEC servers connected to the core networks are available at the expense of a greater latency range. This would make the exploitation of the local MEC server more convenient than the distant ones. However, offloaded requests need to be distributed accordingly to avoid processing bottlenecks and a dropping of processed requests.

With regard to the processing model, we assume that there is a service queue with a maximum capacity of 50 for each service component ($c \in C$) configured at a MEC server ($s \in S$); service queues will start dropping incoming requests as soon as the buffer is full. In addition, we model processing times of simulated service components on MEC servers using a random variable with exponential distribution. To

Table 4 Summary of the scenario configuration

Parameter	Description	Value
Service_time(c_1)	Processing time (Exponential Distribution)	$\lambda = 1/0.250$
Service_time(c_2)	Processing time (Exponential Distribution)	$\lambda = 1/0.300$
Service_time(c_3)	Processing time (Exponential Distribution)	$\lambda = 1/0.200$
Service_time(c_4)	Processing time (Exponential Distribution)	$\lambda = 1/0.250$
TRD for (c_1, c_2, c_3, c_4)	Exponential Decay Function	(Half-life) 1000 ms
PRD for (c_1, c_2, c_3, c_4)	Linear Decay Function	(Half-life) 1 km
$T_{comm}(RAN)$	T^{RAN} end-to-end latency (normal distribution)	$\mu = 5ms, \sigma = 2ms$
$T_{comm}(RAN)$	T^{CORE} end-to-end latency (normal distribution)	$\mu = 15ms, \sigma = 5ms$
$U(v, c_1)$	Users utility in service component c_1	1.65
$U(v, c_2)$	Users utility in service component c_2	1.3
$U(v, c_3)$	Users utility in service component c_3	1.45
$U(v, c_4)$	Users utility in service component c_3	1.80

this end, it is worth considering that we modeled service queues to process offloading requests sequentially, i.e., one request at a time.

On the other hand, we model offloading requests by reenacting sets of users that generate, at different time slots, offloading requests with an initial VoI value -- $VoI_0(r, c)$ in Eq. (3). This would simulate the generation of approximately 2300 offloading requests that need to be processed on the available MEC servers. More specifically, we adopt the following distribution for the 2300 offloading requests: 35% for c_1 , 23% for c_2 , 28% for c_3 , and 14% for c_4 . Each of these requests is to offload the processing of a service component $c \in C$. Moreover, we assume to not have user mobility during the simulation time. Users would be equally distributed in different locations, which correspond to positions in the proximity of MEC servers.

To give readers a comprehensive illustration of the configuration parameters, Table 4 depicts the description and the value of the most important parameters we configured for the MEC scenario. Delving into details, we simulate the generation of offloading requests for four different service components using random variables with exponential distribution. To model communication latency from users to MEC servers and vice versa we use two random variables with normal distribution to represent a communication time for accessing resources within the RAN and another for resources attached to the core network according to other similar efforts [48, 60].

For modeling the user utility function ($U(u_t, m_{r,c})$) described in Eq. (3), we adopt a simplified approach where we have a single user type $u_t = v$, which gains an utility for the successful execution of an offloading request. More specifically, we define for each service component type $c \in C$ a multiplier in the [1, 2] range. This multiplier is described as $U(u_t, m_{r,c})$, where c represents the service component type. Let us note that $U(u_t, m_{r,c}) = 1$ corresponds to an the empty multiplier, while $U(u_t, m_{r,c}) = 2$ is equivalent to a 100% increment. Finally, considering that a request r is uniquely related to a service component type $c \in C$, we simplify the notation $m_{r,c}$ to consider only the service component type. Therefore, to describe the utility for a user of type

Table 5 Configuration Parameter for the Q-network

Parameter	Value
Number of hidden layers	2
γ	0.95
lr	0.0005
ϵ_{start}	1
ϵ_{end}	0.01
Mini-batch size	64

v and a service component $c = c_1$, we adopt the notation $U(v, c_1)$, as illustrated in Table 4.

These experiments aim to verify the capabilities of MECForge in learning a policy that maximizes the value of (2), which indicates the total VoI generated by the processing of offloaded service components during a fixed time window. More specifically, MECForge will learn a reward-maximizing policy that configures computing resources for the processing of service components on the available MEC servers using the total VoI generated as feedback.

6.2 MECForge Configuration Parameters

MECForge is a Deep Q-Learning Network (DQN) algorithm that we implemented in the Python programming language to solve the VoI allocation problem. We leveraged the PyTorch library² for implementing the training of the Deep Q-network (neural-network) responsible for mapping states in action values. We chose Python as a programming language because it is a valuable tool for implementing machine learning and data analysis tasks. In addition, the PyTorch library provides a user-friendly API for implementing the state-of-the-art machine learning models and optimization algorithms.

The Q-Network is implemented as a two-layers neural network with 64 nodes for each hidden layer, a $\gamma = 0.95$ the discount rate to determine the present value of future rewards, a learning rate of $lr = 0.0005$, and ϵ value (the exploration rate) starting from 1 and annealing to 0.01. Finally, for experience replay, we set the mini-batch size to 64. To summarize the configuration, Table 5 shows the value for each configuration parameter. We experimentally validated these values to be adequate for the resolution of the scenario we will present in the following Section.

During the training phase, MECForge interacts with the MEC simulator (the Digital Twin of the scenario) by means of an HTTP REST interface, thus allowing the two different software modules to interact. More specifically, given a state S representing a MEC resource configuration, MECForge sends a request (via HTTP) to the MEC simulator for calculating the value of (2) for the given configuration. Even

² <https://pytorch.org>.

if performance-wise is not great, this is a common practice to integrate software components written in different programming languages. We plan to design a better integration of MECForge in future versions of this work to speed up the learning process.

With regard to the training configuration, we choose a number of episodes of 1000 to give the agent a fair amount of iterations to learn an optimal policy, i.e. the optimal value for $Q(S, a)$. More specifically, each episode is defined as a sequence of $|S|x|C| = 48$ steps, where $|S| = 12$ and $|C| = 4$. The number of steps in an episode corresponds to the length of the configuration array that tracks which service component types are allocated on each MEC server, which in turn emerges from the fact that we consider 12 servers and 4 service component types.

As described in Algorithm 1, the evaluation of each step requires a simulation run to estimate the value of (2). For this purpose, we configured the simulator to reenact the training scenario for a limited time-window of one minute. It is worth specifying that the computation of the one-minute simulation requires roughly 0.5 seconds and that we selected this value for reducing the Q-network training time, which can be estimated in $1000 \times 48 \times 0.5 = 24000$ seconds.

With regard to the starting state, we select a random resource configuration for service components on MEC servers. Let us note that the random starting state generation could result in an infeasible service components configuration, which would not be considered by MECForge. Other options such as a greedy to calculate a feasible starting state are possible. However, opting for a random state is a good assumption to verify if given a fair amount of training iterations, the agent can learn a sequence of actions leading to a high-value VoI state.

6.3 Results

We configure MECForge using the parameters discussed in Sect. 6.2 to learn the optimal policy $a = \pi(s)$ that would maximize the total VoI (2) on the scenario described in Sect. 6.1. At the beginning of each episode, MECForge initializes the initial state using a randomly generated state. Then, at each step, it learns the actions that maximize the cumulative reward of the episode. During the evaluation, for each episode, we collect the score, i.e. the sum of rewards, and the state $s_{optimal}$, which leads to the best value of (2).

Finally, we would like to specify that the following results are divided into two different parts. First, in Sect. 6.3.1 we present the training and validation of MECForge on the scenario described in Sect. 6.1. Then, in Sect. 6.3.2 we test the performance of MECForge on a modified version of the training scenario, which is characterized by a concentrated request distribution.

6.3.1 Training and Validation with Evenly Distributed Requests

Figure 4a is to report the learning phase of MECForge in maximizing the value of (2). More specifically, Fig. 4a depicts the time series of the cumulative rewards that MECForge achieved during the 1000 episodes of the learning process. The

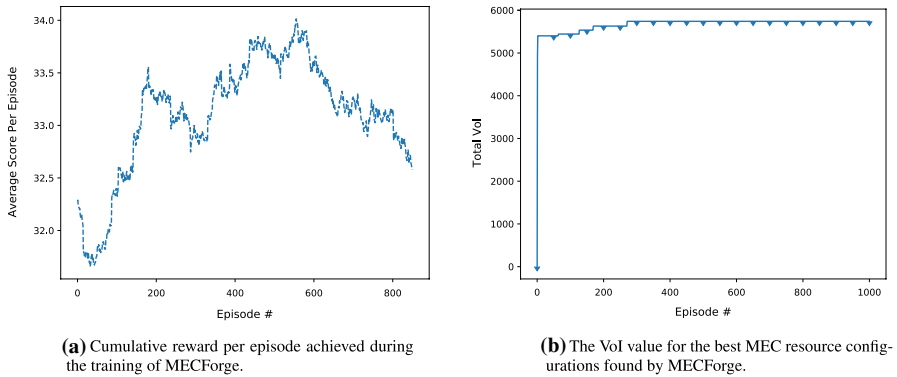


Fig. 4 The results of the MECForge training over 1000 episodes

reported trend is initially increasing, thus indicating that MECForge can learn a good rewarding policy in a relatively low amount of episodes for configuring resources accordingly. We believe this to be an encouraging result, which demonstrates the viability of DRL methodologies for our particular VoI management framework. Finally, we would like to note that for each episode, the algorithm generates an initial random allocation, thus adopting a more-explorative-than-exploitative approach that enables a broader states' space exploration at the price of reducing the cumulative reward per episode. This is also one of the reasons for which Fig. 4a does not report an ever-increasing trend.

On the other hand, Fig. 4b illustrates the best VoI values for (2) achieved during the training of MECForge. It is worth noting how the best value is achieved around the 600-th episode, thus indicating that MECForge is capable of improving the value of (2) in a relatively limited number of iterations. This also shows that MECForge can find a good rewarding policy that improves the allocation described by the random starting state S_{*} , thus demonstrating to be capable of improving the system's state (VoI generated, Θ_{RATIO} , Q_{time}) independently from the starting state.

To compare MECForge with other approaches in the training scenario (evenly distributed), we devised two other solutions with which we compare MECForge: a RANDOM approach and a GREEDY approach, which allocates all offloading requests to the closest MEC server (in terms of geographical distance).

On the one hand, RANDOM selects a random MEC resource configuration to distribute the amount of processing load between MEC servers. Let us note that an approach that assigns computing resources for the requested service types is not completely trivial, given that i) the same service component can be replicated on multiple MEC servers and ii) all MEC servers can be accessed within a reduced latency range. To collect the result for RANDOM, we simulated 1000 random MEC resource configurations from which we extract the best one in terms of generated VoI (2).

On the other hand, GREEDY adopts a proximity-processing policy for the processing of offloading requests by selecting the MEC server closest to the user that

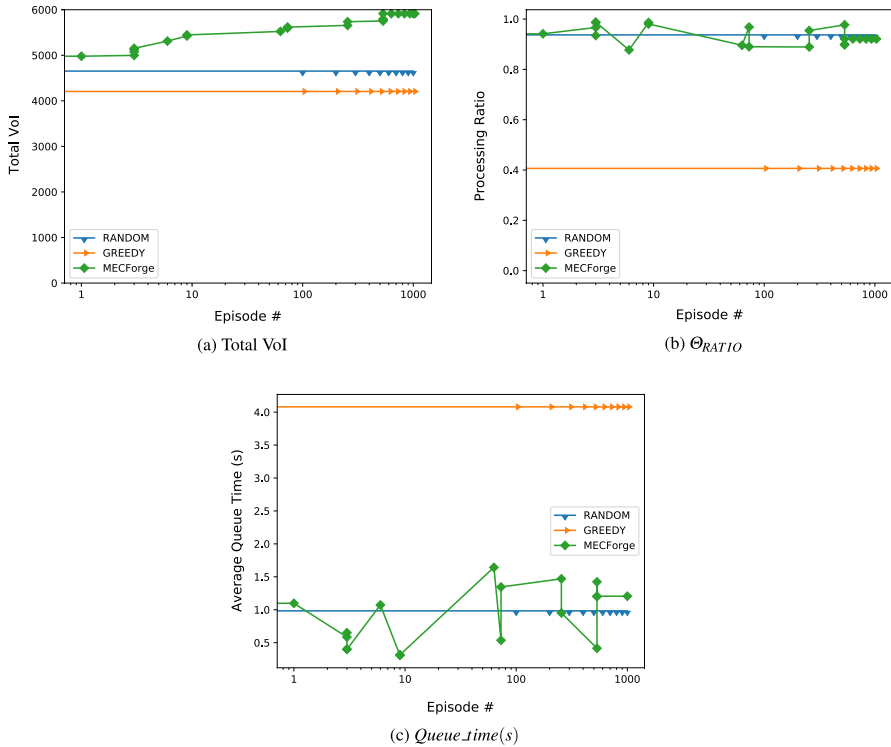


Fig. 5 The best MEC resource configurations found during the MECForge training compared with RANDOM and GREEDY

generated the offloaded request. Therefore, GREEDY tries to reduce the overall latency and the consequent VoI decay of messages by selecting configurations that reduce the transfer time $T_{comm}(u, s)$. This can be a suitable choice when users need to offload the processing of latency-sensitive service components and would like to receive the processed response in the shortest time possible. Finally, let us also note that both RANDOM and GREEDY could be interesting solutions to generate a starting state for MECForge.

We reported the comparison of the best MEC resource configuration between MECForge, RANDOM, and GREEDY in Fig. 5a to c using a logarithmic scale. More specifically, Fig. 5a illustrates the best configuration in terms of delivered VoI values found during the training of MECForge along with the two configurations found using RANDOM and GREEDY. When compared to RANDOM and GREEDY, MECForge finds the best configuration in terms of delivered VoI (5914.31). Let us note that RANDOM finds a lower VoI MEC resource configuration (4563.21), which is greater than the one found in the local policy implemented by GREEDY (4206.09) that seems to poorly perform in the evaluation scenario. These results confirm that MECForge can find high VoI MEC resource configurations, thus demonstrating the effectiveness of the presented approach

in maximizing the total amount of VoI delivered to the end-users. Let us also specify that it is conceivable that RANDOM and GREEDY find lower VoI MEC resource configurations because they implement different policies rather than VoI maximization.

Figure 5b shows the processing ratio (Θ_{RATIO}) associated with the best configurations (in terms of VoI) illustrated in Fig. 5a. More specifically, Fig. 5b shows that the results of RANDOM and MECForge are very similar, thus indicating that these configurations can deliver higher processing ratios. Let us also note that MECForge finds the maximum value for Θ_{RATIO} around the 600-th episode but it is associated with a MEC resource configuration delivering a lower amount of VoI. In fact, if we compare Fig. 5b with Fig. 5a we notice that higher processing ratios do not implicitly result in higher VoI resource configurations. Therefore, prioritizing the processing of offloaded requests with higher VoI can maximize the value of (2) delivered to end-users but can also cause possible starvation of requests with lower VoI values. Let us note that starvation for low VoI requests is possible only when the computing resources available at MEC servers cannot meet the users' demand. Therefore, we believe this to be a suitable trade-off considering that we propose VoI as a solution to deal with the processing of the most valuable requests (mission-critical services) in case of limited computing resources.

On the other hand, the GREEDY result shows that a local processing policy is not feasible for the evaluation scenario. In fact, selecting the closest MEC server to process users' offloaded requests will likely result in processing bottlenecks and request dropping. Therefore, this demonstrates that there is the need to configure MEC resources accordingly and to distribute requests among all available MEC servers.

Another result is the one illustrated in Fig. 5c, which depicts the average queue-time associated with the best configurations found by MECForge during the 1000 training episodes and the results of RANDOM and GREEDY. We calculated the average queue time as the ratio between the overall time spent by served requests within the service queues and the number of served requests. As supposed earlier, Fig. 5c shows that the GREEDY local processing policy saturates the service queues. In fact, GREEDY has the higher average queue time for processing only 964 requests. On the other hand, both RANDOM and MECForge show better results, thus demonstrating better coordination of resources. This is a result of notable importance, which demonstrates how GREEDY alone is not feasible for dealing with latency-sensitive services. More specifically, the choice to associate offloading requests to the closest MEC server might be beneficial for minimizing the communication latency but it does not guarantee having low processing/queue time. Therefore, this makes GREEDY infeasible for latency-sensitive and mission-critical services.

6.3.2 Evaluation in Modified Scenario with Concentrated Request Distribution

To further validate the proposed approach, we test MECForge on a modified scenario, without retraining the software agent. More specifically, we devised a modified scenario in which users have a different geographical distribution. Instead of

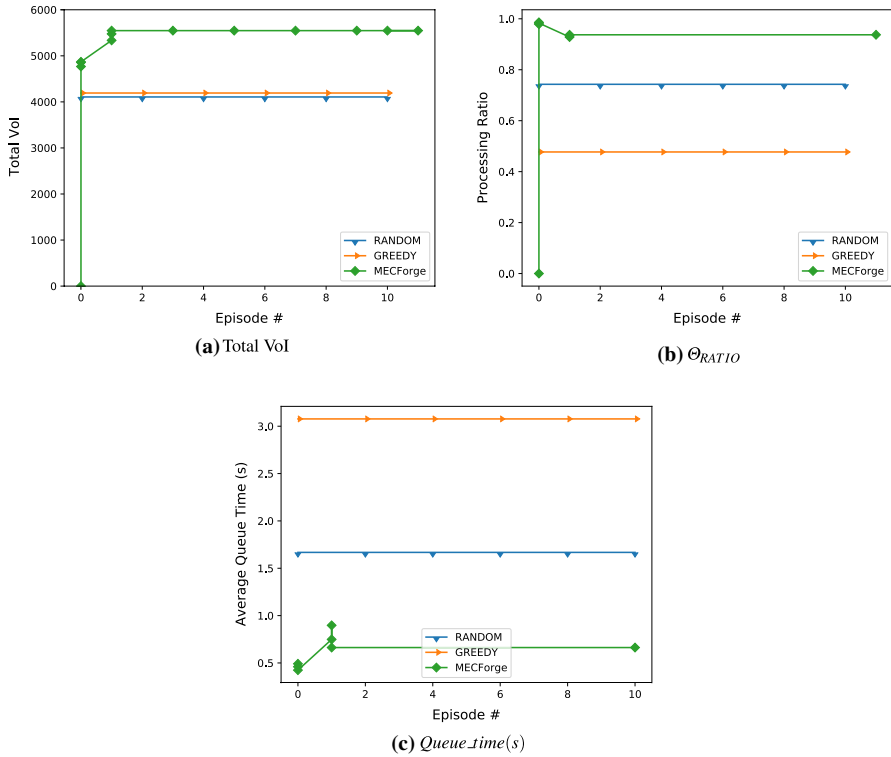


Fig. 6 The best MEC resource configurations for the scenario under modified conditions

considering users to be evenly distributed among the 12 MEC servers locations (as we did for the training scenario), we move a consistent part of the users (~ 80%) into a single MEC server location. We then investigated it to verify if MECForge can find a suitable configuration for MEC computing resources using the experience built upon the previous scenario.

To devise this “concentrated” scenario, we consider that a particular event, such as an outdoor concert, is taking place in a specific location of the Smart City. Therefore, we can imagine that attracted by this event, a consistent part of the users move to this specific location, which we supposed to be in a single RAN coverage.

Let us note that for this evaluation we keep the number of offloading requests consistent with the previous scenario (~2300) on which we trained MECForge. However, due to changed conditions, about 80% of the offloading requests will be generated in the same location, thus requiring a re-configuration of MEC resources to address the different requests’ distribution.

For this validation, we configure MECForge with the Q-network trained at the previous step without running another training procedure. More specifically, we run MECForge for 10 episodes during which we keep track of the best

configurations from which we report the Total VoI, Θ_{RATIO} , and the queue-time. Then, we also execute RANDOM and GREEDY on the “concentrated” scenario to collect their configurations.

We report the result of the validation in Fig. 6, which illustrates that in the “concentrated” scenario MECForge outperforms the other approaches. More specifically, Fig. 6a shows that MECForge finds the configuration that delivers the greater amount of VoI to end-users, while RANDOM and GREEDY find similar but lower solutions. As for the processing-ratio, MECForge achieves the best result when compared to the solutions found by RANDOM and GREEDY.

Let us note that even in this case, the configuration (found by MECForge) with the higher processing-ratio is not the one resulting in the higher VoI. Finally, the average queue-time results confirm the validity of MECForge that outperforms RANDOM and GREEDY, which instead report higher average queue-times about 1.5s for RANDOM and 3.0s for GREEDY.

Finally, it is worth noting that with the adoption of VoI based policies, the processing of most important offloading requests will be prioritized. Therefore, mission-critical and latency-sensitive services would benefit from the adoption VoI methodologies at the expenses of lower VoI services. This is a reasonable choice when dealing with mission-critical service in a resource constrained environment.

7 Conclusion and Future Works

This work presented our efforts for integrating DRL techniques into a VoI management framework for MEC computing resources configuration. Starting from the experience of previous research projects defined in [5, 22], we investigated DRL as an interesting reinforcement learning approach to configure MEC computing resources (CPU cores at MEC servers) for maximizing the amount of VoI delivered to end-users requesting service components offloading. To this end, we defined a system model and an MDP on which a DRL agent can learn a reward-maximization function that allocates computing resources on MEC servers.

Then, we present MECForge, a DQN-based approach that we specifically implemented to consider VoI policies for resource management problems in MEC. MECForge is to learn a resource management function to allocate a set of computing resources for the processing of service components on MEC servers that maximizes the total VoI delivered to the end-users that requested service components offloading. Furthermore, we presented an architecture for a Continuous Optimization Framework that leverages a Digital Twin (a virtual replica) of a real-world MEC deployment to safely test different resource configurations.

To evaluate the capabilities of MECForge, we devised a training scenario for MEC in Smart City on which MECForge proved to be efficient in finding a good-rewarding allocation policy and high VoI resource configurations in a relatively limited number of training iterations. Moreover, we compared MECForge with two other approaches to validate the achieved results both on the training scenario and on a “concentrated” scenario in which most of the users, and 80% of the service

requests, group in a single location, as opposed to the even distribution adopted for the training scenario.

Finally, let us note that even if RL approaches suffer from the curse of high training time, they provide an interesting and promising methodology for implementing an effective continuous optimization of resources in new generation networks. As future works, we intend to investigate different RL algorithms and to test the capabilities of MECForge on other scenarios. Furthermore, we intend to investigate the application of VoI methodologies and tools to network slicing and other network management problems.

Funding Open access funding provided by Università degli Studi di Ferrara within the CRUI-CARE Agreement.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Vega, M.T., Liaskos, C., Abadal, S., Papapetrou, E., Jain, A., Mouhouche, B., Kalem, G., Ergüt, S., Mach, M., Sabol, T., Cabellos-Aparicio, A., Grimm, C., Turck, F.D., Famaey, J.: Immersive interconnected virtual and augmented reality: a 5G and IoT perspective. *J. Netw. Syst. Manag.* **28**(4), 796–826 (2020). <https://doi.org/10.1007/s10922-020-09545-w>
2. Pham, Q.V., Fang, F., Ha, V.N., Piran, M.J., Le, M., Le, L.B., Hwang, W.J., Ding, Z.: A survey of multi-access edge computing in 5G and beyond: fundamentals, technology integration, and state-of-the-art. *IEEE Access* **8**, 116974–117017 (2020). <https://doi.org/10.1109/ACCESS.2020.3001277>
3. Yousefpour, A., Fung, C., Nguyen, T., Kadiyala, K., Jalali, F., Niakanlahiji, A., Kong, J., Jue, J.P.: All one needs to know about fog computing and related edge computing paradigms: a complete survey. *J. Syst. Architect.* **98**, 289–330 (2019). <https://doi.org/10.1016/j.sysarc.2019.02.009>
4. Chen, X., Zhang, H., Wu, C., Mao, S., Ji, Y., Bennis, M.: Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. *IEEE Internet Things J.* **6**(3), 4005–4018 (2019). <https://doi.org/10.1109/JIOT.2018.2876279>
5. Poltronieri, F., Tortonesi, M., Morelli, A., Stefanelli, C., Suri, N.: A value-of-information-based management framework for fog services. *Int. J. Netw. Manag.* **32**(1), e2156 (2022). <https://doi.org/10.1002/nem.2156>
6. Chiha, A., der Wee, M.V., Colle, D., Verbrugge, S.: Network slicing cost allocation model. *J. Netw. Syst. Manag.* **28**(3), 627–659 (2020). <https://doi.org/10.1007/s10922-020-09522-3>

7. Khan, S., Khan, S., Ali, Y., Khalid, M., Ullah, Z., Mumtaz, S.: Highly accurate and reliable wireless network slicing in 5th generation networks: a hybrid deep learning approach. *J. Netw. Syst. Manag.* (2022). <https://doi.org/10.1007/s10922-021-09636-2>
8. Zhao, Y., Zhai, W., Zhao, J., Zhang, T., Sun, S., Niyato, D., Lam, K.Y.: *A Comprehensive Survey of 6G Wireless Communications* (2021)
9. Camelo, M., Mennes, R., Shahid, A., Struye, J., Donato, C., Jabandzic, I., Giannoulis, S., Mahfoudhi, F., Maddala, P., Seskar, I., Moerman, I., Latre, S.: An AI-based incumbent protection system for collaborative intelligent radio networks. *IEEE Wirel. Commun.* **27**(5), 16–23 (2020). <https://doi.org/10.1109/MWC.001.2000032>
10. Zhani, M.F., ElBakoury, H.: FlexNGIA: a flexible internet architecture for the next-generation tactile internet. *J. Netw. Syst. Manag.* **28**(4), 751–795 (2020). <https://doi.org/10.1007/s10922-020-09525-0>
11. Bagaa, M., Taleb, T., Riekkki, J., Song, J.: Collaborative cross system AI: toward 5G system and beyond. *IEEE Netw.* **35**(4), 286–294 (2021). <https://doi.org/10.1109/MNET.011.2000607>
12. Moradi, M., Ahmadi, M., Nikbazm, R.: Comparison of machine learning techniques for VNF resource requirements prediction in NFV. *J. Netw. Syst. Manag.* (2022). <https://doi.org/10.1007/s10922-021-09629-1>
13. Abusubaih, M.: Intelligent wireless networks: challenges and future research topics. *J. Netw. Syst. Manag.* (2022). <https://doi.org/10.1007/s10922-021-09625-5>
14. Goethals, T., Turck, F.D., Volckaert, B.: Self-organizing fog support services for responsive edge computing. *J. Netw. Syst. Manag.* (2021). <https://doi.org/10.1007/s10922-020-09581-6>
15. Arulkumaran, K., Deisenroth, M.P., Brundage, M., Bharath, A.A.: *A Brief Survey of Deep Reinforcement Learning*. arXiv preprint [arXiv:1708.05866](https://arxiv.org/abs/1708.05866) (2017)
16. Mennes, R., De Figueiredo, F., Latré, S.: Multi-agent deep learning for multi-channel access in slotted wireless networks. *IEEE Access* **8**, 95032–95045 (2020). <https://doi.org/10.1109/ACCESS.2020.2995456>
17. Camelo, M., Claeys, M., Latre, S.: Parallel reinforcement learning with minimal communication overhead for IoT environments. *IEEE Internet Things J.* **7**(2), 1387–1400 (2020). <https://doi.org/10.1109/JIOT.2019.2955035>
18. Silver, D., Singh, S., Precup, D., Sutton, R.S.: Reward is enough. *Artif. Intell.* **299**, 103535 (2021). <https://doi.org/10.1016/j.artint.2021.103535>
19. Suri, N., Benincasa, G., Lenzi, R., Tortonesi, M., Stefanelli, C., Sadler, L.: Exploring value-of-information-based approaches to support effective communications in tactical networks. *IEEE Commun. Mag.* **53**(10), 39–45 (2015). <https://doi.org/10.1109/MCOM.2015.7295461>
20. Bharti, S., Pattanaik, K.K., Bellavista, P.: Value of information based sensor ranking for efficient sensor service allocation in service oriented wireless sensor networks. *IEEE Trans. Emerg. Topics Comput.* (2019). <https://doi.org/10.1109/TETC.2019.2891716>
21. Poltronieri, F., Tortonesi, M., Morelli, A., Stefanelli, C., Suri, N.: Value of Information based Optimal Service Fabric Management for Fog Computing. In: *NOMS 2020–2020 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–9 (2020)
22. Poltronieri, F., Tortonesi, M., Stefanelli, C., Suri, N.: Reinforcement Learning for value-based Placement of Fog Services. In: *IM 2021 IFIP/IEEE International Symposium on Integrated Network Management*, pp. 1–7. IFIP/IEEE (2021)
23. Sutton, R.S., Barto, A.G.: *Reinforcement Learning: An Introduction*, 2nd edn. Bradford Books, West Yorkshire (2018)
24. Quang, P.T.A., Hadjadj-Aoul, Y., Outtagarts, A.: A deep reinforcement learning approach for VNF forwarding graph embedding. *IEEE Trans. Netw. Serv. Manag.* **16**(4), 1318–1331 (2019). <https://doi.org/10.1109/TNSM.2019.2947905>
25. Yao, H., Ma, S., Wang, J., Zhang, P., Jiang, C., Guo, S.: A continuous-decision virtual network embedding scheme relying on reinforcement learning. *IEEE Trans. Netw. Serv. Manag.* **17**(2), 864–875 (2020). <https://doi.org/10.1109/TNSM.2020.2971543>
26. Kaur, A., Kumar, K.: Energy-efficient resource allocation in cognitive radio networks under cooperative multi-agent model-free reinforcement learning schemes. *IEEE Trans. Netw. Service Manag.* **17**(3), 1337–1348 (2020). <https://doi.org/10.1109/TNSM.2020.3000274>
27. Mai, L., Dao, N.N., Park, M.: Real-time task assignment approach leveraging reinforcement learning with evolution strategies for long-term latency minimization in fog computing. *Sensors* **18**, 2830 (2018). <https://doi.org/10.3390/s18092830>
28. Wei, F., Feng, G., Sun, Y., Wang, Y., Liang, Y.C.: Dynamic Network Slice Reconfiguration by Exploiting Deep Reinforcement Learning (2020). <https://doi.org/10.1109/ICC40277.2020.9148848>

29. De Schepper, T., Camelo, M., Famaey, J., Latré, S.: Traffic classification at the radio spectrum level using deep learning models trained with synthetic data. *Int. J. Netw. Manag.* **30**(4), e2100 (2020). <https://doi.org/10.1002/nem.2100.E2100nem.2100>
30. Soto, P., Camelo, M., Fontaine, J., Girmay, M., Shahid, A., Maglogiannis, V., De Poorter, E., Moerman, I., Botero, J.F., Latré, S.: Augmented Wi-Fi: An AI-based Wi-Fi Management Framework for Wi-Fi/LTE Coexistence. In: 2020 16th International Conference on Network and Service Management (CNSM), pp. 1–9 (2020). <https://doi.org/10.23919/CNSM50824.2020.9269064>
31. Soto, P., Camelo, M., Mets, K., Wilhelmi, F., Góez, D., Fletscher, L.A., Gaviria, N., Hellinckx, P., Botero, J.F., Latré, S.: ATARI: a graph convolutional neural network approach for performance prediction in next-generation WLANs. *Sensors* (2021). <https://doi.org/10.3390/s21134321>
32. Luong, N.C., Hoang, D.T., Gong, S., Niyato, D., Wang, P., Liang, Y.C., Kim, D.I.: Applications of deep reinforcement learning in communications and networking: a survey. *IEEE Commun. Surv. Tutor.* **21**(4), 3133–3174 (2019). <https://doi.org/10.1109/COMST.2019.2916583>
33. Li, R., Zhao, Z., Sun, Q., Chih-Lin, L., Yang, C., Chen, X., Zhao, M., Zhang, H.: Deep reinforcement learning for resource management in network slicing. *IEEE Access* **6**, 74429–74441 (2018). <https://doi.org/10.1109/ACCESS.2018.2881964>
34. Koo, J., Mendiratta, V.B., Rahman, M.R., Walid, A.: Deep reinforcement learning for network slicing with heterogeneous resource requirements and time varying traffic dynamics. In: 2019 15th International Conference on Network and Service Management (CNSM), pp. 1–5 (2019). <https://doi.org/10.23919/CNSM46954.2019.9012702>
35. Montero, R., Agraz, F., Pagès, A., Spadaro, S.: Enabling multi-segment 5G service provisioning and maintenance through network slicing. *J. Netw. Syst. Manag.* **28**, 340–366 (2020)
36. Liu, Q., Han, T., Zhang, N., Wang, Y.: DeepSlicing: Deep Reinforcement Learning Assisted Resource Allocation for Network Slicing. In: GLOBECOM 2020 - 2020 IEEE Global Communications Conference, pp. 1–6 (2020). <https://doi.org/10.1109/GLOBECOM42002.2020.9322106>
37. Liu, Q., Han, T., Moges, E.: EdgeSlice: Slicing Wireless Edge Computing Network with Decentralized Deep Reinforcement Learning. In: 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS), pp. 234–244 (2020). <https://doi.org/10.1109/ICDCS47774.2020.00028>
38. Nakanoya, M., Sato, Y., Shimonishi, H.: Environment-Adaptive Sizing and Placement of NFV Service Chains with Accelerated Reinforcement Learning. In: 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), pp. 36–44 (2019)
39. Dab, B., Aitsaadi, N., Langar, R.: Q-Learning Algorithm for Joint Computation Offloading and Resource Allocation in Edge Cloud. In: 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), pp. 45–52 (2019)
40. Wang, L., Mao, W., Zhao, J., Xu, Y.: DDQP: a double deep Q-learning approach to online fault-tolerant SFC placement. *IEEE Trans. Netw. Serv. Manag.* **18**(1), 118–132 (2021). <https://doi.org/10.1109/TNSM.2021.3049298>
41. Pujol Roig, J.S., Gutierrez-Estevez, D.M., Gündüz, D.: Management and orchestration of virtual network functions via deep reinforcement learning. *IEEE J. Sel. Areas Commun.* **38**(2), 304–317 (2020). <https://doi.org/10.1109/JSAC.2019.2959263>
42. Huang, L., Feng, X., Zhang, C., Qian, L., Wu, Y.: Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing. *Digit. Commun. Netw.* **5**(1), 10–17 (2019). <https://doi.org/10.1016/j.dcan.2018.10.003>
43. Baek, J., Kaddoum, G., Garg, S., Kaur, K., Gravel, V.: Managing Fog Networks using Reinforcement Learning Based Load Balancing Algorithm. In: 2019 IEEE Wireless Communications and Networking Conference (WCNC), pp. 1–7 (2019). <https://doi.org/10.1109/WCNC.2019.8885745>
44. Zhao, X., Zong, Q., Tian, B., Zhang, B., You, M.: Fast task allocation for heterogeneous unmanned aerial vehicles through reinforcement learning. *Aerosp. Sci. Technol.* **92**, 588–594 (2019). <https://doi.org/10.1016/j.ast.2019.06.024>
45. Davoli, G., Cerroni, W., Borsatti, D., Valieri, M., Tarchi, D., Raffaelli, C.: A fog computing orchestrator architecture with service model awareness. *IEEE Trans. Netw. Service Manag.* (2021). <https://doi.org/10.1109/TNSM.2021.3133354>
46. Multi-access Edge Computing (MEC); General principles, patterns and common aspects of MEC Service APIs. Tech. rep., European Telecommunications Standards Institute (ETSI) (2021)
47. Comşa, I., Trestian, R., Muntean, G., Ghinea, G.: 5SMART: a 5G SMART scheduling framework for optimizing QoS through reinforcement learning. *IEEE Trans. Netw. Service Manag.* **17**(2), 1110–1124 (2020). <https://doi.org/10.1109/TNSM.2019.2960849>

48. Guizani, Z., Hamdi, N.: CRAN, H-CRAN, and F-RAN for 5G systems: key capabilities and recent advances. *Int. J. Netw. Manag.* **27**(5), e1973 (2017). <https://doi.org/10.1002/nem.1973.E1973nem.1973>
49. Labriji, I., Meneghello, F., Cecchinato, D., Sesia, S., Perraud, E., Strinati, E., Rossi, M.: Mobility aware and dynamic migration of MEC services for the internet of vehicles. *IEEE Trans. Netw. Service Manag.* **18**(1), 570–584 (2021). <https://doi.org/10.1109/TNSM.2021.3052808>
50. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Hiedermiller, M., Fiedjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. *Nature* **518**(7540), 529–533 (2015). <https://doi.org/10.1038/nature14236>
51. Howard, R.A.: Information value theory. *IEEE Trans. Syst. Sci. Cybernet.* **2**(1), 22–26 (1966). <https://doi.org/10.1109/TSSC.1966.300074>
52. Kam, C., Kompella, S., Nguyen, G.D., Wieselthier, J.E., Ephremides, A.: On the age of information with packet deadlines. *IEEE Trans. Inf. Theory* **64**(9), 6419–6428 (2018). <https://doi.org/10.1109/TIT.2018.2818739>
53. Bellavista, P., Giannelli, C., Montenero, D.D.P., Poltronieri, F., Stefanelli, C., Tortonesi, M.: HOListic pROcessing and NETworking (HORNET): an integrated solution for IoT-based fog computing services. *IEEE Access* **8**, 66707–66721 (2020). <https://doi.org/10.1109/ACCESS.2020.2984930>
54. Dong, H., Dong, H., Ding, Z., Zhang, S., Chang: *Deep Reinforcement Learning*. Springer, Singapore (2020)
55. Puterman, M.L.: *Markov decision processes: discrete stochastic dynamic programming*. Wiley, Hoboken (2014)
56. Qi, Q., Tao, F.: A smart manufacturing service system based on edge computing, fog computing, and cloud computing. *IEEE Access* **7**, 86769–86777 (2019). <https://doi.org/10.1109/ACCESS.2019.2923610>
57. Zhang, K., Cao, J., Maharjan, S., Zhang, Y.: Digital twin empowered content caching in social-aware vehicular edge networks. *IEEE Trans. Comput. Social Syst.* (2021). <https://doi.org/10.1109/TCSS.2021.3068369>
58. Wilhelmi, F., Barrachina-Munoz, S., Bellalta, B., Cano, C., Jonsson, A., Ram, V.: A flexible machine-learning-aware architecture for future WLANs. *IEEE Commun. Mag.* **58**(3), 25–31 (2020). <https://doi.org/10.1109/MCOM.001.1900637>
59. Poltronieri, F., Stefanelli, C., Suri, N., Tortonesi, M.: Phileas: A Simulation-based Approach for the Evaluation of Value-based Fog Services. In: 2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), pp. 1–6 (2018). <https://doi.org/10.1109/CAMAD.2018.8514969>
60. Alameddine, H.A., Sharafeddine, S., Sebbah, S., Ayoubi, S., Assi, C.: Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing. *IEEE J. Sel. Areas Commun.* **37**(3), 668–682 (2019). <https://doi.org/10.1109/JSAC.2019.2894306>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.


Filippo Poltronieri received the Ph.D. degree from the University of Ferrara, Italy, in 2021. He is an Assistant Professor at the Department of Engineering at the University of Ferrara. His research interests include Distributed Systems, Edge/Fog Computing, Cloud Computing, Service Management, and tactical networks. He has been visiting the Florida Institute for Human & Machine Cognition (IHMC) in Pensacola, FL (USA) in 2016–2017 and 2018.

Cesare Stefanelli is a full professor of computer science engineering at the University of Ferrara, Italy, in the area of distributed systems. He received his Ph.D. in computer science from the University of Bologna in 1996. At the University of Ferrara he coordinates an industrial research and technology transfer Technopole Lab. He is in the Board of Directors of Bi-Rex, a nation wide Industry 4.0 Competence Center. He holds several patents jointly developed with several companies. His research interests include distributed and mobile computing in wireless and ad hoc networks, network and systems management, and network security.

Niranjan Suri is the Division Associate for Research in the Information Sciences Division at the US Army Research Laboratory and an Associate Director at the Florida Institute for Human and Machine Cognition. He received his PhD in Computer Science from the University of Lancaster in 2008. He has co-authored over 200 publications in a variety of international books, journals, and conferences. His research interests include communications, networking, distributed systems, agile computing, information management, and Internet of Things.

Mauro Tortonesi received the Ph.D. degree in computer engineering from the University of Ferrara, Italy, in 2006. He was a Visiting Scientist with the Florida Institute for Human & Machine Cognition (IHMC), Pensacola, FL, USA, from 2004 to 2005 and with the United States Army Research Laboratory, Adelphi, MD, USA, in 2015. He is currently an Associate Professor with the Department of Mathematics and Computer Science, University of Ferrara. He has co-authored over 90 articles in the distributed systems research area, with particular reference to IoT solutions in industrial and military environments, Cloud and Fog computing, wireless middleware, and IT service management.

Authors and Affiliations

Filippo Poltronieri¹ · Cesare Stefanelli¹ · Niranjan Suri^{2,3} · Mauro Tortonesi⁴ 

¹ Department of Engineering, University of Ferrara, Ferrara, Italy

² Florida Institute for Human and Machine Cognition (IHMC), Pensacola, FL, USA

³ US Army Research Laboratory (ARL), Adelphi, MD, USA

⁴ Department of Mathematics and Computer Science, University of Ferrara, Via Machiavelli 30, 44121 Ferrara, Italy