Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH Open Access

Deep Reinforcement Learning techniques for dynamic task offloading in the 5G edge-cloud continuum

Gorka Nieto[1,2*], Idoia de la Iglesia[1], Unai Lopez-Novoa[2] and Cristina Perfecto[2]

Abstract
The integration of new Internet of Things (IoT) applications and services heavily relies on task offloading to external devices due to the constrained computing and battery resources of IoT devices. Up to now, Cloud Computing (CC) paradigm has been a good approach for tasks where latency is not critical, but it is not useful when latency matters, so Multi-access Edge Computing (MEC) can be of use. In this work, we propose a distributed Deep Reinforcement Learning (DRL) tool to optimize the binary task offloading decision, this is, the independent decision of where to execute each computing task, depending on many factors. The optimization goal in this work is to maximize the Quality-of-Experience (QoE) when performing tasks, which is defined as a metric related to the battery level of the UE, but subject to satisfying tasks' latency requirements. This distributed DRL approach, specifically an Actor-Critic (AC) algorithm running on each User Equipment (UE), is evaluated through the simulation of two distinct scenarios and outperforms other analyzed baselines in terms of QoE values and/or energy consumption in dynamic environments, also demonstrating that decisions need to be adapted to the environment's evolution.

Keywords Task offloading, Performance evaluation, Energy consumption, Reinforcement Learning (RL), Quality-of-Experience (QoE), Multi-access Edge Computing (MEC), Internet of Things (IoT), Edge-Cloud-Continuum

Introduction

As a key enabler of a new industrial and social revolution, Internet of Things (IoT) drives a wide range of possibilities, facilitating the generation and utilization of previously nonexistent data. In fact, IoT is becoming more popular in many industries, such as manufacturing, healthcare, retail, or vehicle industry [26]. IoT and its connectivity capabilities, combined with data analytics, open up new possibilities not only to develop cutting-edge goods and services but also to improve operational efficiency.

However, in many scenarios, these applications are limited by the constrained resources of IoT devices, involving both computation capabilities and battery constraints, among others [19]. That is why, often, these devices need to rely on external resources in order to offload some of their computational tasks, such as getting insights from the data caught by themselves. Even if the task could be run in the device itself, the energy usage may be too high, which would result in the need to recharge batteries more frequently or even replacing them.

To avoid this high battery consumption and to ensure the service, Cloud Computing (CC), or in the context of mobile communications, Mobile Cloud Computing (MCC) paradigm, emerged as a promising solution for tasks where latency is not critical, above all.

*Correspondence:
Gorka Nieto
gnieto@ikerlan.es
[1] Ikerlan Technology Research Centre, Basque Research and Technology Alliance (BRTA), Pº. J. M. Arizmendiarrieta, 2, Arrasate-Mondragón 20500, Spain
[2] University of the Basque Country (UPV/EHU). School of Engineering in Bilbao, Alameda Urquijo s/n, Bilbao 48013, Spain

Unfortunately, when the completion time must be very short, this solution may not be adequate because the Round-Trip Time (RTT) from the device to the cloud server might be too high.

To solve both local issues and the problems derived from reaching remote cloud servers, European Telecommunications Standards Institute (ETSI) developed a new paradigm called Multi-access Edge Computing (MEC), formerly Mobile Edge Computing, but which became updated to enable the communications through different technologies, such as 5G-New Radio (NR) or Wireless-Fidelity (Wi-Fi). MEC, as stated by ETSI, is "a natural development in the evolution of mobile Base Station (BS) and the convergence of IT and telecommunications networking" and "offers application developers and content providers cloud-computing capabilities and an IT service environment at the edge of the network" [18]. In short, it brings the benefits of the CC paradigm closer to the User Equipment (UE). 5G was the communication technology that enabled MEC paradigm for the first time, and is also expected to enable more connected devices. It is also supposed to bring improved energy-saving capabilities for IoT devices, leading to reduced energy use and increased battery life [25].

In this context, it is important to consider the drawbacks of local execution, but also the implications task offloading may have. Many works in the literature have presented tools that optimize the decision of where to execute the computing tasks, depending on many factors. Apart from classical mathematical optimization methods, with the recent advancements in Artificial Intelligence (AI), many different approaches have been set up, including Machine Learning (ML) or Deep Learning (DL) approaches, techniques that learn from data and improve their performance over time. However, a significant limitation of this approach is the need of dataset that represents every possible state of the environment, so the ML algorithm learns how to act each time. Having the proper datasets containing every situation to learn from may be unrealistic [9], so there is a need to have a tool that can learn without needing a dataset. Here is where Reinforcement Learning (RL) can be of use as, instead of relying on an existing dataset, it learns by continuously interacting with the environment and assessing the effects of its actions.. Deep Reinforcement Learning (DRL) is an improved version of RL that uses Deep Neural Network (DNN) to optimize a specific metric, through the estimation of the value or policy function for high-dimensional state and action spaces.

Another important matter is where to take this decision. On the one hand, running such algorithms can be time- and computing-costly for a resource-constrained device [17]. On the other, if one of the outputs of the algorithm is to decide not to offload any information because the communications are not guaranteed or the server is not available, it would not make sense for the decision to be taken by an external agent, as it would have a communication problem with the device. This would also add more information or noise to a channel that could already be saturated [32], apart from the fact that no decision would be taken.

As commented earlier, the main goals to be optimized in an IoT computation offloading problem may be to reduce the energy consumption of the device and fulfil the latency requirements of the application, although there can be different approaches, like optimizing the bandwidth or accomplishing computation tasks while maintaining a certain level of privacy. Taking all this into account, this work presents a DRL-based technique to maximize the User Experience (UX) of IoT services, this is, guaranteeing a level of Quality-of-Experience (QoE) in the decision-making so the tasks', and therefore, the users' requirements are fulfilled.

Thus, the contributions of this work are:

- A novel system model that considers devices' and servers' occupation, the network status and different task types, rather than focusing solely on tasks' characteristics.
- A QoE calculation based on the energy performance, conditional on meeting the latency requirements of the tasks and trying to maximize their success.
- A distributed DRL approach, with which each UE aims to maximize the QoE value in a time-varying environment.
- An empirical evaluation of the algorithm in two distinct scenarios that shows the performance of the proposed solution, along with other baselines, proving the need for a dynamic and adaptive approach.

The performance of the algorithm is demonstrated through the experiment outcomes in terms of QoE values, energy consumption, and the evolution in the decisions of the algorithms for each independent task. These results not only show that solutions like ML-based ones are necessary in dynamic environments, as the decisions need to be adapted to its evolution, but they also show that the proposed algorithm outperforms other studied baselines.

The remainder of this paper is structured as follows: some recent proposals from the literature that tackle the task offloading problem are reviewed in Related work section. System model section presents the scenario raised in this studio, for which Problem formulation and proposed approach section presents its problem formulation and the optimization goal, along with the suggested

DRL-based approach to solve it. After that, Simulation environment section provides a detailed exposition of the experimental environment configuration and the baselines that will be compared in Discussion section, which presents a comprehensive discussion of the obtained experimental results, highlighting the advantages of the proposed solution. In Conclusions and future work section, we conclude this work and outline some directions for further research.

Related work

Being computation offloading such a trending problem, a diverse range of approaches has emerged to address this challenge, each with its own optimization goals and employing distinct techniques or perspectives. In this section, a dual taxonomy has been established, classifying the employed techniques into two distinct groups: works based on the principles of ML and those utilizing alternative AI methodologies such as heuristics or metaheuristics.

Non ML-based solutions

These solutions are designed to solve specific problems through a set of predefined rules or algorithms that do not involve learning from data or ML, for instance, heuristics ("rules of thumb") and metaheuristics (higher-level strategies used to find optimal solutions in complex problems). These solutions are designed and implemented based on problem-specific knowledge and do not adapt over time.

First, [42] model apps as application graphs and the physical computing system as a physical graph, with resource demands or availability annotated on these graphs. They propose a heuristic algorithm to find where to place an application and then generalize the formulation and obtain online approximation algorithms with a polynomial-logarithmic (poly-log) competitive ratio for tree application graph placement, trying to minimize the maximum resource utilization at physical nodes and links. They consider node and link assignments and incorporate multiple types of computational resources at nodes. However, they do not consider devices' battery and channel status, nor delay requirements of the tasks.

The work in [6] proposes a joint and adaptive optimization of resource and task allocation in mobile stream applications in 5G-supported mobile-fog-cloud virtualized ecosystems. The objective is to minimize the energy of the overall ecosystem while meeting hard constraints on the minimum streaming rate and maximum computing-plus-networking resources. The authors model the energy of the target ecosystem and develop an optimality-preserving decomposition into a continuous resource allocation sub-problem and a discrete task allocation sub-problem. They propose a set of gradient-based adaptive iterations for the resource allocation sub-problem, and an ad-hoc-developed Genetic Algorithm (GA) for the task allocation sub-problem. All the reported numerical results are carried out by the VirtFogSim toolbox developed in [36], which allows for dynamic joint optimization of energy and delay performance by optimizing the placement of application tasks and allocation of computing-networking resources. This framework also allows customizable simulation of the energy-delay performance of the overall system. The limitations of this work are the scalability challenges when the size and complexity of the application Dynamic Acyclic Graph (DAG) and system parameters increase and/or introduce overhead. Besides, despite VirtFogSim supports dynamic tracking of resource allocation under time-varying operational environments, the ability to adapt in real-time to rapidly changing operational environments is limited. DAG and other parameters related to the computing and networking aspects of the Mobile-Fog-Cloud platforms are used as inputs to make decisions.

In contrast with the previous works, the authors in [28] resort to Lyapunov optimization to model energy consumption in mobile cloud systems, so no edge server is considered. This approach considers real-time network conditions, workload types, and various workload arrivals as inputs; and decisions on where to execute and task scheduling as outputs, among others, along with the stability of the queues. The main limitations of this approach are the sensitivity to network variability and the complexity of the joint optimization, along with its potential issues with scalability and performance in large-scale real-world mobile cloud systems.

Works like [15] present task offloading as an enabler for different real applications. In this case, the real-time object detection for mobile Augmented Reality (AR) is presented, which would require too much energy if made fully in the device. A task placement and offloading framework is developed and used to extract DL tasks from the AR application, which are the most consuming tasks in this scenario, so they can be offloaded to nearby Graphics Processing Unit (GPU)-powered edge devices. With the developed infrastructure simulator, they carry an empirical analysis of both centralized and distributed execution of the task placement algorithm, evaluating their *Nimbus* algorithm performance against other related solutions. They consider latency, battery consumption, and task coordination as the constraints of the apps and choose a server to which to offload considering the battery consumption and the time required to get to it. However, they do not consider the status of the server.

Instead of choosing a specific kind of application as an example, [4] shows real measurements using

Raspberry Pi devices as evidence of real implementation of offloading algorithms in actual devices, aiming to reduce the energy consumption and execution time measurements of different IoT apps running on physical IoT sensor nodes. They propose adaptive schemes that consider the resources available at the IoT nodes and available Edge servers. Nevertheless, the adaptive algorithms proposed in this work are not tested in dynamic environments when server unavailability or channel degradation may occur, so they may not fit in time-changing environments.

Regarding environments with multiple servers, [46] presents a joint optimization problem where the objective is to minimize the cost of collaboration or information interchange among edge servers while maximizing the offloading ratio of sensing data and satisfying the limitations of system energy consumption and network latency. UX is enhanced when many edge servers collaboratively offload all or part of the sensory data that was originally provided to a cloud centre, but there is a cost of cooperation due to the system resources used for the exchange of information amongst edge servers, so the goal is to find the balance. The problem is solved using Heuristics techniques, specifically, Alternating Direction Method of Multipliers (ADMM), but they do not consider fluctuations in the transmission time between IoT and edge servers, between edge servers, nor between edge and cloud servers.

In [45], the goal is to minimize the maximum long-term execution delay while ensuring that the long-term energy consumption of each UE in the network does not exceed the maximum energy consumption. They propose a centralized solution, where the edge server makes the task offloading decision based on the system state of all UEs, gathering the data from each of them. The pros of the proposed solution are that it provides an optimal solution to the task offloading problem, and it achieves the lowest execution delay compared to some baseline algorithms. Anyway, more complex scenarios are missing, where communication and computation resources are affected by the actions taken by many UEs and some factors like the channel quality are dynamic

Aiming to minimize network resource usage and maximize requests admissions with Heuristics, [27] formulate the simultaneous optimization of communication and computational resource allocation in response to requests for computation offloading with strict bounding constraints, modelled as a two-tier computational offloading problem. In this case, as some differences with our work, UEs are not considered as a computing entity and always try to fill the MEC servers, before sending to the cloud server, instead of considering tasks with no latency requirements that can be directly offloaded to the cloud server.

A distributed multi-hop task offloading decision model for task execution efficiency is proposed in [10], with a vehicle selection mechanism and a task offloading decision algorithm. The optimization model is established to increase the task execution efficiency. It is solved by the greedy algorithm and the discrete bat algorithm, respectively, under the scenarios set in this article. The random selection and the completely local computing schemes were used as compared baselines. In this work, they do not consider the dynamics of computing resources and network channels, which can be a consequence of the offloading decisions from multiple vehicles.

Despite their diverse approaches, these proposals share the same shortcomings. They heavily depend on the specific characteristics of the problem they have been designed for, without the ability to adapt and learn from past experiences, so they are not capable of handling dynamic and evolving environments, which are not considered in many of them. These limitations could be solved using ML-based techniques, as they can learn from past experiences. Other limitations are the scalability challenges, above all, for the centralized solutions, as they might become unfeasible when they have to consider the information of an increasing number of nodes.

ML-based solutions

In contrast with the previous solutions, AI-based solutions can learn from data and make decisions based on patterns that are not explicitly created and introduced into them, but rather discovered through their training.

First, [24] evolves the *DROO* algorithm from [22] to address the task offloading problem, along with data from *EUA* dataset [29]. Different from our work, this solution is centralized, so decision-making might be challenging mainly because of the overhead produced in both the network and the MEC server by sending much information, apart from the privacy concerns this could raise [13] and the scalability limitation of RL-based algorithms due to the huge decision space, increasing network congestion and, consequently, latencies [12].*LyDROO* [7], as another proposal emanated from *DROO*, applies Lyapunov optimization to decouple the multi-stage stochastic Mixed-Integer Nonlinear Programming (MINLP) and solves each resulting subproblem via *DROO* algorithm.

The work from [5] proposes an approach whose goal is to minimize this sum-cost of the system, this is, minimizing the end-to-end delay and energy consumption while complying with the application requirements by offloading to available edge servers. They use a two-stage RL-based mechanism to optimize where the transmission from UEs to servers and also from one server to another one, in a second stage. Thus, the first stage, the offloading decision stage, is performed in a decentralized way, while the second stage is performed in the servers.

Many works combine ML-based solutions with other tools like meta-heuristics. For example, [3] uses DL with Seagull Optimization (TORA-DLSGO) algorithm aiming to minimize energy consumption subject to the latency requirements and restricted resources, using SGO algorithm for the parameter tuning of the Deep Belief Network (DBN) model, which is used for optimum offloading decision-making purposes. An objective function is derived based on minimizing energy consumption subject to the latency requirements and restricted resources. Despite the results being promising and different network parameters being tested, there is no evidence of their results in time-changing scenarios.

Also mixing many techniques and trying to optimize many objectives at the same time, [43] shows a joint optimization problem, which tries to optimize both the delay and energy consumption based on queuing theory, obtaining the optimal offloading strategy in dynamic and random multi-user offloading environments through MA-Deep Reinforcement Learning based on Queuing theory (QDRL). Many indicators are used, such as task arrival rate, bandwidth, energy consumption, or latency. The problem is defined as an Partially Observable Markov Decision Process (PO-MDP) and actors and critics share most of the network except for the output layer. Also, it seems to be feasible to implement in a real-world scenario because the training is centralized but the execution is performed in a decentralized way. Nevertheless, due to the dynamic nature of real environments, MA-QDRL may suffer the problem of high complexity when the number of nodes increases.

Different environments are also considered in [11], which tackle single- and multi-MEC systems, aiming to minimize the energy consumption of each UE while satisfying the delay requirement. They use the amount of energy consumed by the UE as the reward, along with the reward for the fact of completing the task and fulfilling its delay constraints. Other works like [33] that consider dependencies among different tasks. This work uses an Improved Policy Loss Clip (IPLC)-based Proximal Policy Optimization (PPO) algorithm to achieve lower latency, reduce energy consumption, and improve Quality-of-Service (QoS). Though, no cloud computation is considered for big tasks or non-delay critical tasks.

[48] uses a centralized Double Deep Q-Learning Network (DDQN) to determine the joint policy of computation offloading and resource allocation. The state space comprises the available computation resources of the MEC, the percentage of the available spectrum resources at present, and the energy consumption at each time slot is observed to compare whether the optimal state is reached. For the long-term consideration, a state value function is defined by the cumulative discounted reward value for agents in the state. Despite the promising results of this work, it is not compared to other DRL proposals, nor consider more complex scenarios, where cloud servers are considered and where other phenomena can affect the wireless channel gain, such as shadow fading or fast fading phenomena.

The convergence with technologies such as Digital Twins or caching is also considered in many works. For instance, an offloading decision problem based on Digital Twins Edge Networks is planned in [38] to reduce overall system overhead and improve the cache hit rate. The DRL method Deep Deterministic Policy Gradient (DDPG) exports the compute offloading choice and the DTCC algorithm increases the cache hit percentage, respectively. These promising results should be checked in more complex environments, considering different task requirements and also leaning on cloud servers.

Also, [44] converts the problem into a per-frame deterministic problem through the Lyapunov optimization technique. Thus, it results in 2 sub-problems, resource allocation, and computation offloading, which are strongly coupled and difficult to solve directly. The authors design a queue-aware computation offloading scheme based on AC to address both sub-problems separately. The Actor module is implemented based on a DL model and quantization strategy COGA for generating computation offloading actions. In the critic module, on the other hand, they integrate mathematical reasoning and learning-based methods for resource allocation. The proposed model should also support cloud-edge-device collaboration architecture.

Considering that directly deploying RL in edge computing implies a bad experience due to an initial arbitrary exploration in real online environments, in [47] a static offline dataset is used, so the exploration is reduced, which, in RL, is expensive or even dangerous in online environments. Thus, they model the problem as a repeated game between 2 agents and apply model-based offline RL (specifically, Soft Actor-Critic (SAC)) to optimize the offloading decisions. The online environment and the simulated offline environmental model may not be so similar, though, as the environment dynamics can only be inferred implicitly.

As a paradigm shift, reverse offloading is proposed is presented in [35]. Concretely, the authors present a framework for reverse offloading that carefully balances the relationship between task completion time and UE energy consumption. The sensory data from every device is transferred to the MEC server for data fusion. Reverse offloading is the process of returning tasks, data, and sensory information from the MEC to the mobile that initiated them for computing. Tasks can either be finished on the MEC server or reverse-offloaded to the originating device. Sending so much information to the MEC server may cause overhead in the network.

Additionally, [31] aims to optimize sub-channel assignment and task offloading decisions together to guarantee the QoS that delay-sensitive workloads require, modifying the original problem through DRL. Furthermore, a distributed DRL approach is presented to minimize significant signalling overhead for each Industrial Internet of Things (IIoT) device to acquire explicit knowledge on full system states, which turns the problem into a PO-MDP. In order to address this issue, Long short-term memory (LSTM) is additionally utilized to forecast the current load status of every sub-channel and MEC server, which is utilized to inform RL agents' decisions. Anyway, this work aims to maximize the number of tasks with satisfied delay requirements without due consideration to consumption, but considering channel gain and queue length.

Next, [16] work's goal is to reduce the overall latency of the MEC system while ensuring that each task is completed within its deadline and resource constraints. It handles both partial and complete offloading tasks and uses energy consumption as a factor in the offloading decision. It also adapts to changing network conditions and resource availability by using a DDQN to learn the optimal offloading policy, using Karush-Kuhn-Tucker (KKT) to minimize the total latency and DQN to reduce the energy consumption. The proposed MEC distributed scheduling algorithm must consider not only the individual state information of each agent but also the global interrelationships between the nodes. [39] propose a semantic-driven computation offloading and resource allocation scheme. For the offloading decision, they use a Convolutional Neural Network (CNN) segmentation scheme, and for the allocation problem, they use a MA-DQN algorithm, aiming to optimize latency, energy consumption, and task performance.

Also focusing on energy and latency, this time through a QoE calculation, [21] propose Gated Recurrent Unit Fictitious Self-Play Dueling Double Deep Recurrent Q Network. It adopts a Recurrent Neural Network (RNN) structure to learn from the history of observations and Neural Fictitious Self-Play (NFSP), which is essentially an extension of Fictitious Self-Play (FSP) and addresses the non-stationary issue. It is also a PO-MDP due to its distributed nature.

In the case of [30], the authors tackle the joint optimization problem of the task offloading and resource allocation in small cell MEC networks, where multiple small-cell BSs integrating MEC servers provide computing services for the served UEs. The proposed reward function for the Proximal Policy Optimization (PPO) agent used is composed of two parts: the total energy consumption and the delay constraint satisfaction probability.

Finally, the scenario considered in [37] consists of a heterogeneous network of UEs that can offload to a network of MEC servers through an agent that orchestrates a group of 5G Small Cells enhanced with computation and storage capabilities to optimize resource utilization and minimize latency. This agent is an Advantage Actor-Critic (A2C) agent, which takes into account computation, battery, latency, and communication constraints, some of which are often overlooked in traditional approaches. The solution is also shown to be scalable, data-efficient, robust, stable, and adaptable, ensuring optimal system performance even under worst-case scenarios.

Summary

Table 1 summarizes all of these works, both ML and non-ML-based, showing briefly the technique used; if it solves a Task Offloading (TO) or Resource Allocation (RA) problem; if it is a centralized (C) or a Decentralized (D) proposal; where the tasks can be executed (locally (L), MEC/Fog server (M/F) or Cloud server (C)), also marking with an * where the decision is taken; and which the optimization goal (or goals) is.

Among all these works, our work is the only ML-based distributed one that considers different task classes, based on some of their characteristics, such as a range in delay requirement, to make a decision offloading to a MEC or a cloud server, along with dynamic behaviour of the different elements composing the network, such as the channel gain or the computing resources availability.

System model

This section presents the way the scenario of our proposal is modeled, which shows the entities that form part of it; the modeling of the computational tasks used to configure the simulations; and the modeling of the energy consumption and elapsed timed, according to where the tasks are executed.

System architecture

The considered system in this work is shown in Fig. 1: A three-tier architecture encompassing a cloud layer, a MEC layer and a device layer. The system is considered to represent a Base Station (BS) located within the facility with an adjacent MEC server. This scenario acknowledges the presence of various entities that may potentially obstruct the signal propagation between devices and the BS.

Table 1 Related work

Ref	Non-ML	ML	Technique	TO	RA	C/D	L	M/F	CC	Goal(s)
Wang et al. [42]	X		Heuristics		X	C	X	X*	X*	Minimize resource utilization
Baccarelli et al. [6]	X		Meta-Heuristics		X	C	X	X*	X*	Minimize the energy of the overall ecosystem
Scarpiniti et al. [36]										
Kwak et al. [28]	X		Control Theory		X	C	X		X*	Minimize energy in mobile cloud systems
Cozzolino et al. [15]	X		Heuristics	X	X	D	X	X*	X	- Reduce energy consumption - Reduce execution time
Al Aidaros et al. [4]	X		Control Theory	X	X	D-C	X*	X*		- Reduce energy consumption - Reduce execution time
Yuan et al. [46]	X		Heuristics	X		D		X*	X	- Maximize offloading ratio - Reduce collaboration cost - Satisfy energy and latency constraints
Xu and Yang [45]	X		Deterministic	X		C	X	X*		Minimize long-term delay (energy constraints for UE)
Kovacevic et al. [27]	X		Heuristics	X	X	C		X*	X	- Minimize overall usage of network - Maximize latency accomplishment
Chen et al. [10]	X		Meta-Heuristics	X		D	X*	X		Minimize the delay
Jiao et al. [24]	X	X	Meta-Heuristics & DRL	X		C	X	X*		Optimize time-energy trade-off (weighted sum of time and energy)
Huang et al. [22]		X	DRL	X	X	C	X	X*		Maximize computation rate
Bi et al. [7]	X	X	DRL & Control Theory	X	X	C	X	X*		Maximize computation rate
Avgeris et al. [5]		X	RL	X	X	C	X*	X*		Maximize computation rate
Pan et al. [33]		X	DRL	X		C	X	X*		Improve QoS (energy and latency)
Wu et al. [43]	X	X	Queuing Theory & DRL	X		D	X*	X		- Reduce energy consumption - Reduce execution time
Chen and Liu [11]		X	DRL	X	X	D	X*	X		- Minimize energy consumption - Satisfying delay requirement
Zhou et al. [48]		X	DRL	X	X	C	X	X*		Minimize energy consumption of the system
Song and Shen [38]		X	DRL	X		C	X	X*		- Reduce energy consumption in the system - Reduce execution time in the system
Abdullaev et al. [3]	X	X	Meta-Heuristics & DRL	X	X	C	X	X*	X	- Minimize energy consumption - Satisfying delay requirement
Xu et al. [44]	X	X	Control Theory & DRL	X	X	D	X*	X		Maximize the total real Computing Rate
Zhang et al. [47]		X	DRL	X		D	X*	X		Maximize the long-term utility (quality and latency)
Saeed et al. [35]		X	DRL	X		C	X	X*		- Reduce energy consumption - Reduce execution time
Lin et al. [31]		X	DRL	X	X	D	X*	X		Maximize tasks with satisfied delay
Dong et al. [16]	X	X	Math. Opt. & DRL	X	X	D	X	X*		Minimize latency

Table 1 (continued)

Ref	Non-ML	ML	Technique	TO	RA	C/D	L	M/F	CC	Goal(s)
Sun et al. [39]		X	DRL	X	X	D	X	X*	X	- Minimize energy cost - Minimize delay cost - Optimize task Performance
Hou et al. [21]		X	DRL	X		D	X*	X		QoE (energy- and latency- dependant)
Li et al. [30]		X	DRL	X	X	C	X	X*		Minimize devices' energy consumption (delay constraints)
Silva et al. [37]		X	DRL	X	X	C	X	X*		- Minimize energy cost - Minimize delay cost
Ours		**X**	**DRL**	**X**		**D**	**X***	**X**	**X**	- Reduce device energy - Satisfy latency constraints

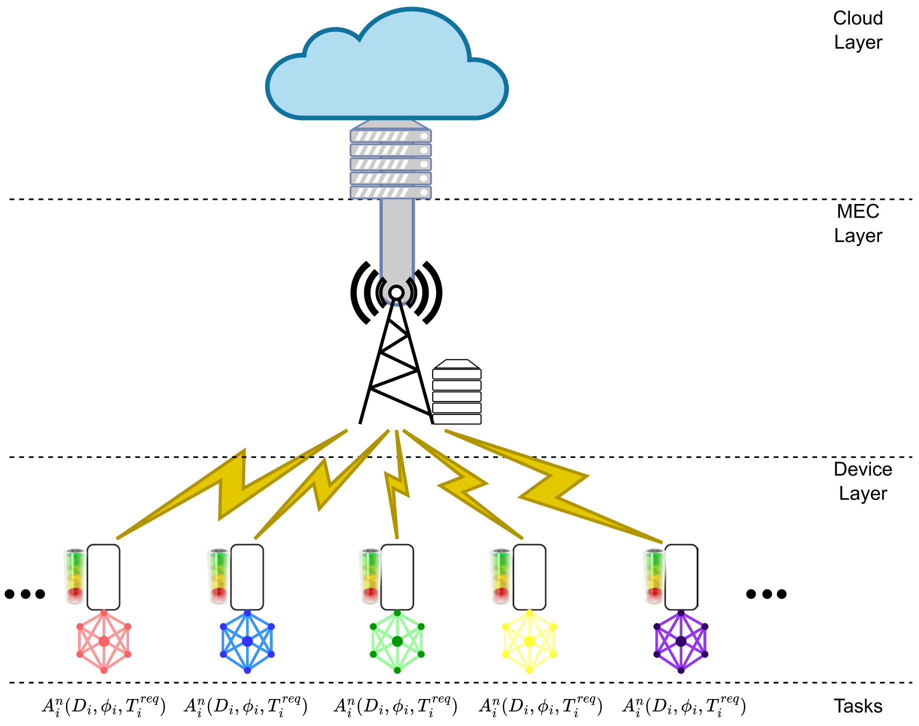
Cloud Layer
MEC Layer
Device Layer
Tasks
$A_i^n(D_i, \phi_i, T_i^{req})$
$A_i^n(D_i, \phi_i, T_i^{req})$
$A_i^n(D_i, \phi_i, T_i^{req})$
$A_i^n(D_i, \phi_i, T_i^{req})$
$A_i^n(D_i, \phi_i, T_i^{req})$

Fig. 1 System model

The device layer is composed of every UE in the environment, which must decide what to do with the incoming tasks, making use of their own distributed decision-making algorithm instances and considering many conditions, such as their current battery, as it will be explained later in Problem formulation and proposed approach section.

Regarding the MEC layer, each MEC server in this layer is located next to a Base Station (BS), so the distance between them is 0. Here, when a device decides to offload a task to its attached server in the MEC layer, it must send the task through a wireless link and wait for the task to reach back once the MEC server has processed it with its remaining available resources.

Finally, the cloud layer is reached from any BS in the MEC layer through a modelled fibre link, in case the decision is to offload the task to the cloud server. Thus, it is important to note that every task that needs to be processed in the cloud server needs to go first through a BS, so 2 steps are needed: the wireless transmission from the UE to the BS and the transmission through an optical fibre from the BS to the cloud server.

Task model

Computation tasks are denoted by A_i^n $(1 \leq i \leq I)$, where i indicates the timestep where the task arrives at the n-th UE, considering I is the total number of tasks or timesteps, and n indicates the identification of the UE among the total number of UEs N. A_i is characterized by 3 parameters, $A_i^n(D_i, \phi_i, T_i^{req})$, where D_i is the data size of A_i^n in bits, ϕ_i is the number of computation cycles needed to complete A_i^n, and T_i^{req} is the maximum delay for the task to be completed in seconds.

With this information, each task is classified into one of the following classes (C_i):

- **Class 1 - Delay-sensitive tasks ($C_i = 1$):** Tasks with strict latency requirements and small size and computational requirements, which must be executed quickly. An example application with theses tasks could be a smart sensor, which monitors and adjusts environment parameters, such as temperature ,quickly.
- **Class 2 - Energy-sensitive tasks ($C_i = 2$):** Tasks with larger computational demands than the previous ones and some latency requirements to be fulfilled - but not as restrictive. An AR app could be one of these tasks, needing to process data, render graphics, and respond to interactions on time.
- **Class 3 - Insensitive tasks ($C_i = 3$):** Tasks with larger limits of both size (and the consequent computation requirements) and latency. For example, a Smart Grid IoT application could have tasks like these. Massive amounts of data may be collected, implying a very large data size that does not need to be processed instantaneously.

As a note, our system does not model a partial offloading problem, which means dividing a particular task into subtasks that can scheduled individually. Instead, we consider non-partitionable independent tasks, i.e., the tasks cannot be further reduced and they contain sufficient information to be treated separately. Interdependence relationships between tasks have not been considered.

Local computation model

When a UE decides to execute one task locally, there is no intervention by external entities, so the energy and latency calculations are made using A_i^n task data and n UE conditions, explained in more detail as follows.

Local latency

The time or latency experienced when executing a task locally (t_i^{cL}) can be represented as in Eq. 1

$$t_i^{cL} = \frac{\phi_i}{f_i^n} \tag{1}$$

, where ϕ_i is the computation consumption of task A_i as stated previously and f_i^n represents the available resources in the n UE in timestep i. Other delays such as the queuing delays in the nodes have not been considered.

Energy consumption in local computation

Regarding the energy consumption when the decision is to execute a task locally, e_i^{cL}, and based on works like [11], can be represented as in Eq. 2

$$e_i^{cL} = \kappa^n (f_i^n)^2 \phi_i \tag{2}$$

, where κ^n represents the energy coefficient determined by the chip structure of the UE.

MEC communication and computation model

Latency when offloading to the MEC server

Regarding the wireless environment characteristics, [23] has been used as a reference. As stated there, for an indoor environment with Sparse clutter High Base Station (SH), the 3GPP TR 38.901 [1] uses Alpha-Betta-Gamma (ABG) model to specify the Path Loss (PL) model as in Eq. 3.

$$PL_{SH,i}^n(f_c, d) = 32.4 + 23\log_{10}(d_i^{nm}) + 20\log_{10}(f_c) \tag{3}$$

Shadow Fading (SF) is calculated assuming a normal distribution with zero mean and a standard deviation of $5.9dB$, as suggested by the 3GPP model for indoor factory environments [1].

For the channel gain calculation, the path loss and the shadow fading effects are considered, so the channel gain h_i^{nm} is obtained by subtracting the PL and the SF instantaneous values. Thus, the channel gain is calculated as in Eq. 4:

$$h_i^{nm} = -PL_{SH,i}^n - SF \tag{4}$$

The Fast Fading (FF) phenomena, which is usually modelled as a zero-mean Rayleigh distribution in mobile NLOS environment, is considered to be averaged out during the offloading process, so the variations in the channel gain due to fast fading are ignored during the offloading process, as explained in [41].

In this scenario, Orthogonal Frequency-Division Multiple Access (OFDMA) is implemented in the uplink to avoid conflicts between different UEs in the access to the m base station. Thus, the total frequency bandwidth W is divided into N equal sub-bands (one for each UE), $W^{nm} = W/N$. This way, according to Shannon's formula, the uplink transmission rate (r_i^{nm}) can be stated as:

$$r_i^{nm} = W^{nm} \log_2\left(1 + \frac{p_i^n h_i^{nm}}{N_0}\right) \tag{5}$$

The upload transmission delay ($t_i^{tx_m}$) is then calculated dividing the task size D_i by the result of the previous Eq. 5,

$$t_i^{tx_m} = \frac{D_i}{r_i^{nm}} = \frac{D_i}{W^{nm}\log_2\left(1 + \frac{p_i^n h_i^{nm}}{N_0}\right)} \tag{6}$$

As in the calculation of the local processing delay (Eq. 1), the processing latency experienced when executing a task in a MEC server ($t_i^{c_m}$) is calculated as in Eq. 7:

$$t_i^{c_m} = \frac{\phi_i}{f_i^m} \tag{7}$$

, being f_i^m the available resources in the m MEC server in timestep i.

The total offload delay includes the upload transmission delay ($t_i^{tx_m}$), the time the MEC server takes to process the task A_i ($t_i^{c_m}$) and the downlink transmission delay ($t_i^{rx_m}$). However, as the transmission power of the MEC server (p_i^m) is considerably higher than the UE's p_i^n and the resulting data of the processed task is much lower than the task data size (D_i), the downlink transmission delay ($t_i^{rx_m}$) can be neglected. Both transmission queues and computation queues operate as FIFO (First-In-First-Out) queues, ensuring that the current task completes within the current timeslot, and the subsequent task begins processing at the start of the next time slot. Thus, the total offload delay is:

$$t_i^m = t_i^{tx_m} + t_i^{c_m} + t_i^{rx_m} = \frac{D_i}{W^{nm}\log_2\left(1+\frac{p_i^n h_i^{nm}}{N_0}\right)} + \frac{\phi_i}{f_i^m} \tag{8}$$

Energy consumption when offloading to the MEC server

The energy consumption for the UE to offload a task to the MEC server (e_i^m) can be easily calculated as the product of the UE transmission power (p_i^n) and the transmission time of the task to the MEC server (t_i^m). Therefore, using the previously stated upload transmission delay (Eq. 6), the energy consumed when offloading a task is set as in Eq. 9:

$$e_i^m = p_i^n \cdot t_i^{tx_m} = p_i^n \cdot \frac{D_i}{r_i^{nm}} = p_i^n \cdot \frac{D_i}{W^{nm}\log_2\left(1+\frac{p_i^n h_i^{nm}}{N_0}\right)} \tag{9}$$

Cloud communication and computation model

Latency when offloading to the cloud server

In the same way as in local computation (Eq. 1) and MEC computation (Eq. 7), the processing latency experienced when executing a task in the cloud server (t_i^{cC}) is calculated as in Eq. 10:

$$t_i^{cC} = \frac{\phi_i}{f_i^C} \tag{10}$$

Regarding the transmission delay to the cloud server, both the propagation time from the BS to the cloud server in the uplink and the downlink and the transmission delay must be considered, as well as the upload transmission delay from the UE to the MEC server (Eq. 6).

The propagation delay is defined in Eq. 11:

$$t_i^{pC} = \frac{L_{mC}}{v} \tag{11}$$

, where L_{mC} is the distance between the BS where the server m is located and the cloud server, and v is the fibre propagation speed, and is calculated as the division between the speed of light in vacuum c and the refractive index of the fibre ρ ($v = c/\rho$).

The transmission delay follows the statement of dividing the task size D_i by the transmission rate of the fibre (r^f), which is simplified and calculated in the next Equation:

$$r^f = C^{h_f} \cdot (1-O^f)\cdot(1-F^f) = \frac{C^f}{\sqrt{WDM}} \cdot (1-O^f)\cdot(1-F^f) \tag{12}$$

, being C^{h_f} the channel capacity, (which depends on the fibre capacity C^f and the fibre modulation Wavelength Division Multiplexing (WDM)), O^f the fibre overhead and F^f the fibre Forward Error Correction (FEC)

Then, the cloud transmission rate can be calculated as in Eq. 13

$$t_i^{tC} = \frac{D_i}{r^f} = \frac{D_i}{\frac{C^f}{\sqrt{WDM}} \cdot (1-O^f)\cdot(1-F^f)} \tag{13}$$

Thus, the total transmission delay to the cloud server is given by the sum of the processing time of the cloud server, twice the propagation delay (uplink and downlink), and the fibre transmission delay:

$$\begin{aligned} t_i^{txC} &= t_i^{tx_m} + t_i^{tC} + 2\cdot t_i^{pC} = \\ &= \frac{D_i}{W^{nm}\log_2\left(1+\frac{p_i^n h_i^{nm}}{N_0}\right)} + \frac{D_i}{\frac{C^f}{\sqrt{WDM}}(1-O^f)(1-F^f)} + 2\frac{L_{mC}}{v} \end{aligned} \tag{14}$$

Therefore, the total experienced delay when offloading to the cloud server, considering that transmission queuing delays have not been taken into account, is given by Eq. 15:

$$\begin{aligned} t_i^c &= t_i^{cC} + t_i^{txC} = \\ &= \frac{\phi_i}{f_i^C} + \frac{D_i}{W^{nm}\log_2\left(1+\frac{p_i^n h_i^{nm}}{N_0}\right)} + \frac{D_i}{\frac{C^f}{\sqrt{WDM}}(1-O^f)(1-F^f)} + 2\frac{L_{mC}}{v} \end{aligned} \tag{15}$$

Energy consumption when offloading to the cloud server

The energy consumption when the decision is to offload to the cloud is the same as when the decision is to offload to the MEC server (Eq. 9). This is because a task must first go through the BS where the MEC server is located in order to get to the cloud server, which is reached through a fibre link and does not imply more energy consumption from the UE:

$$e_i^c = e_i^m = p_i^n \cdot t_i^{tx_m} = p_i^n \cdot \frac{D_i}{r_i^{nm}} = p_i^n \cdot \frac{D_i}{W^{nm}\log_2\left(1+\frac{p_i^n h_i^{nm}}{N_0}\right)} \tag{16}$$

Problem formulation and proposed approach

This section formally delves into the problem formulation and the optimization goal, also presenting the proposed DRL-based approach implemented as its solution.

Problem formulation

The problem described in this paper is modelled as a Markov Decision Process (MDP) model, employed to depict the decision-making process of a dynamic system where the environment may evolve randomly, leading to diverse decisions being made over time. At each stage, a thorough analysis of the current state S_i is conducted by each of the UEs, each of which has an instance of the developed decision algorithm, in order to determine the appropriate course of action. Thus, each agent tries to learn to optimize their decisions in a shared environment, so each agent's actions could affect the state of the environment and consequently the rewards of other agents.

Given that each device is running its own DRL algorithm and that they do not interact with each other, this can be considered an Multi-Agent Deep Reinforcement Learning (MA-DRL) scenario: agents do not coordinate their actions with each other, but they can influence one another's outcomes as they are part of a Multi-Agent (MA) system, resulting in a non-cooperative and non-competitive MA system.

State space

The defined state refers to the current information about the environment. In this case, the agents located in the UEs can see a full observation of it (except other UEs' status, which is not relevant), that is, they can see the whole information that defines the status of the environment. This information is likewise composed of the status of themselves (UE state), the status of their assigned MEC and cloud servers, the channel conditions, and the task class. Thereby, the state is denoted as the following 6-tuple vector space (Eq. 17):

$$s_i = \{b_i^n, R_i^n, R_i^m, R_i^C, h_i^{nm}, C_i\} \tag{17}$$

Action space

The action represents how an incoming task shall be executed. This can be expressed as $\alpha_i \in \{0, 1, 2\}$, with $\alpha_i = 0$ meaning local execution, $\alpha_i = 1$ meaning the offload to the m MEC server and $\alpha_i = 2$ meaning the offload to the cloud server C.

Reward function

The reward function used for this problem is equal to the Quality-of-Experience (QoE) value, which is defined as in Eq. 18. This calculation is based on the energy consumption experienced by an UE (e_i^{cl}, e_i^m or e_i^c), taking into account its remaining battery.

$$QoE_i = \begin{cases} -e_i^{cl}/B^n & \text{, if success and } \alpha = 0 \\ -e_i^{m}/B^n & \text{, if success and } \alpha = 1 \\ -e_i^{c}/B^n & \text{, if success and } \alpha = 2 \\ \eta & \text{, if fail} \end{cases} \tag{18}$$

In case a task does not see its requirements fulfilled, QoE value is equal to η, which corresponds to the task punishment applied in the case a task fails. As a remark, in our environment, we consider tasks with firm deadlines, i.e., the result of a task would not be of use anymore, but no catastrophic consequences would happen. That is why we used a penalty value η of the QoE when the deadlines were not met, regardless of the type of task involved, as seen in Eq. 18.

As told earlier, the final optimization problem is to maximize the total QoE (Eq. 18), resulting in the problem stated as in Eq. 19:

$$\begin{aligned} &\max_{i} \sum_{i=0}^{T-1} QoE_i \\ &\text{s.t.} \\ &C1: \quad \alpha_i \in \{0,1,2\} \quad &&, \forall i \in I \\ &C2: \sum_i^I f_i^n \le F_i^n &&, \forall i \in I, \alpha = 0 \\ &C3: \sum_i^I f_i^m \le F_i^m &&, \forall i \in I, \alpha = 1 \\ &C4: \sum_i^I f_i^C \le F_i^C &&, \forall i \in I, \alpha = 2 \end{aligned} \tag{19}$$

Where $C1$ is an action constraint that states that each task can only be executed locally or offloaded to the m MEC server; $C2 - C4$ resource constraints mean that the dedicated computation resources to execute task i cannot be higher than the total computation resources of the UE, the MEC and the cloud server respectively.

DRL-based proposed solution

Figure 2 shows the configuration of the proposed solution, an Actor-Critic (AC) model, which combines both policy gradient and value function methods, for what is composed of two separate Artificial Neural Network (ANN) structures: the actor-network and the critic-network, each of which is in turn composed of two Deep Neural Network (DNN)s.

The goal of value-based methods is to find an optimal value function, which is the expected return for each action in each state. The policy is obtained through the value function and determines the action to take in each state. These methods are efficient and guaranteed to converge, although they may not behave well with large or continuous action spaces. Policy-based methods, on the other hand, try to optimize the policy function without a value function, which can be useful if the action space is high-dimensional or continuous; however, these methods can suffer from high variance, leading to slower convergence. AC method combines the strengths of both approaches through its two components: the actor is responsible for deciding which action to take in each state, while the critic evaluates these actions and provides feedback to the actor [34].

Regarding the actor network, the one responsible for selecting actions based on the current state of the environment, it is composed of 2 hidden layers, each of which consists of 256 neurons. In both hidden layers, the Activation Function used to transform the summed weighted input from the node into the output value that will be

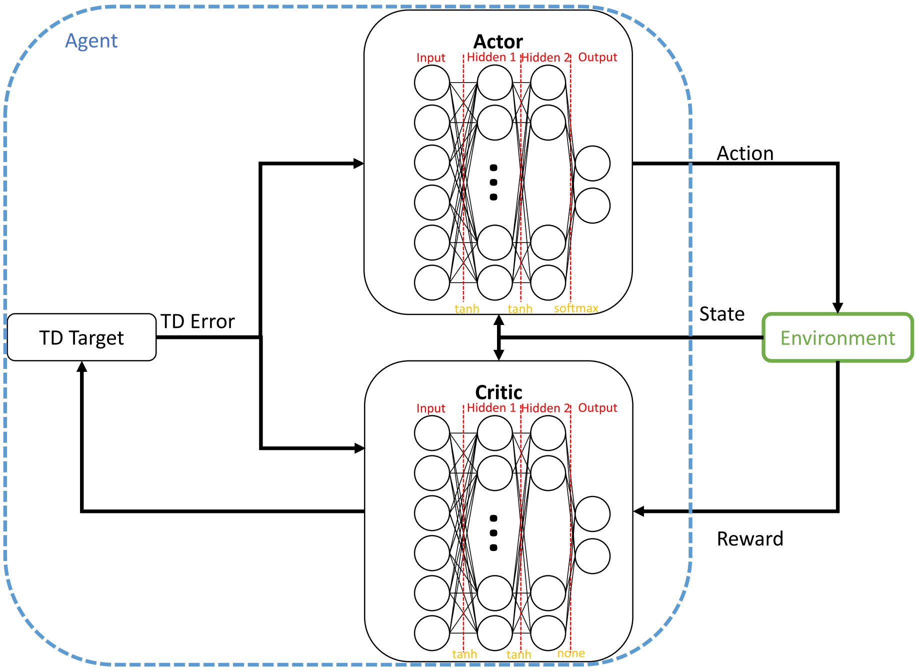

Fig. 2 Implemented AC model

fed to their respective output is the Hyperbolic tangent (TanH) activation function. Also, in order to prevent undesirable behaviours because of a bigger influence of any Deep Neural Network (DNN) weights at the start, these values are initialized using LeCun normalization. Finally, the employed activation function of the output layers is the softmax function, in order to compress the real values of a K-dimensional vector into the range [0, 1]. The learning rate for the actor network is set to 10^{-5}.

In respect of the Critic network, which estimates the value function or the expected future reward for taking a particular action in a given state, it is also composed of 2 hidden layers composed of 256 neurons that use the TanH activation function and Lecun initialization. Finally, in the output layer, a linear activation is used. The learning rate for the critic network is set to 10^{-4}.

This Actor-Critic agent uses the Temporal Difference (TD) error, the difference between the expected reward and the actual reward, to update both the actor and critic networks. This TD error is used to update the actor and critic networks using the Adam optimizer. Finally, the discount factor employed in this actor-critic algorithm is set to 0.99.

Simulation environment

This section shows the configuration of the experimental environment and the baselines that have been implemented to be compared with the proposal explained in DRL-based proposed solution section.

Simulation setup

The experiments are implemented in Python 3.10.13 and the simulations are carried out in a Ubuntu 20.04.3 LTS platform with a 4th Gen IntelCore i7 CPU 2.00 GHz and 32 GB of RAM.

The algorithm presented in DRL-based proposed solution section has been implemented using Tensorflow 2.14.0 ([2]), along with Keras 2.14.0 ([14]). The environment has been modelled using Gymnasium library ([40]) version 0.29.1, which is a maintained fork of OpenAI's deprecated Gym library ([8]).

Simulation parameters

First, the parameters of the environment are defined. As told earlier, the system consists of 3 layers: the *cloud* layer, composed of 1 *cloud* server C; the MEC layer, composed of 1 MEC server M; and the device layer, composed of multiple n UE devices ($n \in N$). Figures 3 and 4 show how each of these entities is located.

Table 2 shows every environment-related parameter, that is: UE-related, such as battery; MEC-related, such as BS transmission power; cloud-related, such as cloud computation resources; and communications-related parameters, such as the Additive White Gaussian noise (AWGN). Also, QoE-related parameters, such as the fail punish (η) in case a task is not correctly performed, are defined.

As mentioned in System model section, each computation task for which the associated UE needs to make

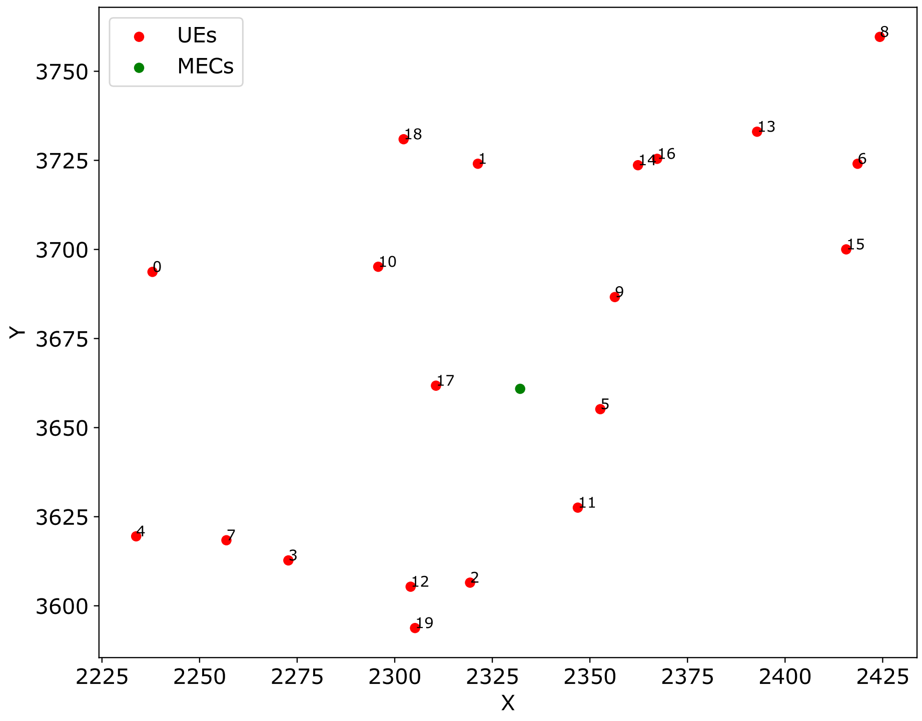

Fig. 3 UEs and MEC server distribution

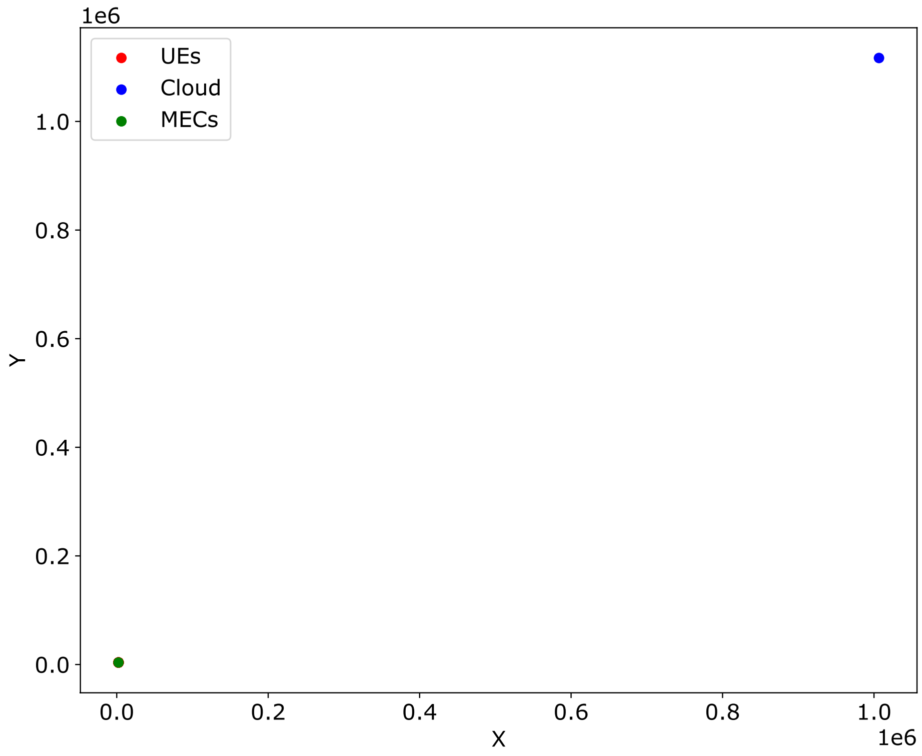

Fig. 4 UEs, MEC & cloud server distribution

Table 2 Environment parameters

	Parameter	Symbol	Value
MEC	Number of Servers	M	1
	Transmission power	P_{tx}^m	100 dBm
	Max. computation resources	F_i^m	1 GHz
UE	Number of UEs	N	20
	X distance to MEC	x	$0-100$ m
	Y distance to MEC	y	$0-100$ m
	Transmission power	P_{tx}^n	30 dBm
	Max. computation resources	F_i^n	40 MHz
	Energy coefficient	κ^n	10^{-21}
	Residual consumption each t	b_r^n	0.1 J
	Maximum battery	B^n	4000 J
Cloud	Number of Servers	S	1
	Max. computation resources	F_i^S	10 GHz
Wireless link	Total channel bandwidth	W	100 MHz
	AWGN	N_0	-100 dBm
Optical fiber	Capacity	C^f	100 Gbps
	Modulation	WDM	16-QAM
	Overhead	O^f	10%
	FEC	F^f	20%
	Propagation speed	v	$2 \cdot 10^8$
	Refractive index	ρ	1.5
Environment	Fail punishment	η	-0.1

a decision is classified into one of 3 classes (delay-sensitive, energy-sensitive, and insensitive). According to task size, latency requirements, and computational requirements, the tasks are included in one of the classes above, which are defined according to the parameters shown in Table 3. The task class, along with the UE, MEC server, cloud server, and channel conditions, will be the parameters the model will use to make the decision and maximize the QoE value defined in Problem formulation and proposed approach section.

Compared baselines

The method proposed in this paper, the AC approach explained in DRL-based proposed solution section, is compared against static and dynamic methods.

Table 3 Task parameters

Parameter	Symbol	$C_i = 1$	$C_i = 2$	$C_i = 3$
Task data size	D_i	$10-40KB$	$20-50KB$	$200-400KB$
Required CPU	ϕ_i	D_i	$8 \cdot D_i$	$8 \cdot D_i$
Latency constraints	T_i^{req}	$0.5 \cdot 10^{-3} D_i$	$10^{-3} D_i$	$2 \cdot 10^{-3} D_i$

The static methods entail making the same decision over time, regardless of the environmental conditions. These methods are **local**, which means running every task locally (for each UE n); **MEC**, offloading every task to the MEC server M; and **cloud**, which implies offloading every task to the cloud server C.

Regarding the dynamic methods, first, a **random** algorithm is proposed, choosing random actions without regard to the environment. All other methods presented in this paper are AI-based algorithms, and more specifically, DRL-based algorithms. Two different configurations of AC are presented, one of them being a single network approach, **Join-Networks AC**, where both the actor and the critic share their hidden layers; and the other one, the one proposed in this article and explained in DRL-based proposed solution section, in which the actor and the critic have their own distinct networks, **Separated-Networks AC**. Finally, a **Double Dueling DQN (DDDQN)** algorithm is presented, which combines Double Deep Q-Learning Network (DDQN) and Dueling Deep Q-Network (DQN), with the objectives of solving over-estimation, instability and convergence problems of DQN.

Discussion

This section presents the results of the evaluation. Three different metrics are used to compare the performance of each algorithm: the evolution of the QoE values; the remaining battery each timestep, so the battery consumption evolution can be tracked; and the decision evolution of the agents of the Separated Networks AC algorithm, so it is relatively easy to see the decisions made depending on the environment conditions.

Therefore, in the results figures included in this section (Figs. 7 & 10), these metrics are shown in different columns. Thus, the first column shows the QoE evolution. Actually, and in order to make the figures more easily interpretable, the average of the last 10 samples has been plotted every 10 steps. It is also important to note that the lines represent the average of the values obtained by every n UE. The second column shows the remaining battery each timestep, and, as the battery from every UE is considered, it also shows the average value of the remaining battery of each of them. The third column shows the evolution of the agents of the Separated Networks AC algorithm.

In order to check the performance of the Separated Networks AC algorithm, two different scenarios have been modelled, which will be explained in the following subsections.

Scenario 1: MEC unavailable for a certain time

In this scenario, the MEC server is available during a big part of the execution time, considering it is available when 80% − 100% of its computing resources are free to use.

As seen in Fig. 5, it is available between timesteps 0 − 500, 750 − 1250 and 1500 − 2000. However, there is an important service drop between timesteps 500 − 750 and 1250 − 1500, resulting in an unavailable MEC server.

Figure 6 shows the channel gain experienced between each UE (each UE with a different color) and the serving BS evolves normally considering it is a stable wireless communication, according to the explanation of its modelling in System model section.

For the results analysis in this scenario, which are shown in Fig. 7, the number of tasks for each class has been distributed in the following way:

In the first distribution (first row), 90% of incoming tasks are class 1 tasks, being the rest of the tasks of class 2 or 3. Here, cloud computation is giving the worst QoE results, as, according to the definition of Class 1 tasks, the latency experienced by these tasks when offloading to the cloud is bigger than the affordable values. Looking at Fig. 7a, regarding local computation, the QoE results seem to be near 0, which would be the ideal QoE value, but not as close to 0 as the results when offloading to the MEC server when it is available. However, in the second figure, it can be seen that the UE runs out of battery earlier when the tasks are executed locally (Fig. 7b). With respect to the dynamic approaches, both AC configurations and the DDDQN algorithm start with similar values to the random proposal, as in the early phases of the execution they are still exploring their options. Anyway, both AC algorithms seem to identify the bad results of cloud computation earlier, being the Separated-Networks-AC is the one that has the best results. Also, these AI-based algorithms get bad results when the MEC server becomes unavailable, although they identify it and change the decision after a few steps. This is reflected in Fig. 7c, where most of the UEs decide to offload the tasks to the MEC server or execute them locally until the MEC is not available, the moment from when they decide to execute almost every task locally. In the last part of the QoE figure, the values are lower because many UEs run out of battery, so the QoE associated with each task turns η.

In the next case, where 90% of incoming tasks are Class 2 tasks, Fig. 7d shows that the worst QoE results are obtained when they are executed locally. Besides, the battery runs out earlier than when performing an offload (Fig. 7e). Concerning the offloading options, offloading to the MEC server seems to be a better decision, due to the existence of some Class 1 tasks. The battery consumption in both cases is the same, as there is no extra energy consumption in the UE when the task is sent from the entities in the MEC layer to the one in the cloud layer.

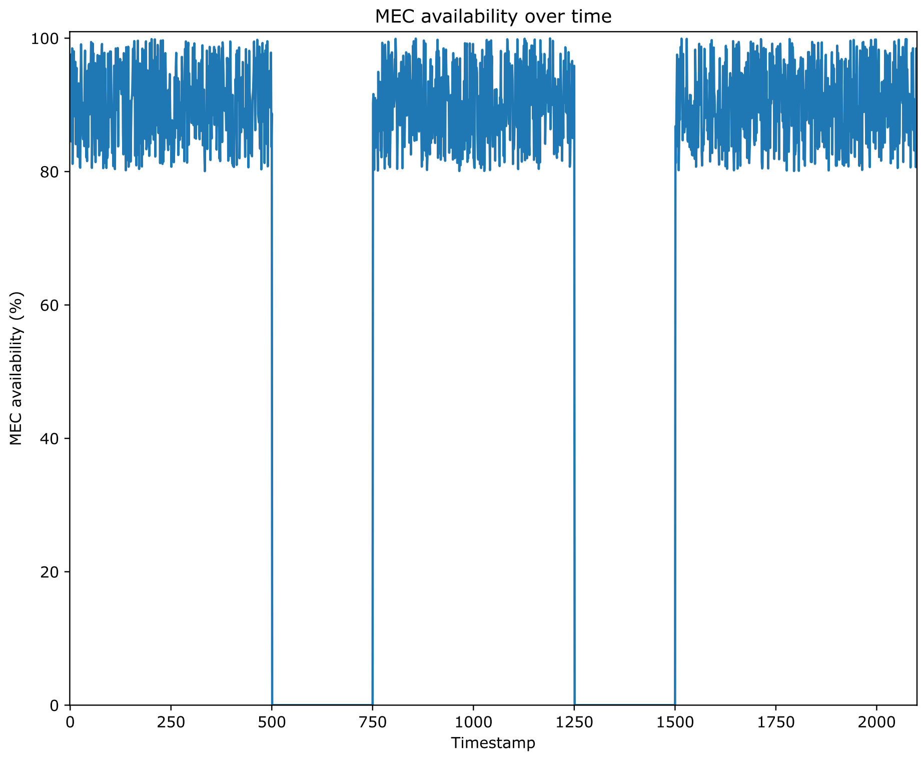

Fig. 5 MEC status in Scenario 1

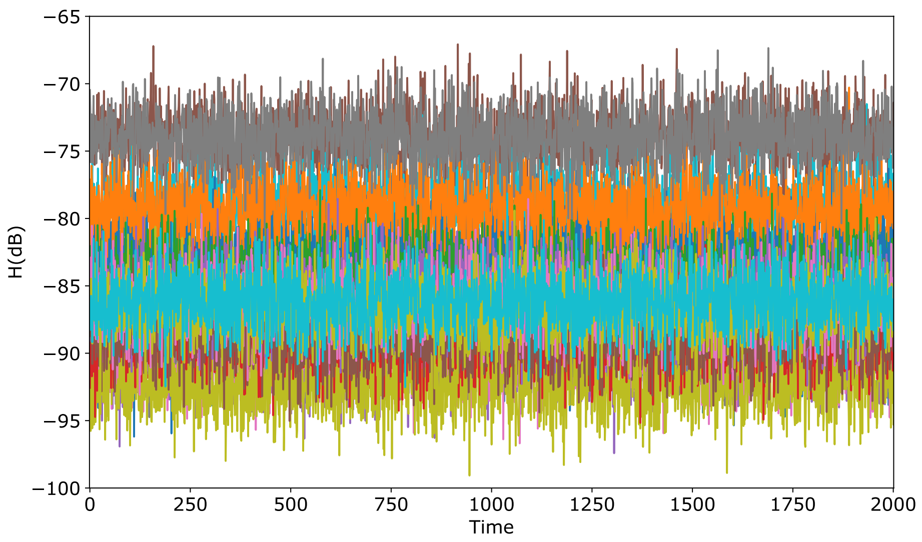

Fig. 6 Channel gains in Scenario 1

As in the previous case, the DRL-based solutions have similar QoE values to random in the beginning, but they improve over time. Both AC configurations get similar QoE values, being the Separated-Networks-AC slightly better. The DDDQN algorithm, learns slower but has better values in certain periods (steps 1000 − 1250). When the MEC server fails the first time, the AI-based algorithms get their results negatively affected, as they have not experienced that casuistry before. The second time, in contrast, they have already learnt that QoE degradation is not that bad regarding the AC algorithms. The explanation for this can easily be seen in the proposed AC algorithm's decisions figure, Fig. 7f, where the algorithms decide to send the tasks to the cloud server in order to be performed. Finally, in the Figure corresponding to the battery, it can also be seen that the battery consumption is not very big in AI proposals, similar to the offloading options.

Regarding the 90%-Class 3 tasks distribution, as they do not have strict latency requirements, every static proposal achieves high QoE values as seen in Fig. 7g, although local computation gets the worst QoE values mainly because of two reasons, which are the existence of Class-2 tasks and the fact that the energy consumption is bigger in this case (which affects the obtained QoE value). Besides, after very few steps, there is no battery left in the UEs (Fig. 7h). Both cloud and MEC proposals get QoE values near to 0, except MEC offloading when the MEC server is not available. The fact of computing a Class 3 task locally, implies lowering the remaining battery level considerably, so, on the one hand, random proposal also gets UEs' battery drained quickly, and, on the other hand, the first part of the DRL algorithms (where exploration is bigger) make some UEs lose their batter, as it can be seen in Fig. 7h and i). In fact, in DDDQN algorithm curve, as with this proposal, the agents do not learn as fast as with its AC competitors, and there is no UE with battery left around step 600. Finally, it can be seen that QoE values are better in the separated AC algorithm, as there are more UEs with remaining battery. It can also be seen that more agents choose MEC offloading with this configuration as it has better results until the time the MEC server is unavailable, the moment from which every agent decides to offload nearly everything to the cloud server during the simulation.

Finally, in the case where the distribution of tasks is random, MEC offloading decision seems to be the best decision among the static decisions, as local computing fails with Class 2 tasks and Cloud computing fails with Class 1 tasks, resulting in bad QoE values, according to Fig. 7j. The random proposal also experiences these problems, along with the early battery consumption (Fig. 7k) consequence of running tasks locally. Focusing on the AI based solutions, DDDQN proposal gets the worst values, in terms of battery and QoE. The AC proposal with the separated networks gets the best QoE results in general. This can be understood with Fig. 7l, which clearly shows that the agents begin to send the maximum number of tasks to the MEC server, which is the configuration that gives the best QoE values, except in the moments the MEC server is not available. In those moments, the agents gradually decide not to send tasks to the MEC server, but to run these tasks locally or send them to the cloud server. Anyway, these local execution decisions imply more energy consumption than the offloading, as can be seen in the second figure.

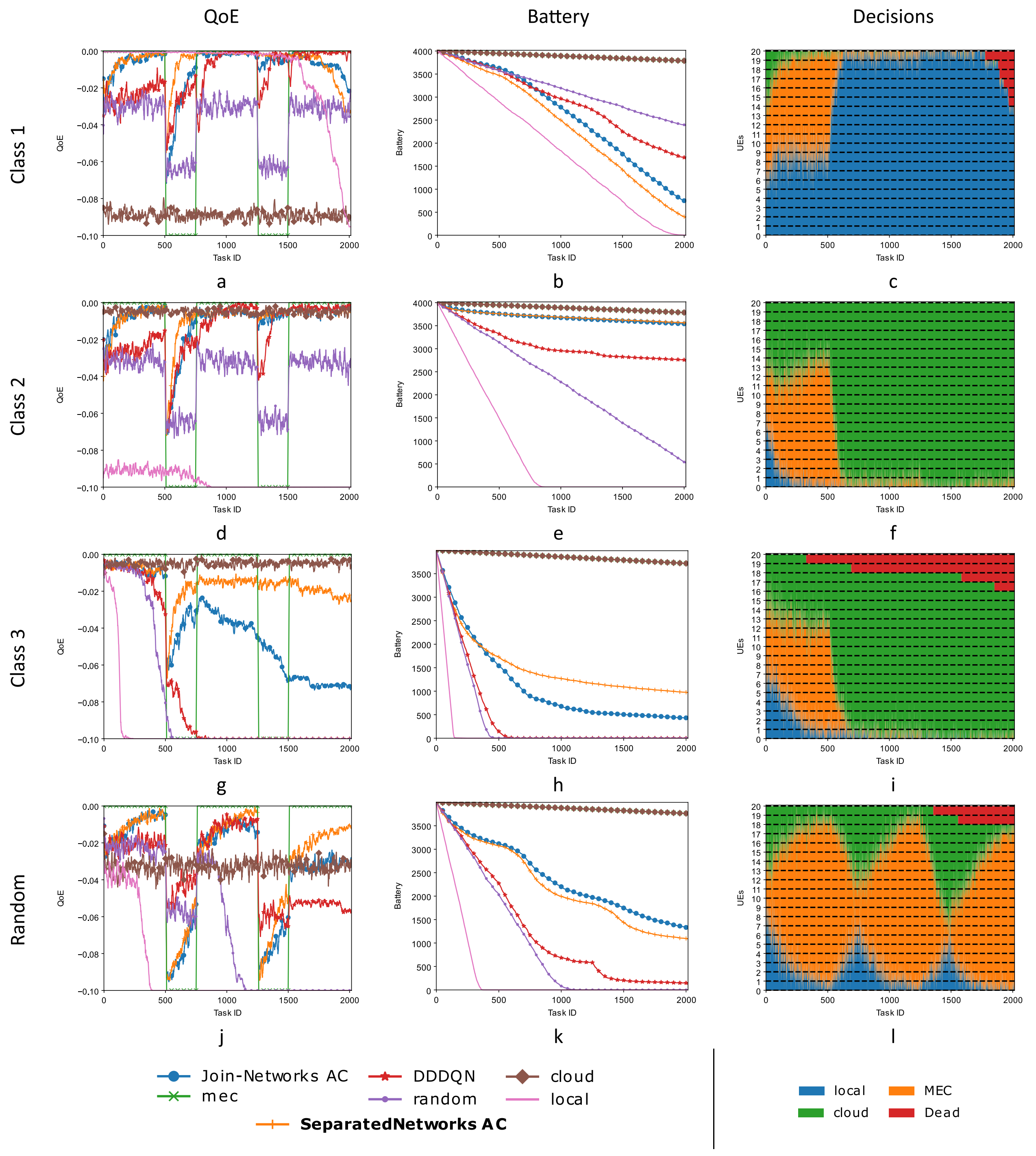

Fig. 7 Scenario 1 results

Scenario 2: Communications failure

This scenario is characterized by a stable MEC server availability, which has always between 80% and 100% of its computational resources available. However, the wireless communication link disappears between tasks 500 and 1000, making it impossible for the UEs to reach both the MEC server and the cloud server, which are only reachable through the BS with which there is connectivity. This situation is represented in Figs. 8 and 9 respectively.

As in the previous scenario, Fig. 10 shows the 4 class distributions that have been considered (90% Class 1

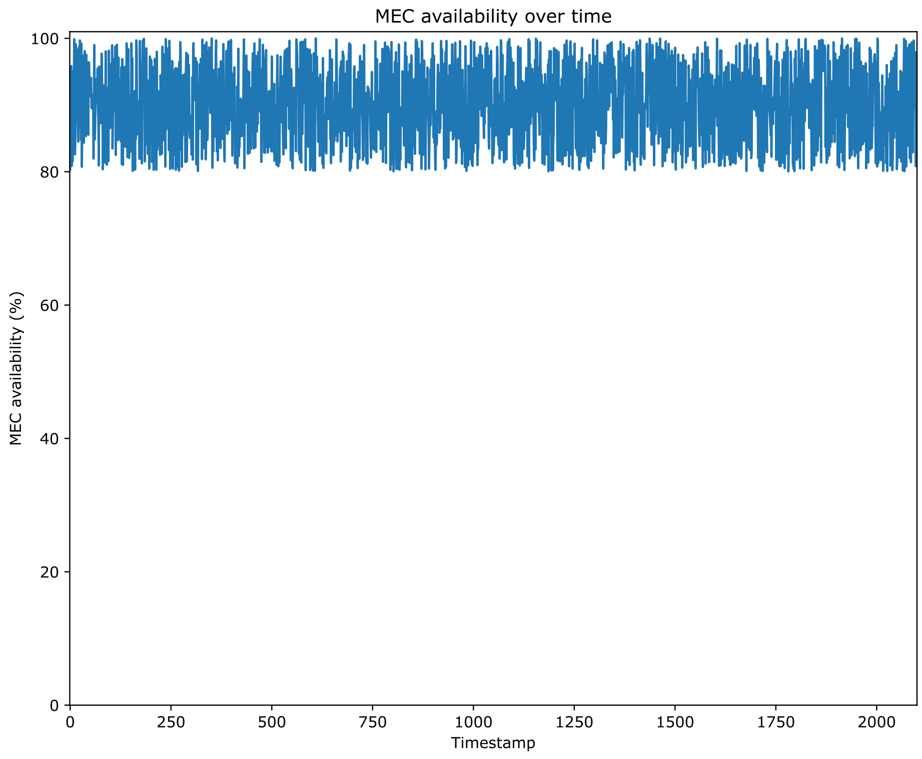

Fig. 8 MEC status in Scenario 2

tasks, 90% Class 2 tasks, 90% Class 3 tasks, and random distribution).

When there is a majority of Class 1 tasks, as seen earlier, offloading to the cloud server gives the worst QoE results, as the experienced latency is too big for these tasks. Local computation and offloading to the MEC server return high QoE values (Fig. 10a). Also, the UE consumes more battery when the tasks are executed locally (Fig. 10b). AC algorithms behave similarly and the DDDQN algorithm makes more erratic decisions, but each of them identifies the bad results of cloud computation and minimizes this decision. When the communications fail, offloading to the MEC server is not a good decision, as it will not reach it, so the

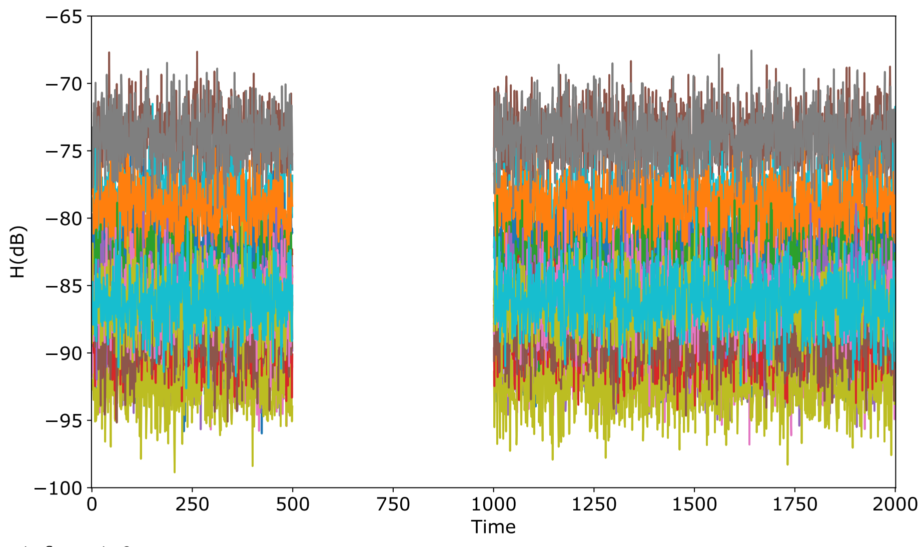

Fig. 9 Channel gains in Scenario 2

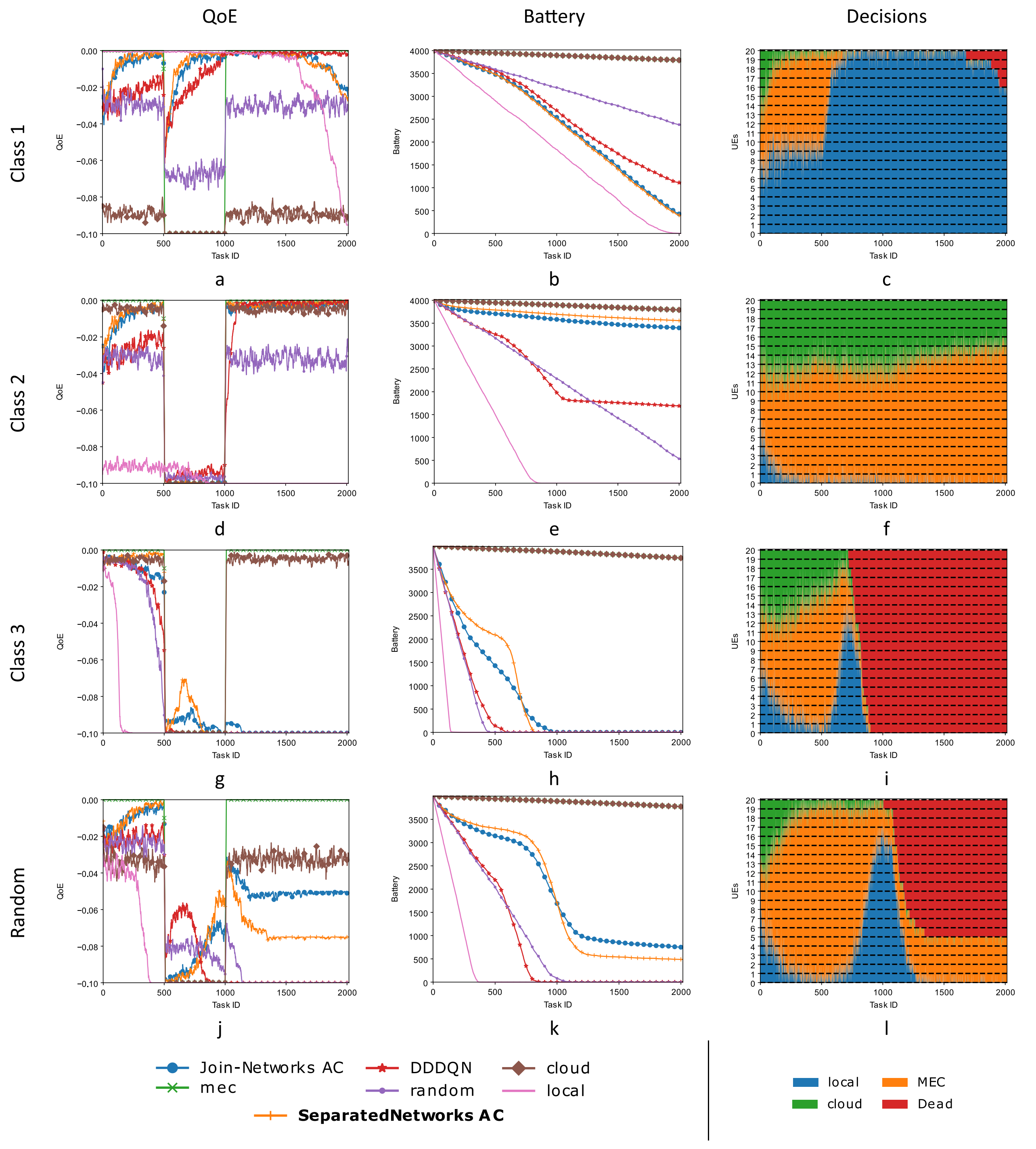

Fig. 10 Scenario 2 results

separated-network AC algorithm decides to execute locally from that moment on, as seen in Fig. 10c.

Regarding the case where 90% of the incoming tasks are Class 2 tasks, and according to their definition, QoE results when executed locally are bad (Fig. 10d), as no Class 2 task fulfils its requirements, while the battery runs out before the 1000-th timestep (Fig. 10e). Offloading to the MEC server, in general, returns better results than cloud computation. However, when there is no connectivity, the offloading options will not return a good QoE value, as every task will fail. In this case, there is no good choice for the algorithm for Class 2 tasks, as they will fail with any of the 3 possible options. The proposed algorithm, as seen in Fig. 10f, will keep choosing MEC and cloud offloading decisions because local computation will fail too, but the offloading will not require so much battery. When the connectivity comes back, this figure also shows that the decisions gradually become MEC server offloading, as the QoE values that way are slightly better than offloading to the cloud.

In the "majority of class 3 tasks" case, in the beginning, every proposal gets high QoE values due to their lax requirements. However, local computation gets the worst values as explained in the first scenario and seen in Fig. 10g. TheAC proposals begin to decide to prioritize MEC and cloud offloading (Fig. 10i), but DDDQN needs more time for this, so the UEs run out of battery earlier. Anyway, when communications get unavailable in timestep 500 (and until timestep 1000), the proposed AC algorithm starts to decide to execute the tasks locally, as it is the only way they can be correctly executed. This implies a big battery consumption, as can be seen in Fig. 10h, but does fulfil tasks' requirements.

Last, when there is a 33% of occurrences of each Class of tasks, offloading to the MEC is the best decision among the static decisions in terms of QoE (Fig. 10j), as local computing fails with Class 2 tasks and Cloud computing fails with Class 1 tasks. The proposed AC algorithm determines to send data to the MEC server progressively until there is no possible communication with the BS. When this happens, and even though the response is not quick due to the lack of sufficient experience, the algorithms decide to run the tasks locally as soon as there are no communications (Fig. 10l), resulting in battery depletion for many of the UEs (Fig. 10k).

Conclusions and future work

The MEC paradigm offers the advantage of having computational capacity close to where tasks originated (in case latency is critical), as well as the capabilities of an external server as in CC paradigm, to overcome local resources limitation. However, due to the dynamic nature of channel and servers' availability, offloading to the MEC server may not always be the best option. To overcome this, this work has presented an AI-based algorithm, specifically an AC algorithm, which adapts to the varying scenarios.

In order to test the performance of this algorithm, two different scenarios have been analyzed. In the first one, the MEC server is temporally unavailable, so every task sent to the MEC server fails. The algorithm is able to identify when this happens and decides to change its decision to improve the QoE. The algorithm is also able to decide to send it to a cloud server if the tasks can be processed before their deadline, or it decides to execute the tasks locally if the latency constraints are critical, trying to maximize the QoE value, which is related to the UE's remaining battery level. The second scenario, unlike the first one, is characterized by a stable MEC server availability, but in which the wireless communication link disappears, so the UEs cannot reach MEC nor cloud servers. In this case, the algorithm decides to run the tasks locally, although it implies a bigger battery consumption and lower QoE values than it would get by offloading to an external working entity. The algorithm can also see that trying to execute big tasks with strong latency requirements locally will also mean failed tasks, along with big battery consumption, so it decides to keep the offloading decision, in order not to consume energy.

Thereby, AC algorithms have been proven to choose the proper action and stabilize their learning quickly, as a consequence of combining value-based and policy-based methods, reducing the variance in policy gradient estimation with the value estimation, this is, leveraging the strengths of both policy-based and value-based methods. That is why the present study demonstrates the suitability of DRL-based algorithms for task offloading optimization in environments characterized by real-time variability.

As future work, the proposed algorithms could be improved by adding adaptive hyperparameters, so they react earlier and more effectively to the changes in the environment. For example, the learning performance with a constant discount factor can be limited when uncertainties are involved in the training. Thus, having an adaptive Discount Factor could lead to better learning performance and adapt itself when results (in our case, QoE value) are wrong. In the same way as the discount factor, with an adaptive learning rate such as a cyclical learning rate, which consists of varying the learning rate cyclically between two boundary values [20], there would be no need to find the best values and schedule for the global learning rates, so accuracy could be improved in fewer iterations.

Also, in the proposed framework, the algorithms must choose between 3 options, being local execution, offloading to the attached MEC server, or offloading to the cloud server. Thus, in this scenario, there is no possibility of sending to another MEC server, which could also be a good option in case the nearest MEC server is overflown. Another possibility would be to make other kinds of decisions, such as regulating the transmission power of the UE or selecting the quantity of their computational resources, to improve their battery life while fulfilling tasks' requirements.

A partial offloading scenario could also be considered, where computing tasks could be partitioned, so the agents choose an action for each of these parts, considering a task-related dependency, as the functional progress of a task may depend on the completion of one or more subtasks. For this, each UE's incoming apps should be defined as a composition of several dependent computing tasks, being modelled as DAGs that would represent the tasks and the dependency constraints between them.

Another case study could be the real implementation of this algorithm in a resource-constrained device, such as a Jetson nano. There are two key aspects to consider to run this kind of algorithm in resource-constrained devices: the time the device needs to run the algorithm and the consumption it implies. Thus, depending on the results of implementing this algorithm, different techniques such as pruning shall be studied to reduce the algorithms while keeping allowable results. This actual implementation could also present another dilemma to be studied, being the training-inference trade-off, or, more specifically in RL, the exploration-exploitation trade-off, which can be critical in terms of execution time.

Abbreviations

ABG	Alpha-Betta-Gamma
AC	Actor-Critic
A2C	Advantage Actor-Critic
ADMM	Alternating Direction Method of Multipliers
AI	Artificial Intelligence
ANN	Artificial Neural Network
AR	Augmented Reality
AWGN	Additive White Gaussian noise
BS	Base Station
CC	Cloud Computing
CNN	Convolutional Neural Network
DAG	Dynamic Acyclic Graph
DDPG	Deep Deterministic Policy Gradient
DDQN	Double Deep Q-Learning Network
DDDQN	Double Dueling DQN
DL	Deep Learning
DQN	Deep Q-Network
FEC	Forward Error Correction
DBN	Deep Belief Network
DNN	Deep Neural Network
DRL	Deep Reinforcement Learning
ETSI	European Telecommunications Standards Institute
FF	Fast Fading
GA	Genetic Algorithm
GPU	Graphics Processing Unit
IIoT	Industrial Internet of Things
IoT	Internet of Things
IPLC	Improved Policy Loss Clip
LSTM	Long short-term memory
MA	Multi-Agent
MA-DRL	Multi-Agent Deep Reinforcement Learning
MCC	Mobile Cloud Computing
MDP	Markov Decision Process
MEC	Multi-access Edge Computing
MINLP	Mixed-Integer Nonlinear Programming
ML	Machine Learning
NR	New Radio
OFDMA	Orthogonal Frequency-Division Multiple Access
PL	Path Loss
PO-MDP	Partially Observable Markov Decision Process
PPO	Proximal Policy Optimization
QDRL	Deep Reinforcement Learning based on Queuing theory
QoE	Quality-of-Experience
QoS	Quality-of-Service
RA	Resource Allocation
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RTT	Round-Trip Time
SAC	Soft Actor-Critic
SF	Shadow Fading
SH	Sparse clutter High Base Station
TanH	Hyperbolic tangent
TD	Temporal Difference
TO	Task Offloading
UE	User Equipment
UX	User Experience
Wi-Fi	Wireless-Fidelity
WDM	Wavelength Division Multiplexing

Acknowledgements
Authors would like to thank anonymous reviewers for their insightful comments that have enriched the manuscript.

Authors' contributions
GN: Conceptualization, Methodology, Software, Validation, Formal Analysis, Investigation, Data Curation, Writing - Original Draft, Visualization. II: Conceptualization, Supervision, Validation, Writing - Review & Editing. UL: Conceptualization, Supervision, Validation, Writing - Review & Editing. CP: Conceptualization, Supervision, Validation, Writing - Review & Editing.

Funding
This work was partially supported by the Basque Government under the Elkartek funding program through the project SONETO: La red social de los Activos (grant KK-2023/00038) and the Spanish Ministerio de Asuntos Económicos y Transformación Digital and the European Union NextGenerationEU through the project LocoForge: Mimbres instantiation for railways and Industry 5.0 vertical sectors (grant TSI-063000- 2021-47), funded by the Plan for Recovery, ransformation and Resilience.

Availability of data and materials
No datasets were generated or analysed during the current study.

Declarations

Competing interests
The authors declare no competing interests.

Received: 7 February 2024 Accepted: 24 April 2024
Published online: 03 May 2024

References
1. 3GPP (2020) Study on channel model for frequencies from 0.5 to 100 ghz. Technical report (tr), 3rd Generation Partnership Project (3GPP). version 16.1.0. https://www.etsi.org/deliver/etsi_tr/138900_138999/138901/16.01.00_60/tr_138901v160100p.pdf
2. Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viégas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y, Zheng X (2015) TensorFlow: Large-scale machine learning on heterogeneous systems. https://www.tensorflow.org/. Accessed 26 Mar 2024
3. Abdullaev I, Prodanova N, Bhaskar KA, Lydia EL, Kadry S, Kim J (2023) Task offloading and resource allocation in iot based mobile edge computing using deep learning. Comput Mater Continua 76(2). https://doi.org/10.32604/cmc.2023.038417
4. Al Aidaros O, Kardjadja Y, Bouida Z, Ibnkahla M (2023) Energy and time-effective computation offloading for edge computing-enabled iot

networks. In: 2023 IEEE Sensors Applications Symposium (SAS), pp 1–6. https://doi.org/10.1109/SAS58821.2023.10254051
5. Avgeris M, Mechennef M, Leivadeas A, Lambadaris I (2023) A two-stage cooperative reinforcement learning scheme for energy-aware computational offloading. In: 2023 IEEE 24th International Conference on High Performance Switching and Routing (HPSR), pp 179–184. https://doi.org/10.1109/HPSR57248.2023.10147932
6. Baccarelli E, Scarpiniti M, Momenzadeh A (2019) Ecomobifog-design and dynamic optimization of a 5g mobile-fog-cloud multi-tier ecosystem for the real-time distributed execution of stream applications. IEEE Access 7:55565–55608. https://doi.org/10.1109/ACCESS.2019.2913564
7. Bi S, Huang L, Wang H, Zhang YJA (2021) Stable online computation offloading via lyapunov-guided deep reinforcement learning. In: IEEE ICC, pp 1–7. https://doi.org/10.1109/ICC42927.2021.9500520
8. Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, Zaremba W (2016) OpenAI Gym. arXiv:1606.01540
9. Carbone MR (2022) When not to use machine learning: A perspective on potential and limitations. MRS Bull 47(9):968–974. https://doi.org/10.1557/s43577-022-00417-z
10. Chen C, Zeng Y, Li H, Liu Y, Wan S (2023) A multihop task offloading decision model in mec-enabled internet of vehicles. IEEE Internet Things J 10(4):3215–3230. https://doi.org/10.1109/JIOT.2022.3143529
11. Chen X, Liu G (2021) Energy-efficient task offloading and resource allocation via deep reinforcement learning for augmented reality in mobile edge networks. IEEE Internet Things J 8(13):10843–10856. https://doi.org/10.1109/JIOT.2021.3050804
12. Chen X, Wu C, Liu Z, Zhang N, Ji Y (2021) Computation offloading in beyond 5g networks: A distributed learning framework and applications. IEEE Wirel Commun 28(2):56–62. https://doi.org/10.1109/MWC.001.2000296
13. Cho B, Xiao Y (2021) Learning-based decentralized offloading decision making in an adversarial environment. IEEE Trans Veh Technol 70(11):11308–11323. https://doi.org/10.1109/TVT.2021.3115899
14. Chollet F, et al (2015) Keras. https://keras.io. Accessed 26 Mar 2024
15. Cozzolino V, Tonetto L, Mohan N, Ding AY, Ott J (2023) Nimbus: Towards latency-energy efficient task offloading for ar services. IEEE Trans Cloud Comput 11(2):1530–1545. https://doi.org/10.1109/TCC.2022.3146615
16. Dong Y, Alwakeel AM, Alwakeel MM, Alharbi LA, Althubiti SA (2023) A heuristic deep q learning for offloading in edge devices in 5 g networks. J Grid Comput 21(3):37. https://doi.org/10.1007/s10723-023-09667-w
17. Dulac-Arnold G, Levine N, Mankowitz DJ, Li J, Paduraru C, Gowal S, Hester T (2021) Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. Mach Learn 110(9):2419–2468. https://doi.org/10.1007/s10994-021-05961-4
18. ETSI (2024) Multi-access edge computing (mec). https://www.etsi.org/technologies/multi-access-edge-computing. Accessed 26 Jan 2024
19. Farhan L, Kharel R, Kaiwartya O, Quiroz-Castellanos M, Alissa A, Abdulsalam M (2018) A concise review on internet of things (iot) -problems, challenges and opportunities. In: 2018 11th International Symposium on Communication Systems, Networks & Digital Signal Processing (CSNDSP), pp 1–6. https://doi.org/10.1109/CSNDSP.2018.8471762
20. Gulde R, Tuscher M, Csiszar A, Riedel O, Verl A (2020) Deep reinforcement learning using cyclical learning rates. In: 2020 Third International Conference on Artificial Intelligence for Industries (AI4I), IEEE, pp 32–35. https://doi.org/10.1109/AI4I49448.2020.00014
21. Hou J, Wu Y, Cai J, Zhou Z (2023) Qoe-guaranteed distributed offloading decision via partially observable deep reinforcement learning for edge-enabled internet of things. Neural Comput Applic 35(29):21603–21619. https://doi.org/10.1007/s00521-023-08905-2
22. Huang L, Bi S, Zhang YJA (2020) Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks. IEEE Trans Mob Comput 19(11):2581–2593. https://doi.org/10.1109/TMC.2019.2928811
23. Jiang T, Zhang J, Tang P, Tian L, Zheng Y, Dou J, Asplund H, Raschkowski L, D'Errico R, Jämsä T (2021) 3g pp standardized 5g channel model for IIOT scenarios: A survey. IEEE Internet Things J 8(11):8799–8815. https://doi.org/10.1109/JIOT.2020.3048992
24. Jiao X, Ou H, Chen S, Guo S, Qu Y, Xiang C, Shang J (2023) Deep reinforcement learning for time-energy tradeoff online offloading in mec-enabled industrial internet of things. IEEE Trans Netw Sci Eng 1–14. https://doi.org/10.1109/TNSE.2023.3263169
25. Khan BS, Jangsher S, Ahmed A, Al-Dweik A (2022) Urllc and embb in 5g industrial iot: A survey. IEEE Open J Commun Soc 3:1134–1163. https://doi.org/10.1109/OJCOMS.2022.3189013
26. Khanna A, Kaur S (2020) Internet of things (iot), applications and challenges: A comprehensive review. Wirel Pers Commun 114(2):1687–1762. https://doi.org/10.1007/s11277-020-07446-4
27. Kovacevic I, Harjula E, Glisic S, Lorenzo B, Ylianttila M (2021) Cloud and edge computation offloading for latency limited services. IEEE Access 9:55764–55776. https://doi.org/10.1109/ACCESS.2021.3071848
28. Kwak J, Kim Y, Lee J, Chong S (2015) Dream: Dynamic resource and task allocation for energy minimization in mobile cloud systems. IEEE J Sel Areas Commun 33(12):2510–2523. https://doi.org/10.1109/JSAC.2015.2478718
29. Lai P, He Q, Abdelrazek M, Chen F, Hosking J, Grundy J, Yang Y (2018) Optimal edge user allocation in edge computing with variable sized vector bin packing. In: Pahl C, Vukovic M, Yin J, Yu Q (eds) Service-Oriented Computing. Springer International Publishing, Cham, pp 230–245. https://doi.org/10.1007/978-3-030-03596-9_15
30. Li H, Xiong K, Fan P, Letaief KB (2023) Deep reinforcement learning based task offloading and resource allocation in small cell mec. In: 2023 IEEE International Performance, Computing, and Communications Conference (IPCCC), pp 475–480. https://doi.org/10.1109/IPCCC59175.2023.10253839
31. Lin L, Zhou W, Yang Z, Liu J (2023) Deep reinforcement learning-based task scheduling and resource allocation for noma-mec in industrial internet of things. Peer-to-Peer Netw Appl 16(1):170–188. https://doi.org/10.1007/s12083-022-01348-x
32. Mitsis G, Tsiropoulou EE, Papavassiliou S (2022) Price and risk awareness for data offloading decision-making in edge computing systems. IEEE Syst J 16(4):6546–6557. https://doi.org/10.1109/JSYST.2022.3188997
33. Pan M, Li Z, Qian J (2023) Energy-efficient multiuser and multitask computation offloading optimization method. Intell Converged Netw 4(1):76–92. https://doi.org/10.23919/ICN.2023.0007
34. Plaat A (2022) Deep reinforcement learning, vol 10. Springer. https://link.springer.com/content/pdf/10.1007/978-981-19-0638-1.pdf
35. Saeed MM, Saeed RA, Mokhtar RA, Khalifa OO, Ahmed ZE, Barakat M, Elnaim AA (2023) Task reverse offloading with deep reinforcement learning in multi-access edge computing. In: 2023 9th International Conference on Computer and Communication Engineering (ICCCE), IEEE, pp 322–327. https://doi.org/10.1109/ICCCE58854.2023.10246081
36. Scarpiniti M, Baccarelli E, Momenzadeh A (2019) Virtfogsim: A parallel toolbox for dynamic energy-delay performance testing and optimization of 5g mobile-fog-cloud virtualized platforms. Appl Sci 9(6). https://doi.org/10.3390/app9061160
37. Silva C, Magaia N, Grilo A (2023) Task offloading optimization in mobile edge computing based on deep reinforcement learning. In: Proceedings of the Int'l ACM Conference on Modeling Analysis and Simulation of Wireless and Mobile Systems, Association for Computing Machinery, pp 109–118. https://doi.org/10.1145/3616388.3617539
38. Song Y, Shen Y (2023) Computing offloading based on deep reinforcement learning for virtual reality scene. In: 2023 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB), pp 1–5. https://doi.org/10.1109/BMSB58369.2023.10211194
39. Sun X, Chen J, Guo C (2022) Semantic-driven computation offloading and resource allocation for uav-assisted monitoring system in vehicular networks. In: IECON 2022-48th Annual Conference of the IEEE Industrial Electronics Society, IEEE, pp 1–6. https://doi.org/10.1109/IECON49645.2022.9969083
40. Towers M, Terry JK, Kwiatkowski A, Balis JU, Cola Gd, Deleu T, Goulão M, Kallinteris A, KG A, Krimmel M, Perez-Vicente R, Pierré A, Schulhoff S, Tai JJ, Shen ATJ, Younis OG (2023) Gymnasium. https://doi.org/10.5281/zenodo.8127026
41. Tran TX, Pompili D (2019) Joint task offloading and resource allocation for multi-server mobile-edge computing networks. IEEE Trans Veh Technol 68(1):856–868. https://doi.org/10.1109/TVT.2018.2881191
42. Wang S, Zafer M, Leung KK (2017) Online placement of multi-component applications in edge computing environments. IEEE Access 5:2514–2533. https://doi.org/10.1109/ACCESS.2017.2665971
43. Wu G, Xu Z, Zhang H, Shen S, Yu S (2023) Multi-agent drl for joint completion delay and energy consumption with queuing theory in mec-based

iiot. J Parallel Distrib Comput 176:80–94. https://doi.org/10.1016/j.jpdc.2023.02.008
44. Xu A, Hu Z, Zhang X, Xiao H, Zheng H, Chen B, Zheng M, Zhong P, Kang Y, Li K (2023) Qdrl: Queue-aware online drl for computation offloading in industrial internet of things. IEEE Internet Things J. https://doi.org/10.1109/JIOT.2023.3316139
45. Xu J, Yang D (2023) Optimal task offloading for edge computing with stochastic task arrivals. In: 2023 IEEE International Performance, Computing, and Communications Conference (IPCCC), pp 24–31. https://doi.org/10.1109/IPCCC59175.2023.10253860
46. Yuan P, Shao S, Zhang J, Zhao X (2023) Cooperative edge offloading strategy for sensory data with delay and energy constraints. Wirel Netw 29(8):3469–3478. https://doi.org/10.1007/s11276-023-03404-7
47. Zhang B, Xiao F, Wu L (2023) Offline reinforcement learning for asynchronous task offloading in mobile edge computing. IEEE Trans Netw Serv Manag. https://doi.org/10.1109/TNSM.2023.3316626
48. Zhou H, Jiang K, Liu X, Li X, Leung VC (2021) Deep reinforcement learning for energy-efficient computation offloading in mobile-edge computing. IEEE Internet Things J 9(2):1517–1530. https://doi.org/10.1109/JIOT.2021.3091142

Publisher's Note