



# A multi-layer guided reinforcement learning-based tasks offloading in edge computing

Alberto Robles-Enciso\*, Antonio F. Skarmeta

Department of Information and Communications Engineering, University of Murcia, Murcia, 30100, Murcia, Spain

## ARTICLE INFO

### Keywords:

Internet of Things  
Fog computing  
Edge computing  
Task offloading  
Resource allocation  
Markov decision process  
Reinforcement learning  
Q-learning

## ABSTRACT

The breakthrough in Machine Learning (ML) techniques and the popularity of the Internet of Things (IoT) has increased interest in applying Artificial Intelligence (AI) techniques to the new paradigm of Edge Computing. One of the challenges in edge computing architectures is the optimal distribution of the generated tasks between the devices in each layer (i.e., cloud-fog-edge). In this paper, we propose to use Reinforcement Learning (RL) to solve the Task Assignment Problem (TAP) at the edge layer and then we propose a novel multi-layer extension of RL (ML-RL) techniques that allows edge agents to query an upper-level agent with more knowledge to improve the performance in complex and uncertain situations. We first formulate the task assignment process considering the trade-off between energy consumption and execution time. We then present a greedy solution as a baseline and implement our multi-layer RL proposal in the PureEdgeSim simulator. Finally several simulations of each algorithm are evaluated with different numbers of devices to verify scalability. The simulation results show that reinforcement learning solutions outperformed the heuristic-based solutions and our multi-layer approach can significantly improve performance in high device density scenarios.

## 1. Introduction

We are currently experiencing one of the most important changes in our history, the beginning of digitalization of our entire society, bringing the Internet to everything around us. IoT and mobile technologies could be considered part of the devices of the future and all technology will evolve to adapt to these changes. Alongside IoT, other useful technologies are emerging that will be an important foundation for a fully connected future, for example, machine learning will enable mobile devices networks to provide “intelligence” to be able to automatically process their data, draw conclusions from it and even take security measures if they detect a cyberattack.

Even so, all innovative progress come with challenges, and IoT and mobile technologies are no exception, as discussed in [1,2]. One of them is the latency problem, some mobile applications, such as autonomous driving [3] or virtual reality [4], need to meet a maximum delay requirement so in some cases it is not possible to send the data to the cloud [5]. The edge computing paradigm addresses this problem by providing a distributed data processing network near to the users, enabling latency-critical applications and reducing bandwidth usage and congestion of cloud servers.

However, to make efficient use of devices in edge computing architecture, it is necessary to use a computational offloading framework

that optimizes the allocation of tasks to computing devices in the best possible way. Our work will address this topic in which each edge device receives tasks with specific requirements and has to decide whether to perform the computation itself or offload the task to another edge node or the cloud.

The procedure for deciding the best allocation of tasks to devices is called “task assignment problem” and is a combinatorial optimization problem defined as the process of determining where the computation of each task is performed in order to minimize certain parameters, such as the aforementioned latency and energy consumption, an important topic for an eco-friendly future [6] and a key component of the green computing philosophy [7].

The task assignment problem (TAP) can be formulated as a Generalized Assignment Problem (GAP) [8], which is an NP-hard [9,10]. Among the traditional methods for solving GAP, new methods based on dynamic programming and machine learning [11] techniques have emerged, such as reinforcement learning (RL) and neural network reinforcement learning (Deep RL). Our work proposes to improve the performance of current RL-based solutions through a multi-layer architecture that provides a passive knowledge transfer mechanism.

\* Corresponding author.

E-mail address: [alberto.robles@um.es](mailto:alberto.robles@um.es) (A. Robles-Enciso).

### 1.1. Motivation and contributions

Edge computing together with artificial intelligence is one of the most successful IoT technologies today. In fact, machine learning techniques can provide promising self-learning solutions, such as those based on reinforcement learning. Reinforcement learning is a technique that allows an agent to learn by itself in an interactive environment by trial and error using feedback from their actions and past experiences.

In our article we designed an edge computing architecture in which edge devices receive tasks and can process them locally or send them either to another edge device, to the fog server or the cloud. We present a first approach to solve the TAP by using a greedy solution, and then we propose the use of reinforcement learning techniques to increase effectiveness.

The initial RL proposal solution aims to improve performance and dynamically reduce energy consumption and latency in contrast to the fixed and non-dynamic performance achieved by the greedy solution. However, in real-world scenarios, edge devices have a reduced view of their environment and no real-time data, which limits the performance of RL algorithms due to biased actions. Therefore, it is necessary to modify the RL algorithms to improve their performance.

Our main proposal is to extend the RL agents' functionality in order to make them able to delegate the offloading decision in situations of uncertainty to agents in a higher layer to avoid making biased decisions when local knowledge is limited. For this purpose, we designed a multi-layer reinforcement learning system in which edge devices are independent RL agents modified to be able in specific situations to delegate the offloading process to a fog layer agent with a global view of the system. Our extended RL algorithm adds to the edge layer RL agents an extra action to perform the offloading query to the upper layer agent and learn from the result.

Having said that, we can summarize the main contributions of this work as the following:

- We propose a greedy approach to TAP as a baseline algorithm, edge devices will make use of heuristics to determine where to offload tasks.
- Based on the modelled task assignment problem, we propose a reinforcement learning system for the task offloading decision. Each edge device will be an RL agent that can decide to compute its tasks locally or send them to the edge, fog or cloud layer.
- We propose a novel RL approach based on a multi-layer system in which the RL agents of the devices can delegate the offloading decision to an agent of a higher layer.
- The performance of the proposed solution is compared with the Greedy and the single-layer RL algorithm, showing that the proposed solution is superior to the other algorithms.

### 1.2. Organization

This paper is organized as follows. In Section 2 we explore the state of the art of the task assignment problem. In Section 3 we formulated the assignment problem with its main components. In Section 4 we present a greedy approach as a baseline solution. In Section 5 we briefly introduce reinforcement learning theory and then propose our single-layer reinforcement learning approach. In 6 we present our novel multi-layer RL approach. In Section 7 we evaluate the proposed algorithms and present the results. Finally, the conclusion and future works are drawn in Section 8.

## 2. Related work

In this section, we will cover some of the most important methods obtained through literature research. Since it is difficult to find optimal solutions to the task assignment problem, especially given the prohibitive computational complexity in IoT devices, many approaches use

techniques or methods based on heuristics that search for sub-optimal solutions.

Greedy algorithms are common techniques that provide sub-optimal solutions but at a low computational cost and, in some cases, achieve solutions very close to the optimal. In [12] Rahabri et al. present a greedy approach to solve the task assignment problem using a knapsack-based scheduling algorithm (GKS). In their work they propose an algorithm that ranks tasks by their benefit and weight and assigns those with the highest value to fill the knapsack. They evaluate the solution using iFogSim, one of the most popular fog computing simulators.

Another way of solving optimization problems are algorithms based on metaheuristics, the most popular are Genetic Algorithms which are inspired by the process of natural selection. In [13] ZhenyueJia et al. formulate a cooperative multiple task assignment problem for UAVs and solve the problem using a modified genetic algorithm. They conclude, through numerical simulations, that their algorithm provides computationally feasible and efficient solutions.

Reinforcement learning (RL) is a novel technique based on machine learning that is not part of the well-known supervised and unsupervised learning paradigms. The purpose of reinforcement learning is to learn an optimal, or near-optimal, policy that maximizes the reward function and provides an optimal set of actions for different agent states and environmental conditions. Reinforcement learning algorithms learn iteratively through the immediate rewards they receive each time they perform an action based on their state.

In [14] Tanmoy Sen et al. propose to use RL to solve the TAP taking into account timeliness and energy consumption, and then propose a heuristic for solving the problem and design the reinforcement model based on the result of the proposed heuristic. They implement the proposed RL system (RILTA) as a model-free Q-learning algorithm and evaluate it in the iFogSim simulator. Their simulation results show that RILTA can reduce tasks processing time and energy consumption by about 10%–20% compared to other methods.

In some cases it is not possible to use RL algorithm directly, such as in scenarios where the agents' state is a large number of variables, making Q-Learning algorithms inefficient, or even when the state variables are not discrete. To address these limitations, researchers propose the use of neural networks to model the agent's learning process.

Jiada Wang et al. propose in [15] a Deep Reinforcement Learning based Resource Allocation (DRLRA) scheme, which can allocate computing and network resources adaptively, reduce the average service time and balance the use of resources under varying MEC environment. Their solution uses a neural network to calculate the Q-Value of each agent's state and select an action according to the  $\epsilon$ -greedy strategy. Since they use neural networks, it is necessary to perform a training state before using the algorithm. They evaluate the solution using a real-world network topology from Topology Zoo and develop a Python simulator for testing the DRLRA algorithm. They conclude, through various simulations, that compared to the classical OSPF algorithm under multiple conditions, their proposed DRLRA achieves much better performance.

Similarly, Xiaoyan Huang et al. propose in [16] a distributed multi-agent DRL scheme with the objective of minimizing the overall energy consumption while ensuring the latency requirements and Liang Xiao et al. in [17] present a RL and DRL offloading algorithm to address smart jamming and heavy interference in MEC considering transmission rate and power.

In general, RL algorithms have a common problem, the convergence time. This type of algorithm requires a series of iterations to reach the optimal solution, and in some cases using random factor policies may take even longer to reach the optimal solution. Mauricio Fadel Argerich et al. from NEC Laboratories propose in [18] to use external knowledge to guide the agents' decisions and experience. The method improves the performance during training by using expert or domain knowledge from a knowledge database in the form of programmable functions.

Their proposed system consists of an RL agent together with a “tutor” who accesses a knowledge database defined by the domain expert. The knowledge base is composed of several constraint-type functions, which limit the agent’s behaviour, and guide-type functions, which express heuristics of the domain that the agent will use to guide its decisions, especially in times of high uncertainty. During its first steps, the agent uses these knowledge functions to decide the best action, guiding its exploration and providing better performance from the start. In their tests, Tutor4RL achieves more than three times higher reward at the beginning of its training than an agent with no external knowledge. The work of Mauricio Fadel Argerich et al. has been one of the first inspirations for our work as it is one of the first proposals in which a system external to the RL agent helps it in decision making.

Finally, some approaches in the literature use other novel techniques to solve TAP. In [19] Dadmehar Rahbari et al. use classification and regression trees to solve the problem and use Cloudsim simulator to evaluate it. In [20] Mainak Adhikari et al. design a delay-dependent priority-aware offloading (DPTO) strategy by assigning a priority to each task according to its deadline and prioritizes the delay-sensitive tasks and minimizes the starvation problem of the low priority tasks. In [21] Lindong Liu et al. propose a supervised machine learning approach to solve the task scheduling problem based on classification data mining technique. They develop a novel classification mining algorithm (I-Apriori) based on the Apriori algorithm and implement a task scheduling model based on it. Haibin Zhang et al. propose in [22] the use of a blockchain together with DRL algorithm to improve security between edge servers in an ultra-dense network.

Regarding edge computing architecture, there are examples in the literature of different approaches to both the distribution of devices and the offloading process. Liang Tong et al. propose in [23] a typical hierarchical edge-cloud architecture with mobile devices at the bottom (edge) layer, cloudlet at the middle layer and data centre at the top (cloud) layer. They define and evaluate a workload placement algorithm to ensure efficient utilization of cloud resources. In [24] Wen Sun et al. design a Digital Twin edge network in 6G and propose a DRL-based mobile offloading scheme to reduce the latency in MEC applications.

As shown in this section, there are several architectures and techniques for solving an assignment problem, each with its advantages and disadvantages. Among these, artificial intelligence-based techniques are the most promising, and reinforcement learning stands out for its simplicity and ability to learn by itself through rewards. In this article we propose to use reinforcement learning to solve the task assignment problem in an edge-cloud architecture similar to those mentioned above to enhance the offloading process by deploying collaborative agents at each layer with different views of the environment. We also extend the algorithm to allow delegating decisions to other devices with more knowledge to improve performance in the initial stages of the learning process.

### 3. Task assignment problem

Edge computing systems are composed of a large number of heterogeneous devices with different characteristics and roles. Some devices have high computational power and serve as a host for processing tasks, while others with lower computational power constantly generate tasks for the applications they run. This forms a layered architecture where devices are separated into levels according to their role.

On top of this architecture appears a flow of offloaded tasks, as some lower-capacity devices decide to send tasks to more powerful devices for processing. We define a task as an indivisible piece of computation generated by a particular application, which has its own characteristics and constraints such as maximum latency, data size and computational resources required. One of the key components of these architectures is the Task Assignment Problem since it is necessary to determine the best possible distribution of tasks between devices at each layer.

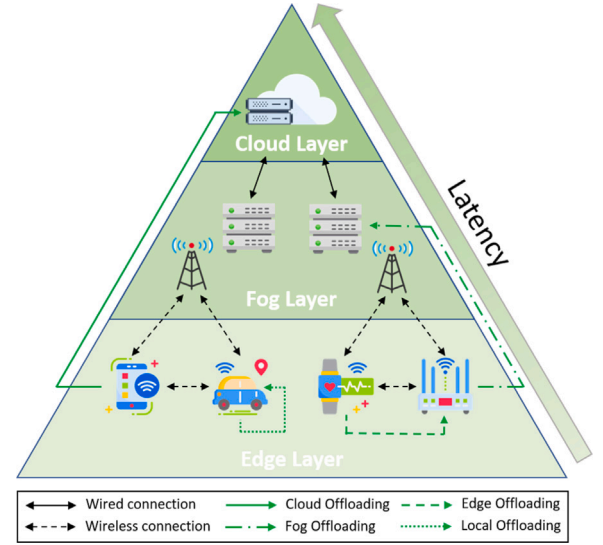


Fig. 1. Edge computing architecture.

An assignment problem is formulated as a combinatorial optimization problem where there is a set of agents and a set of tasks. Any tasks can be assigned to an agent to perform it, incurring some costs which may vary depending on the task-agent assignment. The assignment problem consists of finding the best task-agent assignment of all tasks and agents in order to meet specific constraints and minimize the cost function.

#### 3.1. System model

Our proposed system consists of devices that are separated into three layers, depending on their role, Fig. 1 shows the proposed offloading architecture. The layer closest to the users is the edge layer, which consists of heterogeneous edge devices that might have an intermittent connection and a dynamic position. This layer has the lowest latency and computational capacity, and is where tasks are generated from the edge devices that host IoT applications. Tasks can be processed locally or sent to devices in any of the three layers according to the decision of the offloading algorithm.

The fog layer is the middle layer where fog Servers are deployed. Fog Servers are small datacenters with intermediate computational capacity located between edge devices and the cloud, hence they have intermediate latency.

The upper layer is the cloud layer, which has very high computational capacity and medium-high latency. The cloud acts as a single device, but in real deployments it is typically a large number of high-performance computers in a datacenter, therefore its computational capacity can be very high. However, their latency is also high due to the distance to end-users.

The layers are connected to each other via wired and wireless connections. The cloud and fog layer are connected via cable, while the edge layer is connected to the fog layer and other edge devices via radio links. Each connection between levels has a particular maximum bandwidth.

In the edge layer each edge device executes an offloading algorithm that decides, using the local information of the device, where it offloads the tasks it generates. We call to the decision-making system that uses the perception of the environment an agent.

### 3.2. Task characteristics

In this subsection we define the task characteristics of our task assignment problem. In our three layers edge-fog-cloud architecture the tasks are generated from different applications running on edge devices at the edge layer. Each task has specific characteristics such as incoming data size and maximum latency, depending on the requirements of the application. In our model we have assumed that the size of the incoming data and the millions of instructions in the task are known in advance, but in some edge deployments this is not possible. Therefore, in case these parameters are not known exactly, they will be defined as the upper bound of their plausible value.

All the characteristics of a task are as follows:

1. Start time of the task. Timestamp when the task is executed.
2. End time of the task. Timestamp when the task is completed.
3. Deadline of the task. Maximum assumable task execution time.
4. Incoming data size. The data size refers to the amount of data in bits that the edge node must receive in order to process the task.
5. Millions of instructions (MI) needed to process the task.

We assume that the outgoing data, task output data to be sent by the edge node to the task generator node, are small enough to be omitted from our modelling. Consequently, the list of characteristics of a task  $k$  in device  $d$  can be formulated as the following vector.

$$T^{d,k} = \{start, end, dl, size, mi\}$$

### 3.3. Device characteristics

Our proposed architecture consists of a heterogeneous set of devices, each with specific characteristics, distributed in layers connected with different network technologies. The characteristics of the device will depend on its hardware specifications, its real-time status and the bandwidth of the connection with other devices and layers, so the characteristics of a device are as follows:

1. Current CPU usage (as a percentage).
2. The number of tasks currently in execution.
3. Maximum computational capacity of the processor in millions of instructions per second (MIPS).
4. Energy consumption per million instructions.
5. Energy consumption per transmitted bit.
6. Bandwidth with edge devices in bits per second (bps)
7. Bandwidth with fog servers in bits per second (bps)
8. Bandwidth with the cloud in bits per second (bps)

The list of characteristics of a device  $d$  can be formulated as the following vector.

$$D^d = \{cpu, tr, mips, ee, te, eb, fb, cb\}$$

### 3.4. Local processing

The first decision that can be taken in the offloading process is to process a task locally. When a device decides to process the task itself, it is not necessary to send it to other devices, so the local processing time  $L^l$ , also called latency, can be defined only as the processing time. Given a device  $d$  and a task  $k$  the local processing time is defined as:

$$L_{d,k}^l = \frac{T_{mi}^{d,k}}{D_{mips}^d}$$

Similarly, the power consumption corresponding to task  $k$  of device  $d$  only depends on the processing cost and is denoted by  $E^l$  and defined as:

$$E_{d,k}^l = T_{mi}^{d,k} \cdot D_{ee}^d$$

Therefore, the total cost of local processing is a weighted combination of the local processing time  $L^l$  and the local power consumption  $E^l$ :

$$C_{d,k}^l = L_{d,k}^l + \beta E_{d,k}^l \quad (1)$$

where  $\beta$  is a weighting parameter that regulates the trade-off between latency and consumption.

### 3.5. Edge processing

Alternatively, the processing of the task can be performed on another edge node. When an edge device decides to offload the task to the edge layer, it is necessary to send the task data to an adjacent edge device that will process it, thus both latency and power consumption will include the cost of sending data. Given a device  $d$ , a task  $k$ , and the adjacent edge device  $d'$  the edge processing time is defined as:

$$L_{d,k}^m = \frac{T_{size}^{d,k}}{R_{d,d'}} + \frac{T_{mi}^{d,k}}{D_{mips}^{d'}}$$

where  $R_{d,d'}$  is the maximum bit rate of the network between devices  $d$  and  $d'$  according to the Shannon–Hartley theorem. The channel noise is assumed to be white Gaussian noise with its variance as  $\sigma^2$  and the signal power  $S$  depends on the path loss propagation model [25].

$$R_{d,d'} = D_{eb}^d \log_2(1 + \frac{S}{\sigma^2})$$

Likewise, the energy consumption associate to task  $k$  of device  $d$  depends on the cost of processing at device  $d'$  and the cost of transmitting the data from device  $d$  to device  $d'$ .

$$E_{d,k}^m = T_{size}^{d,k} \cdot D_{te}^d + T_{mi}^{d,k} \cdot D_{ee}^{d'}$$

Accordingly, the total cost of edge processing is a weighted sum of the edge processing time  $L^m$  and the edge power consumption  $E^m$ :

$$C_{d,k}^m = L_{d,k}^m + \beta E_{d,k}^m \quad (2)$$

where  $\beta$  is the same constant as in the previous section.

### 3.6. Fog processing

In the same way as edge processing, the offloading process can decide to offload the task to the fog layer. As a result, the latency and energy consumption will include the cost of transmitting the data since it is necessary to send the task data to a fog node in the upper layer. Given a device  $d$ , a task  $k$ , and a fog device  $f$  the fog processing time is defined as:

$$L_{d,k}^f = \frac{T_{size}^{d,k}}{R_{d,f}} + \frac{T_{mi}^{d,k}}{D_{mips}^f}$$

where  $R_{d,f}$  is again the maximum bit rate of the connection between devices  $d$  and  $f$  according to the Shannon–Hartley theorem. The parameters  $S$  and  $\sigma^2$  have the same meaning as in the previous section.

$$R_{d,f} = D_{fb}^d \log_2(1 + \frac{S}{\sigma^2})$$

Equivalently, the energy consumption of task  $k$  of device  $d$  depends on the cost of processing at fog device  $f$  and the cost of transmitting the data from device  $d$  to device  $f$ .

$$E_{d,k}^f = T_{size}^{d,k} \cdot D_{te}^d + T_{mi}^{d,k} \cdot D_{ee}^f$$

Consequently, the total cost of fog processing is a weighted sum of the fog processing time  $L^f$  and the fog power consumption  $E^f$ :

$$C_{d,k}^f = L_{d,k}^f + \beta E_{d,k}^f \quad (3)$$

where  $\beta$  is again the same weighting constant as in the local processing section.



### 3.7. Cloud processing

Edge devices can choose to offload the task to the cloud layer, in which case it must send the task to a cloud server. This action is the one that generates the highest latency since the cloud layer is the most distant from the edge devices. As before, latency is defined as the time required to send and process task  $k$  from device  $d$  on a cloud server  $c$  and is described as follows:

$$L_{d,k}^c = \frac{T_{size}^{d,k}}{R_{d,c}} + \frac{T_{mi}^{d,k}}{D_{mips}^c}$$

$$R_{d,c} = D_{cb}^d \log_2(1 + \frac{S}{\sigma^2})$$

The bit rate  $R_{d,c}$  is determined by the Shannon–Hartley theorem since our model simplifies the connection of edge devices to the cloud by using a long-range radio link different from the one used in edge-fog communications. Nonetheless, in a scenario where the connection requires several hops to reach the cloud servers the bit rate would be defined according to the QoS of the multilayer backhaul network [26–28].

The energy consumption of task  $k$  of device  $d$  depends on the cost of processing at cloud server  $f$  and the cost of transmitting the data from device  $d$  to the cloud server  $c$ .

$$E_{d,k}^c = T_{size}^{d,k} \cdot D_{te}^d + T_{mi}^{d,k} \cdot D_{ee}^c$$

In consequence, the total cost of cloud processing is a weighted sum of the cloud processing time  $L^c$  and the cloud energy consumption  $E^c$ :

$$C_{d,k}^c = L_{d,k}^c + \beta E_{d,k}^c \quad (4)$$

### 3.8. Problem definition

In edge computing networks, the management of available resources is crucial. One of the key aspects of resource management is the allocation of user-generated tasks to nodes for processing. As shown above, a device can decide to execute a task locally ( $a = 0$ ) or send it to an adjacent node ( $a = 1$ ), a fog server ( $a = 2$ ) or the cloud ( $a = 3$ ), resulting in a specific cost as a weighted sum of execution time and energy consumption.

We define the cost function in the offloading problem as a trade-off, as proposed by several papers [29–31], since it is a common, simple and effective way to measure the performance of offloading actions. However, our main proposal of the multi-layer RL system could work with any other definition of the optimization problem.

Therefore, we formulate the cost of our optimization problem as a piecewise function that depends on the offloading action and its value are defined by the formulas (1), (2), (3) and (4). In a real environment, it may be possible that the offloading process fails ( $a = -1$ ), so it is necessary to define a penalty cost  $\delta$  in the following segmented function:

$$C_{d,k}(a) = \begin{cases} C_{d,k}^l & \text{if } a \text{ is equal to } 0 \\ C_{d,k}^m & \text{if } a \text{ is equal to } 1 \\ C_{d,k}^f & \text{if } a \text{ is equal to } 2 \\ C_{d,k}^c & \text{if } a \text{ is equal to } 3 \\ \delta_d & \text{if } a \text{ is equal to } -1 \end{cases} \quad (5)$$

We propose to design an optimization scheme in which the cost resulting from the allocation of tasks ( $K$ ) produced by a device  $d$  is minimized. Therefore, each device  $d$  aims to perform the best possible assignment of actions for each task  $k$  to minimize the resulting cost of all assignments. The optimization problem is formulated as following:

$$G_d = \min_{a_k} \sum_{k=1}^K C_{d,k}(a_k) \quad (6)$$

subject to:

$$\begin{aligned} \sum_{k=1}^{K'} T_{mips}^{d,k} &\leq D_{mips}^d \\ \forall k, L_{d,k} &\leq T_{dl}^{d,k} \\ a_k &\in \{-1, 0, 1, 2, 3\} \end{aligned}$$

The optimization problem is subject to the following constraints: The tasks assigned to a device ( $K'$ ) must not exceed the computing capacity of the device, no task must exceed its maximum latency (deadline) and the possible actions that can be taken are error, local processing, offload to an adjacent node, offload to the fog layer and offload to the cloud layer.

### 3.9. Proposed solutions

In the following sections we will propose three methods to solve the problem of task offloading. The first one is a greedy method that collects information from the devices in each layer and selects the best of them for offloading tasks according to a heuristic value calculated for each one. The drawback of this method is that it requires a large amount of real-time information that is difficult to achieve in real-world environments. Nevertheless, it provides sub-optimal solutions with good performance and will serve as a baseline for comparison with the other two methods. As an alternative to obtain near-optimal solutions we propose to use reinforcement learning algorithms, which allows the agent to learn and make decisions based on experience. In addition, we extended the functionality of the RL algorithms to a new multi-layer approach that allows the agent to delegate the offloading decision to another agent.

## 4. Greedy task offloading algorithm

As previously mentioned, the offloading problem is NP-hard and is computationally expensive, and in some cases even impossible, to find optimal solutions in a reasonable time. In this section we present an approach to solve the optimization problem using a greedy algorithm as a baseline algorithm to provide sub-optimal solutions at a low computational cost.

For each task produced by the device, the algorithm decides through some heuristics where the offload will take place. The complete process is detailed in Algorithm 1, which has a computational complexity  $O(|D_n| + |D_f| + |D_c|)$ .

At the beginning the algorithm needs to collect information about the current states of some sets of nodes, such as their computational capacity, average CPU usage and the number of currently running tasks. Based on the estimated information the algorithm greedily selects the device  $d$  with the least heuristic value for the task  $t$ .

The procedure followed by the algorithm starts with receiving as input the task to be offloaded, the device that generated it, the set of nearby nodes, the set of available fog nodes and the set of cloud nodes. Then for each set of devices the heuristic value of each device is calculated using the following formula:

$$\min = \omega_x \times \frac{t_{mips} * d_{lr}}{d_{mips}} \times \phi d_{cpu} \quad (7)$$

where  $\omega_x$  is a different weighting constant for each set ( $\omega_n$ ,  $\omega_f$  and  $\omega_c$ ) and  $\phi$  is the trade-off constant between actual CPU usage and running tasks.

**Algorithm 1:** Greedy Tasks Offloading Algorithm

---

**Input:** Task Source Device  $d_i$ , Nearest Edge Device Set  $D_n$ , Fog Server Device Set  $D_f$ , Cloud Device Set  $D_c$  and task  $t$  to offload. Nearest device weighing  $\omega_n$ , Edge server weighing  $\omega_f$ , Cloud weighing  $\omega_c$  and CPU trade-off  $\phi$ .

**Output:** Device  $d$

```

1 begin
  /* Local Offloading */
2   $d \leftarrow d_i$ 
3   $min \leftarrow \frac{t_{mips} * d_{tr}}{d_{mips}} \times \phi d_{cpu}$ 

  /* Nearest Device Offloading */
4  for  $d' \in D_n$  do
5     $min' \leftarrow \omega_n \times \frac{t_{mips} * d'_{tr}}{d'_{mips}} \times \phi d'_{cpu}$ 
6    if  $min' < min$  then
7       $min \leftarrow min'$ 
8       $d \leftarrow d'$ 
9    end
10 end

  /* Fog Server Offloading */
11 for  $d' \in D_f$  do
12    $min' \leftarrow \omega_f \times \frac{t_{mips} * d'_{tr}}{d'_{mips}} \times \phi d'_{cpu}$ 
13   if  $min' < min$  then
14      $min \leftarrow min'$ 
15      $d \leftarrow d'$ 
16   end
17 end

  /* Cloud Offloading */
18 for  $d' \in D_c$  do
19    $min' \leftarrow \omega_c \times \frac{t_{mips} * d'_{tr}}{d'_{mips}} \times \phi d'_{cpu}$ 
20   if  $min' < min$  then
21      $min \leftarrow min'$ 
22      $d \leftarrow d'$ 
23   end
24 end
25 return  $d$ 
26 end

```

---

**5. RL-based task offloading algorithm**

To better understand our proposal, in this section we briefly introduce the theory of reinforcement learning and then present our single-layer RL-based approach. In the following subsections we will explain the Markov decision process, then the concept of reinforcement learning and Q-Learning together with the mathematical definition of value, quality and reward functions. Finally, we will present an RL-based approach to solve the TAP, detailing how the agent's state is defined, how the reward function is calculated and the general operation of the implemented algorithm.

**5.1. Reinforcement learning**

One of the most widely used machine learning approaches is supervised learning, which uses input-output examples to predict the output of new inputs. However, in some scenarios it is only possible to obtain information as a reward for specific actions, the only thing that is known is the effect of the actions that are taken. The feedback obtained by performing an action guides the system towards the best actions according to its utility. Reinforcement learning aims to iteratively learn, by trial and error, the best action to take in each situation according to

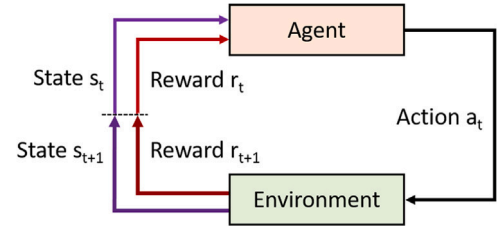


Fig. 2. Reinforcement learning process.

a given reward function, rather than using defined input-output pairs as in supervised learning.

The agent has a perception (state) of the environment to which it is connected and a set of possible actions to perform in order to change the environment. Thus, in each iteration the agent receives as input the current state of the environment and then select an action to perform from the action set. The action will change the state and produce a reward that the agent will use to learn if the action was the right one. Fig. 2 shows the typical behaviour of RL algorithms.

Therefore, an agent's behaviour can be summarized as a system that should choose actions that tend to increase the long-term total rewards. The agent's goal will be to find a policy ( $\pi$ ) that maps states to actions and maximizes a given reward function.

Reinforcement learning algorithms are modelled as a Markov Decision Process (MDP), a mathematical framework for describing stochastic control processes. It provides a theory for define sequential decision making in an evolving environment determined by a probability transition function that maps states and actions to other states. Markov decision processes are memoryless, the probability of transition from one state to another depends only on the current state and the action taken, it is independent of all previous states and actions. This behaviour of stochastic processes is called the Markov property and almost all reinforcement problems can be formalized by Markov decision processes that satisfy the Markov property. Mathematically, a Markov decision process is defined as a 4-tuple  $M = \langle S, A, P, R \rangle$ , where:

- $S$  is the state space, a finite set of all possible states of the system.
- $A$  is the action space, a finite set of actions that the agent can perform. Alternatively,  $A_s$  is the set of actions available from state  $s$ .
- $P$  is a set of transition probabilities from one state to another for a particular action.  $P_a(s, s')$  denotes the probability to go to state  $s'$  from state  $s$  by action  $a$ .
- $R$  is the reward function that determines the value of the immediate reward obtained after transitioning from state  $s$  to state  $s'$  by action  $a$ , denoted by  $R_a(s, s')$ .

Given a state  $s_t$  at a time slot  $t$ , the agent will select the action  $a_t$  to change to state  $s_{t+1}$  according to a  $\pi$  policy, and then receive a reward  $r_{t+1}$ . The policy  $\pi$  is a mapping from the state space to the probabilities of choosing a particular action. The goal of the MPD is to find an optimal policy  $\pi^*$  that maximizes the expected cumulative rewards  $R$ . The expected cumulative reward of a state  $s_t$  can be defined as the sum of the geometrically discounted future state rewards using the  $\gamma$  factor ( $0 \leq \gamma \leq 1$ ) as follows:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \quad (8)$$

Consequently, the value of a state can be defined in terms of future rewards  $r_t$  that can be expected. This value will give an estimate of "how good" a particular state is for the agent, and will depend on the actions taken when using the  $\pi$  policy. The value function of the state

$s$  under a policy  $\pi$  is defined as follows:

$$V^\pi(s) = \mathbb{E}_\pi [R_t | s_t = s] = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \mid s_t = s \right] \quad (9)$$

Similarly, the quality of an action  $a$  under  $\pi$  policy is expressed as the expected value of the cumulative reward starting from the state  $s$ , taking action  $a$ , and then following the policy  $\pi$ :

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[ R_t \mid \begin{matrix} s_t = s, \\ a_t = a \end{matrix} \right] = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \mid \begin{matrix} s_t = s, \\ a_t = a \end{matrix} \right] \quad (10)$$

As mentioned above, the MDP agent tries to find the best possible policy by comparing value functions. A policy  $\pi'$  is considered better than another policy  $\pi$  if for each state the expected value function has a higher value. In other words, a policy  $\pi'$  is better than or equal to  $\pi$  ( $\pi' \geq \pi$ ) if and only if the value of the state-value function of  $\pi'$  is greater than that of  $\pi$  ( $V^{\pi'}(s) \geq V^\pi(s)$ ) for all states ( $\forall s \in S$ ). In fact, there is always at least one policy that is better or equal to all others according to the Banach's fixed point theorem [32] and we can find it iteratively [33]. The best possible policy, or best policies set, is called the optimal policy ( $\pi^*$ ) and is the objective to be achieved by the MDP agent. Thus, the value and quality functions can be rewritten in terms of optimal policy and recursively according to the Bellman optimality equations [34] as follows:

$$V^*(s) = \max_\pi V^\pi(s) = \max_a \sum_{s'} P_a(s, s') [R_a(s, s') + \gamma V^*(s')] \quad (11)$$

$$Q^*(s, a) = \max_\pi Q^\pi(s, a) = \sum_{s'} P_a(s, s') [R_a(s, s') + \gamma \max_{a'} Q^*(s', a')] \quad (12)$$

In some cases, reinforcement learning algorithms need to ensure that the agent explores sufficiently. Off-policy methods separate the policy to be searched from the one used to make decisions, ensuring exploration and exploitation by balancing both types of action selection. One of the most important off-policy reinforcement learning algorithms based on temporal difference learning is Q-Learning. The learning process focuses on the optimization of the action-value function (Q) using an iterative update based on previous values and temporal difference. The general update process of the Q-function is defined by:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha [R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (13)$$

where  $s_t$  is the current state,  $a_t$  is the current action,  $\gamma$  is the discount factor and  $\alpha$  is the learning rate, between 0 and 1, which determines the proportion between the new learned value and the previous value. The Q-Values start with predefined or random values and are updated while the agent learns until it converges on the near-optimal policy.

## 5.2. Reinforcement learning-based task offloading solution

To solve the Task Assignment Problem defined in Eq. (6) we propose a reinforcement learning approach based on Tabular Q-Learning. Each edge device will run its own reinforcement learning algorithm to explore the optimal task offloading policy by minimizing the long-term cumulative discounted cost. Fig. 3 shows an overview of an agent of the proposed multi-agent RL system.

When a task is received, the agent will decide an offloading action ( $a_t \in A = \{0, 1, 2, 3\}$ ), whenever possible, whether to process the task locally (action 0), send it to a nearby node (action 1), send it to the fog layer (action 2) or send it to the cloud (action 3).

The decision will depend on the environment, which is based on the characteristics of the task, the state of the device and the last average state of fog and cloud. Classical Q-Learning algorithms require a finite state space, so it is necessary to discretize continuous values such as CPU usage and maximum latency. The state is composed of the following discrete parameters according to their maximum and minimum values:

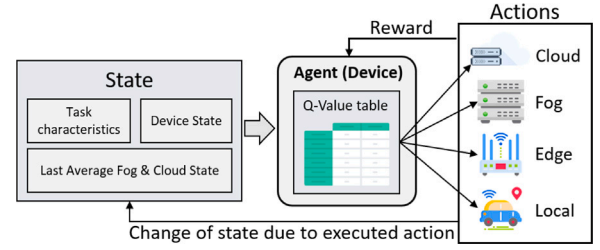


Fig. 3. Q-Learning agent.

- **MIPS required by the task:** Up to 20 000 MIPS the value is “Low”, from 20 000 to 100 000 is “Medium” and above 100 000 is “High”.
- **Maximum allowed latency:** Up to 6 s the value is “Low”, from 6 to 15 is “Medium” and above 15 is “High”.
- **Current use of device MIPS:** Up to 30 000 MIPS the value is “Low”, from 30 000 to 130 000 is “Medium” and above 130 000 is “High”.
- **Current use of device CPU:** From 0% to 25% of CPU use the value is “Low”, from 25% to 50% is “Medium”, from 50% to 75% is “Busy” and from 75% to 100% is “High”.
- **Last average CPU use of fog servers:** The percentage is divided into four equal parts in the same way that device CPU use.
- **Last average CPU use of cloud server:** The percentage is divided into four equal parts in the same way that device CPU use.

In a real world scenarios it is difficult to know the real-time status of the servers deployed in the fog and cloud layers as it would have a high impact on network usage and might even be impossible to implement without affecting system latency. Therefore, it is only possible for edge devices to obtain statistical information about the average CPU usage of the last minutes (configurable parameter, e.g. 5–10 m), which is calculated by fog and cloud servers and broadcasted to the network periodically (configurable parameter).

The learning process uses a table, the Q-Value table, to store and query the value of the Q-function for each state and action. When an action is performed, the new Q-Value in the table is updated according to the following one-step Q update formula based on (13):

$$Q(s_t, a_t) = (1 - \alpha) Q(s_t, a_t) + \alpha (C_t + \gamma \min_a Q(s_{t+1}, a)) \quad (14)$$

This formula updates the Q-Value using a percentage  $(1 - \alpha)$  of the old Q-Value and a percentage  $(\alpha)$  of the learned value, which is formed by the reward obtained  $C_t$  and the discounted Q-Value of the best action of the new state. The behaviour of the update function indicates that the algorithm is off-policy, since for the choice of action an  $\epsilon$ -greedy policy is followed while for updating the Q-Values a greedy approach is followed.

To determine the reward  $C_t$  obtained after the execution of an action at time  $t$ , we define a two-element piecewise function that depends on the task execution time to model the two possible completion states of a task, when it finishes within its deadline and when it does not (also includes failed tasks).

In case that the execution time of the task ( $T_{end}^t - T_{start}^t$ ) is less than its deadline ( $T_{dl}^t$ ) it is considered that its execution has been successful and its reward is calculated as a weighted sum between the execution time and the energy consumption of the task ( $T_{energy}^t$ ), as introduced in Section 3.

On the other hand, if the time required to complete the task exceeds its deadline, the task is considered not to have finished its execution properly despite having been executed successfully because it has not been completed within the maximum time allowed. Therefore, it is considered that the offloading choice was not appropriate and the

reward value should be penalized. Thus, the reward is calculated in the same way as the previous case, and then multiplied by a configurable penalty factor  $\delta$ . In addition, this case included the scenario where the task fails to execute for other reasons such as a transmission failure, a runtime error, a device out of battery, etc.

In summary, the piecewise function of the reward is defined as follows:

$$C_t = \begin{cases} (T_{end}^t - T_{start}^t) + \beta T_{energy}^t & T_{end}^t - T_{start}^t < T_{dl}^t \\ \delta \cdot ((T_{end}^t - T_{start}^t) + \beta T_{energy}^t) & \text{otherwise} \end{cases} \quad (15)$$

The reward function only considers energy and execution time, all other parameters are part of the state and do not need to be included as they will have an indirect impact on the latency.

The Q-Learning-based reinforcement learning solution proposed in this paper is detailed in the Algorithm 2, which has a computational complexity  $O(kSA)$  according to [35], where  $k$  is the number of learning steps,  $S$  is the number of states and  $A$  is the number of actions. In our proposal, the state space consists of a total of 1728 fixed discrete states, and the set of actions is 4, so a table of  $1728 \times 4$  elements is needed to store the Q-Values. Edge devices have enough memory space and computational power to store the whole Q-Table and run the algorithm.

Regarding the algorithm, the process starts with the definition of the algorithm's parameters, such as the discount factor  $\gamma$  of the rewards, the learning rate  $\alpha$  of the Q-function and the exploration rate  $\epsilon$  of the control policy. Then the algorithm goes into a loop that iterates over each step of the learning process. Each step  $t$  corresponds to the offloading decision process of a task  $T^t$ , and begins by obtaining the current state of the agent (*based on local status and latest average data from fog and cloud servers*) and determining the set of possible actions that the agent can perform (e.g., sensors cannot do local processing).

Once the state and set of actions are available, the Q-Value table is used to determine the best action. Since the algorithm is off-policy and follows an  $\epsilon$ -greedy control policy the best solution, the one with the lowest Q-Value, will only be selected if a random number  $e$  is greater than or equal to  $\epsilon$ , otherwise the action is chosen randomly from the set of feasible actions.

The action chosen will establish where the offloading of the task takes place, so it is necessary to send the task and its data to the appropriate node and wait for the result. Finally, the agent's new state  $s_{t+1}$  is determined, the reward  $C_t$  is calculated by piecewise function (15) and the Q-Value  $Q(s_t, a_t)$  is updated using the formula (14).

---

**Algorithm 2:**  $\epsilon$ -greedy Q-Learning Algorithm

---

**Parameters:** discount factor  $\gamma$ , learning rate  $\alpha$ , exploration rate  $\epsilon$  and penalty factor  $\delta$

```

1 begin
2   for each step  $t$  do
3     Observe actual state  $s_t$ 
4     Determine feasible action set  $A'$  from  $A$ 
5      $e \leftarrow$  random number from  $[0,1]$ 
6     if  $e < \epsilon$  then
7        $a_t \leftarrow$  randomly select an action from  $A'$ 
8     else
9        $a_t \leftarrow \arg \min_{a \in A'} Q(s_t, a)$ 
10    end
11    Execute offloading action  $a_t$ 
12    Wait for the task to be completed
13    Observe new state  $s_{t+1}$ 
14    Calculate reward  $C_t$  by (15)
15    Update  $Q(s_t, a_t)$  according to (14)
16  end
17 end

```

---

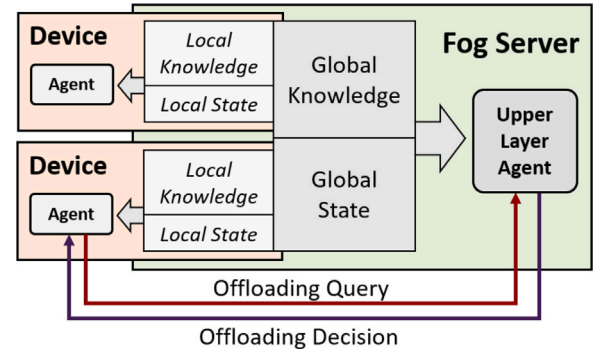


Fig. 4. Offloading query process.

## 6. Multi-layer RL-based task offloading algorithm

In this section we extend the previous solution to a multi-layer approach that improves the performance of reinforcement learning algorithms in the multi-agent task offloading problem.

In the algorithm explained in the previous section, each device works independently using its local information and aggregated global information. Thus, decisions are made according to the local state and knowledge of the agent. However, the biased view of the environment and the lack of knowledge in the early stages of the algorithm causes low performance in complex situations. To overcome this drawback, we propose to allow the RL agent of a device to delegate the offloading decision to a upper level agent in case it does not have enough information. The upper level agent, e.g. deployed in the fog layer, will decide according to its knowledge and the global state of the system. The offloading decision will be sent to the querying device and both the local and the upper level agent will learn from the reward obtained after executing the action. Fig. 4 shows the process of the offloading query.

In this enhanced architecture, both edge devices and fog servers run an independent RL algorithm that can collaborate between layers. The offloading query allows a passive knowledge transfer from fog agents to edge agents, especially useful when an edge agent starts with no knowledge and performs queries to learn from the decisions of the upper level agent.

We extend the functionality of the RL offloading algorithm from the previous section to include the new offloading query action in the set of feasible actions of edge devices and the query management in the fog server. Edge devices and fog servers execute the same algorithm but with different behaviour, as they have a different view of the environment.

The edge device agent works in a similar way to the previous one, with the difference that it now has the action ( $A \cup \{4\}$ ) of performing the offloading query. When a reward is received for an action queried to a fog server, it is considered to have a higher value since it is assumed that the upper agent will take better decisions. Also, the Q-Value of the query action is penalized as it is used to decrease the probability of being selected while increasing the agent's local knowledge. On the other hand, the upper layer agent waits to receive offloading queries from other agents and uses its own knowledge and view of the current state, including cloud and fog realtime CPU usages, to take an offloading decision and send it back. As a result, a reward will be obtained from the execution of the offloading action that will improve the knowledge of both the agent who made the request and the upper layer agent.

The proposed multilayer solution to the offloading problem based on  $\epsilon$ -greedy Q-Learning is shown in Algorithm 3. As mentioned before, all devices run the same algorithm but with their own view of the state



and their own knowledge. This means that each device will manage an independent Q-Table that will be trained locally. In addition, the upper layer agent will take advantage of all interactions with the devices to update their global status.

---

**Algorithm 3:  $\epsilon$ -greedy Multilayer Q-Learning Algorithm**


---

**Parameters:** discount factor  $\gamma$ , learning rate  $\alpha$ , exploration rate  $\epsilon$ , penalty factor  $\delta$ , query reward factor  $\rho$  and query use penalty  $\omega$

```

1 begin
2   for each step  $t$  do
3     Observe actual state  $s_t$ 
4     Determine feasible action set  $A'$  from  $A$ 
5      $isQuery \leftarrow false$ 
6      $e \leftarrow$  random number from  $[0,1]$ 
7     if  $e < \epsilon$  then
8        $a_t \leftarrow$  randomly select an action from  $A'$ 
9     else
10       $a_t \leftarrow \arg \min_{a \in A'} Q(s_t, a)$ 
11    end
12    if  $a_t$  is to ask a fog server then
13       $isQuery \leftarrow true$ 
14      Send the offloading request to a fog server
15       $a_t \leftarrow$  get the fog server decision
16    end
17    Execute or send the offloading action  $a_t$ 
18    Wait for the task to be completed
19    Observe new state  $s_{t+1}$ 
20    Calculate reward  $C_t$  by (15)
21    if  $isQuery$  then
22       $C_t \leftarrow \rho \cdot C_t$ 
23       $C_t^q \leftarrow \omega \cdot 1 \cdot C_t$ 
24      Update  $Q(s_t, 4)$  using (14) with  $C_t^q$ 
25    end
26    Update  $Q(s_t, a_t)$  according to (14) with  $C_t$ 
27  end
28 end

```

---

The algorithm begins by defining the parameters of the learning process, such as the discount factor  $\gamma$  of the rewards, the learning rate  $\alpha$  of the Q-function, the exploration rate  $\epsilon$  of the control policy, the latency-energy trade-off parameter  $\beta$ , the penalty  $\delta$  factor in case of offloading error, the  $\rho$  reward multiplier of queried actions and the penalty  $\omega$  factor of using the query action. In general, all devices will have the same parameter values to maintain consistency of results.

The algorithm then starts in a loop that iterates each time a task is received. In the case of an edge agent, tasks are received as they are generated, whereas a fog agent will only receive tasks when an offloading query is performed. After that, the current state and the set of feasible actions for the task  $T^i$  is determined and used to decide the offloading action to take following an  $\epsilon$ -greedy policy. If the action is to delegate the decision to an upper-level agent, then it is necessary to send the request and wait for the response. Once the offloading action is known, it can be executed or, in the case of a fog agent, sent to the agent that made the request and wait for the task to finish in order to calculate the reward obtained. If the offloading action is decided by the upper level agent, the value of the reward is increased by the  $\rho$  multiplier and the Q-Value of the query action is decreased by  $\omega t$  penalty factor. Finally, the Q-value of the action performed is updated according to the calculated reward using the formula (14).

## 7. Performance evaluation

In this section we will evaluate our proposed solution compared to the greedy and single layer reinforcement learning alternatives

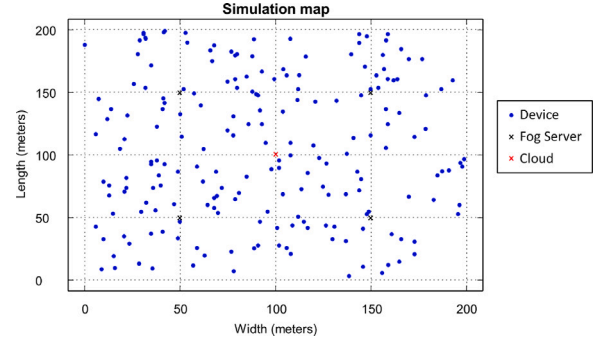


Fig. 5. Example of a simulation map of 200 devices.

**Table 1**  
PureEdgeSim simulation parameters.

Simulation parameter	Value
Simulation duration	10 min
Number of averaged simulations	10 per configuration
Min number of edge devices	10
Max number of edge devices	200
Simulation area	200 m $\times$ 200 m
Edge and fog bandwidth	100 Mbps
Cloud bandwidth	20 Mbps
Edge devices range	40 m

in a simulated edge computing environment. We will perform many simulations with each algorithm and measure their performance using a set of metrics to compare their effectiveness. The simulation results are available in our GitHub repository [36].

### 7.1. Methodology

The purpose of our evaluation is to obtain enough data to fairly compare the offloading algorithms. Therefore, we will make use of an edge computing simulator to test the behaviour of the algorithms from low-density to high-density device scenarios. We have considered an area of 200  $\times$  200 m is realistic for a Smart-Campus IoT scenario [37], and the minimum number of devices that makes sense would be 10. On the other hand, we consider high-density a scenario with more than 200 edge devices.

The output of each simulation will be a set of metrics used to determine the performance of the algorithm in the specific simulation scenario. To prevent inaccurate results caused by the random component of the simulator, the metrics will be calculated by averaging the result of several simulations on the same conditions.

### 7.2. Experiment setup

The evaluation has been performed on a modified version of PureEdgeSim [38,39] edge computing simulator, the source code of our extension is available on GitHub [40]. We have improved the functionality of the simulator to meet our requirements, including new performance metrics and reinforcement learning algorithms. The simulator runs in a Java 1.8 environment and on a computer with an AMD Ryzen 9 5900X and 32 GB of DDR4 RAM.

The simulated edge computing scenario consists of three layers of devices in an edge-cloud architecture, where the edge devices are randomly distributed in a 200  $\times$  200 m area (Fig. 5 shows an example). To verify the scalability of the proposed algorithms, the number of edge devices in each simulation is increased by 10 until 200. Each simulation lasts 10 min and is executed 10 times per configuration to calculate the average result. The bandwidth of the connection between devices is 100 megabits per second at the edge and fog layers, while

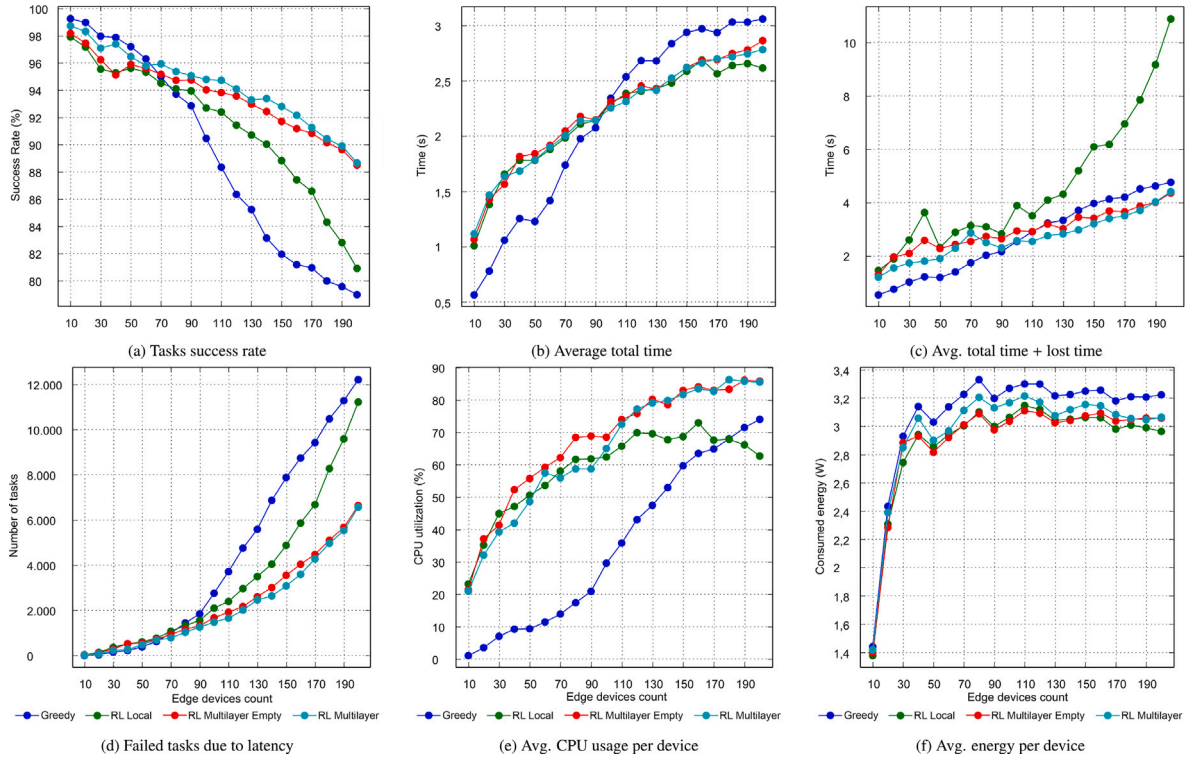


Fig. 6. Simulations results.

**Table 2**  
PureEdgeSim device characteristics.

Devices type	Cloud datacenter	Edge datacenter	Smartphone	Raspberry	Laptop	Sensor
Layer	Cloud	Fog	Edge	Edge	Edge	Edge
Number of devices	1	4	30%	10%	20%	40%
CPU cores	8	8	8	4	8	–
MIPS	2 000 000	1 000 000	25 000	16 000	110 000	–
Generate tasks	–	–	Yes	No	No	Yes
Idle consumption (Wh/s)	0.00015	0.00015	0.078	1.6	1.7	0.0011
Max consumption (Wh/s)	0.016	0.016	3.3	5.1	23.6	0.036

the connection to the cloud layer is 20 megabits per second. The maximum range of the wireless connection of the edge devices is 40 metres. The simulation parameters of the PureEdgeSim environment are summarized in Table 1.

The characteristics required by PureEdgeSim simulator to define the devices that compose the proposed three-layer architecture are detailed in Table 2. All devices with the exception of sensors have enough computing capacity to receive and process tasks, so sensors do not have the functionality to perform local offloading or to receive tasks from other devices. Among the edge devices only sensors and smartphones generate tasks in the system.

To test the behaviour of the offloading algorithms, three applications with specific latency and computational requirements are simulated. Table 3 shows the percentage of usage on each device and the specifications of each application according to PureEdgeSim specifications.

### 7.3. Metrics

To compare the performance of each algorithm we define the following benchmark metrics:

- **Task Success Rate:** The percentage of the tasks that finish their execution over the total. A task is not considered to complete its execution correctly if its execution time exceeds its deadline

**Table 3**  
PureEdgeSim applications parameters.

	Augmented reality	Health App	Data processing
Usage percentage	20	25	30
Generation rate (task/m)	45	35	45
Maximum delay (s)	6	30	10
Task length (MI)	120 000	600 000	18 000

or if the offloading process fails. This metric is one of the most important for the evaluation process.

- **Average Total Time:** The total time required to complete successfully a task, which includes the execution time and the time to send the task to the processing node. This metric is especially useful for comparing the latency incurred by each algorithm.
- **Complete Average Total Time:** Same as **Average Total Time** but also considering the time wasted on tasks that were not executed successfully.
- **Failed tasks due to latency:** The number of tasks that have failed because their execution time exceeds their maximum allowed latency.
- **Average CPU Usage per device:** Average CPU usage of a device. Useful to determine how much the computational resources of the devices are used.

**Table 4**

Greedy algorithm parameters.

Greedy algorithm parameter	Value
Nearest device weighing $\omega_n$	1.1
Fog server weighing $\omega_f$	1.5
Cloud weighing $\omega_c$	1.8
CPU trade-off $\phi$	20

**Table 5**

Reinforcement learning algorithm parameters.

Parameter/RL algorithm	Single	Multi
Learning rate $\alpha$	0.6	0.6
Latency-energy trade-off $\beta$	0.003	0.003
Discount factor $\gamma$	0.3	0.3
Failure penalty $\delta$	1000	1000
Average CPU refresh rate	60 s	60 s
Query reward factor $\rho$	–	0.2
Query use penalty $\omega$	–	10
Initial Q-value	200	200
Initial query Q-value	–	10

- **Average Energy Consumption per Device:** Average power consumption of one device.

#### 7.4. Compared methods

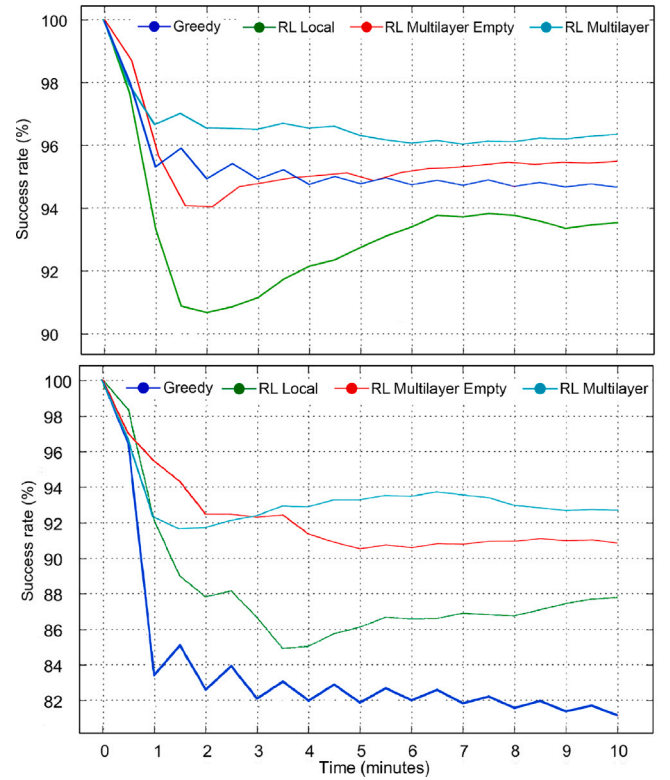
We have implemented in the simulator the three algorithms proposed in this article to evaluate their performance. For our experiments we assume that the CPU and energy cost of the methods is negligible because of their low computational cost and compared to the high cost of processing the offloaded tasks. The greedy solution will serve as a reference for comparison with the single-layer reinforcement learning algorithm and our proposed multi-layer guided RL. The parameters used by the heuristic of the greedy algorithm are shown in Table 4.

On the other hand, we designed three methods based on the implementations of the reinforcement learning solutions explained in previous sections. The first one is the basic implementation of a reinforcement learning algorithm that runs locally on each device without external knowledge, in the tests we will denote it as “*Local RL*”. The second and third methods are the same implementation of the multi-layer RL algorithm but with different initial conditions. The “*RL Multilayer Empty*” version starts each simulation with all Q-Tables (knowledge) of the devices completely empty, while “*RL Multilayer*” version uses on the fog servers the Q-Table resulting from the previous simulations, with the same configuration, to simulate the behaviour of a system that starts with knowledge to improve initial performance. The parameters used by both methods are summarized in Table 5.

#### 7.5. Experimental results and analysis

In this section we will show the most important results of the simulations performed for each algorithm and configuration. Each of the subfigures of Fig. 6 represents the metrics that were defined to make the comparison between algorithms from a scenario with 10 devices up to 200 and the value will be the average obtained by repeating the simulation 10 times.

One of the most critical results is the success rate in task execution, since in practice this has the most negative impact on the end-user. Fig. 6(a) shows the success rate resulting from each algorithm and configuration when performing the simulation. As we can see, in low device density scenarios, the greedy method outperforms the others until it reaches a medium density, 70 devices, where its success rate starts to drop. In the high device density scenarios the performance of the greedy method and the RL single-layer are very low while both multi-layer methods are able to keep an acceptable performance.

**Fig. 7.** Evolution of the success rate of tasks (70 and 170 devices).

This behaviour is due to the fact that in low device density scenarios there are not a large number of tasks and most of them can be executed by the fog and cloud servers without saturation, so the heuristics of the greedy method gives a better result than the reinforcement learning algorithms. When a medium density of devices is reached, the appropriate use of resources becomes more relevant and algorithms using reinforcement learning techniques are able to adapt dynamically to keep the success rate as high as possible. In high density scenarios with a large number of tasks the optimal use of processing nodes is critical, therefore the greedy method cannot achieve good results and even the single-layer RL method cannot improve the result. In contrast, multi-layer RL methods achieve a high success rate due to the possibility of delegating offloading decisions to higher level agents. In fact, the best success rate is achieved with multi-layer RL method that start with the Q-Table of the fog servers filled with the values learned from previous simulations since it allows to provide useful knowledge to the devices in the early stages of the learning process.

Furthermore, the multi-layer method offers superior real-time performance and rate of convergence to single-layer even in the version that starts without initial knowledge as shown in the simulation examples in Fig. 7. The single-layer method provides initially low performance that slowly converges to the best possible at the end of the simulation. However, the multi-layer method quickly achieves the best success rate of the single-layer method, due to the offloading query, to slowly improve the success rate by itself. In addition, the multi-layer method that starts from the Q-tables of the fog agents learned in previous simulations directly achieves the best result and maintains it throughout the simulation. In contrast, the greedy method has a fixed and stable behaviour as its heuristics do not have any dynamic component.

The average time required to complete a task for each algorithm and number of devices is shown in Fig. 6(b). Similar to the success rate, the average total time of the greedy algorithm drastically changes its performance based on the number of devices, while reinforcement

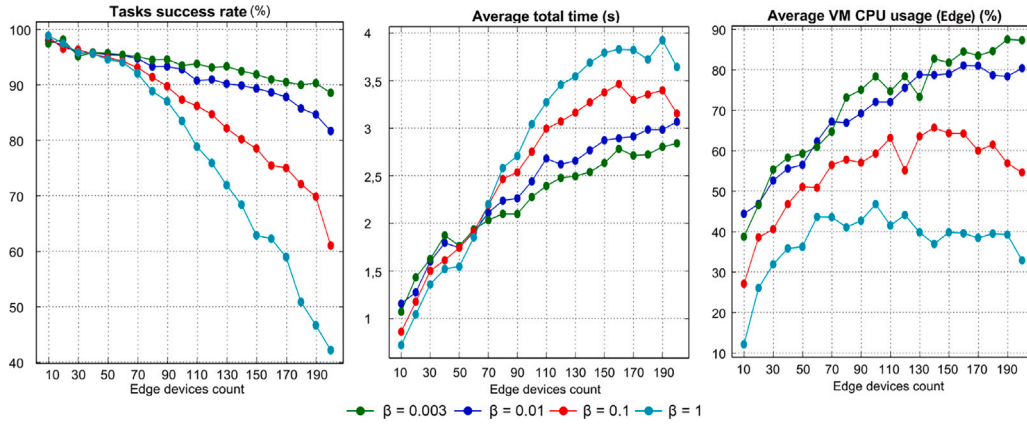


Fig. 8. Comparison of different trade-off  $\beta$  parameters.

learning algorithms slowly change the average total time. The three RL methods provide a similar average total time as this metric only considers tasks that have successfully completed their execution, which can lead to confusion when comparing them.

In order to clarify the performance of the methods in terms of total execution time, another metric is defined that also considers the time lost due to tasks that do not execute correctly because of latency. Fig. 6(c) to show the real impact of the algorithm's actions when deciding to do an unsuitable offloading. The behaviour of the greedy algorithm is similar to Fig. 6(b), but the single-layer RL method substantially increases the time in high device density scenarios as the impact of bad decisions in complex situations is very high (e.g. *deciding to send tasks to the cloud server saturating it instead of distributing them among the edge devices*). In contrast, multi-layer RL methods avoid the initial uncertainty by delegating the decision, thereby making better offloading decisions that reduce latency failures as can be seen in Fig. 6(d).

One more relevant result that can be analysed is the average CPU usage per device, which indicates the degree of utilization of the system's computational resources. A proper distribution of tasks among the devices results in a high average CPU usage per device as resource utilization is maximized. In contrast, low CPU usage indicates that the algorithm is saturating a few devices while many others are idle. As shown in Fig. 6(e), which represents the simulator output for this metric, the two multi-layer RL methods stand out from others, and the greedy method shows a low use of computational resources.

Similarly, the performance of algorithms can be measured in terms of their energy consumption as this is one of the components of the optimization problem, Fig. 6(f) shows the average energy consumption per device obtained from the simulations. The greedy algorithm presents the highest energy consumption while the reinforcement learning algorithms show the lowest energy consumption. The difference between the methods is small as in general all devices in the scenario have low energy consumption. Even so, the remarkable point of this result is how the RL methods manage to reduce consumption even when there is limited potential for improvement, while also improving the result significantly in the other metrics.

All simulations have been performed with a trade-off factor  $\beta$  of 0.003 since in our test scenario application latency is a critical factor that directly affects the rate of successful execution, while energy usage is relevant but does not directly affect the tasks success rate. However, there may be other scenarios where we may be interested in increasing the beta parameter in case energy usage is crucial. For example, in a system where the task maximum allowed waiting times are high and the edge devices are battery powered, it would be more important to properly manage the energy usage to avoid draining the batteries even if this increases the waiting time for the execution of tasks.

Fig. 8 provides a comparison of various values of the beta parameter of our multi-layer RL proposal, where it can be clearly seen that as the value of the beta parameter increases, the tasks success rate decreases rapidly. In our scenario this behaviour is because tasks have a short maximum waiting time, therefore any saturation on a server that causes the task to wait to start its execution will cause the task deadline to be exceeded and then it will be considered failed. As can be seen in the second graph of the figure, the higher the beta, the longer the waiting time of the tasks that are successfully executed as they are sent to more efficient servers but more saturated. This is demonstrated in the third graph where we can see the trend to use less CPU of the edge layer devices as beta increases, therefore the execution of tasks is performed in the fog and cloud layers, which are more energy efficient but unable to handle a high volume of tasks, causing a high impact on task execution time.

After having seen the performance of the algorithms in different simulator scenarios, we can conclude that the greedy algorithm offers acceptable performance in low and medium device density scenarios. However, as device density increases, more complex methods must be applied to maintain system performance. Reinforcement learning algorithms are able to adapt to complex scenarios at a low computational cost, thus providing the best results in simulations. Furthermore, our multi-layer approach stands out from other methods because in complex high-density scenarios it shows high performance in the most important metrics. This improvement is due to enhanced offloading decision system by using external knowledge and serves as evidence of the good performance of our multi-layer RL proposal. Therefore, reinforcement learning algorithms are good methods for solving the task assignment problem and our proposal is a useful and easily applicable extension to any RL algorithm to improve its performance.

## 8. Conclusion and future work

In this paper, we have presented the task assignment problem as a key component of collaborative edge computing architectures. As shown in the first section, there are several methods for solving TAP, but those based on artificial intelligence are the most promising. Reinforcement learning, a machine learning technique, is presented as a solution for the task offloading process in our proposed three-layer edge-fog-cloud architecture.

In this work we have studied different configurations to understand the impact of task distribution and limited vision of RL agents and how this impacts the performance behaviour of the algorithm in complex situations. To overcome these drawbacks, we propose a novel extension of reinforcement learning techniques that allows agents to query an upper-level agent with more knowledge and a broader view of the environment.



We have implemented our proposals together with a greedy alternative in a modified version of PureEdgeSim simulator and performed several tests to compare the performance of each algorithm in different situations following a set of metrics, providing access to the results and simulations for reproducibility. The experimental results showed that, compared with the greedy and classical RL algorithms, under multiple conditions, our proposed multilayer RL algorithm achieved much better performance in scenarios with a high number of devices and tasks.

Our proposal follows a passive approach of knowledge transfer, as future work we want to explore alternatives of knowledge transfer and federated learning to develop distributed decision systems in 5G networks and Computing Continuum paradigm [41,42]. Another promising future research direction is to apply neural networks to improve the performance of Q-Learning, thus overcoming some limitations such as a low number of states and discrete variables.

### CRedit authorship contribution statement

**Alberto Robles-Enciso:** Software, Investigation, Resources, Writing – original draft, Writing – review & editing, Validation, Formal analysis. **Antonio F. Skarmeta:** Validation, Methodology, Conceptualization, Supervision.

### Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

### Data availability

Data and code freely available in our GitHub.

### Acknowledgement

This work was supported by the FPI Grant 21463/FPI/20 of the Seneca Foundation in Region of Murcia (Spain) and partially funded by FLUIDOS project of the European Union's Horizon Europe Research and Innovation Programme under Grant Agreement No 101070473 and the project PID2020-112675RB-C44 funded by MCIN/AEI/10.13039/501100011033.

### References

- [1] A. Čolaković, M. Hadžialić, Internet of Things (IoT): A review of enabling technologies, challenges, and open research issues, *Comput. Netw.* 144 (2018) 17–39, <http://dx.doi.org/10.1016/j.comnet.2018.07.017>.
- [2] T.H. Noor, S. Zeadally, A. Alfazi, Q.Z. Sheng, Mobile cloud computing: Challenges and future research directions, *J. Netw. Comput. Appl.* 115 (2018) 70–85, <http://dx.doi.org/10.1016/j.jnca.2018.04.018>.
- [3] S. Tanwar, S. Tyagi, I. Budhiraja, N. Kumar, Tactile internet for autonomous vehicles: Latency and reliability analysis, *IEEE Wirel. Commun.* 26 (2019) <http://dx.doi.org/10.1109/MWC.2019.1800553>.
- [4] A. Mehrabi, M. Siekkinen, T. Kämäräinen, A. Yla-Jaaski, Multi-tier CloudVR: Leveraging edge computing in remote rendered virtual reality, *ACM Trans. Multimedia Comput. Commun. Appl.* 17 (2) (2021) <http://dx.doi.org/10.1145/3429441>.
- [5] R.K. Naha, S. Garg, D. Georgakopoulos, P.P. Jayaraman, L. Gao, Y. Xiang, R. Ranjan, Fog computing: Survey of trends, architectures, requirements, and research directions, *IEEE Access* 6 (2018) 47980–48009, <http://dx.doi.org/10.1109/ACCESS.2018.2866491>.
- [6] P. Fraga-Lamas, S.I. Lopes, T.M. Fernández-Caramés, Green IoT and edge AI as key technological enablers for a sustainable digital transition towards a smart circular economy: An industry 5.0 use case, *Sensors* 21 (17) (2021) <http://dx.doi.org/10.3390/s21175745>.
- [7] K. Raza, V. Patle, S. Arya, A review on green computing for eco-friendly and sustainable IT, *J. Comput. Intell. Electron. Syst.* 1 (2012) 3–16, <http://dx.doi.org/10.1166/jcies.2012.1023>.
- [8] D. Romero Morales, H.E. Romeijn, The generalized assignment problem and extensions, in: D.-Z. Du, P. Pardalos (Eds.), *Handbook of Combinatorial Optimization*, Vol. B, Springer, Germany, 2005, pp. 259–311, <http://dx.doi.org/10.1007/b102533>.
- [9] M.W.P. Savelsbergh, A branch-and-price algorithm for the generalized assignment problem, *Oper. Res.* 45 (1997) 831–841.
- [10] M. Garey, D. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, 1978.
- [11] A. Shakarami, M. Ghobaei-Arani, A. Shahidinejad, A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective, *Comput. Netw.* 182 (2020) 107496, <http://dx.doi.org/10.1016/j.comnet.2020.107496>.
- [12] D. Rahbari, M. Nickray, Low-latency and energy-efficient scheduling in fog-based IoT applications, *Turk. J. Electr. Eng. Comput. Sci.* 27 (2019) 1406–1427.
- [13] Z. Jia, J. Yu, X. Ai, X. Xu, D. Yang, Cooperative multiple task assignment problem with stochastic velocities and time windows for heterogeneous unmanned aerial vehicles using a genetic algorithm, *Aerosp. Sci. Technol.* 76 (2018) 112–125, <http://dx.doi.org/10.1016/j.ast.2018.01.025>.
- [14] T. Sen, H. Shen, Machine learning based timeliness-guaranteed and energy-efficient task assignment in edge computing systems, in: 2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC), 2019, pp. 1–10.
- [15] J. Wang, L. Zhao, J. Liu, N. Kato, Smart resource allocation for mobile edge computing: A deep reinforcement learning approach, *IEEE Trans. Emerg. Top. Comput.* (2019) 1.
- [16] X. Huang, S. Leng, S. Maharjan, Y. Zhang, Multi-agent deep reinforcement learning for computation offloading and interference coordination in small cell networks, *IEEE TVT* 70 (9) (2021) 9282–9293, <http://dx.doi.org/10.1109/TVT.2021.3096928>.
- [17] L. Xiao, X. Lu, T. Xu, X. Wan, W. Ji, Y. Zhang, Reinforcement learning-based mobile offloading for edge computing against jamming and interference, *IEEE Trans. Commun.* 68 (10) (2020) 6114–6126, <http://dx.doi.org/10.1109/TCOMM.2020.3007742>.
- [18] M.F. Argerich, J. Fürst, B. Cheng, Tutor4RL: Guiding reinforcement learning with external knowledge, in: AAAI Spring Symposium: Combining Machine Learning with Knowledge Engineering, 2020.
- [19] D. Rahbari, M. Nickray, Task offloading in mobile fog computing by classification and regression tree, *Peer-to-Peer Netw. Appl.* 13 (2020) 104–122.
- [20] M. Adhikari, M. Mukherjee, S. Srirama, DPTO: A deadline and priority-aware task offloading in fog computing framework leveraging multilevel feedback queueing, *IEEE Internet Things J.* 7 (2020) 5773–5782.
- [21] L. Liu, D. Qi, N. Zhou, Y. Wu, A task scheduling algorithm based on classification mining in fog computing environment, *Wirel. Commun. Mob. Comput.* 2018 (2018) 1–11, <http://dx.doi.org/10.1155/2018/2102348>.
- [22] H. Zhang, R. Wang, W. Sun, H. Zhao, Mobility management for blockchain-based ultra-dense edge computing: A deep reinforcement learning approach, *IEEE TWC* 20 (11) (2021) 7346–7359, <http://dx.doi.org/10.1109/TWC.2021.3082986>.
- [23] L. Tong, Y. Li, W. Gao, A hierarchical edge cloud architecture for mobile computing, in: IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, 2016, pp. 1–9, <http://dx.doi.org/10.1109/INFOCOM.2016.7524340>.
- [24] W. Sun, H. Zhang, R. Wang, Y. Zhang, Reducing offloading latency for digital twin edge networks in 6G, *IEEE Trans. Veh. Technol.* 69 (10) (2020) 12240–12251, <http://dx.doi.org/10.1109/TVT.2020.3018817>.
- [25] M. Samimi, T. Rappaport, G. Maccartney, Probabilistic omnidirectional path loss models for millimeter-wave outdoor communications, *IEEE Wirel. Commun. Lett.* 4 (2015) <http://dx.doi.org/10.1109/LWC.2015.2417559>.
- [26] K. Mebarkia, Z. Zsoka, Qos modeling and analysis in 5G backhaul networks, in: 2018 IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC, 2018, pp. 1–6, <http://dx.doi.org/10.1109/PIMRC.2018.8580739>.
- [27] X. Ge, H. Cheng, M. Guizani, T. Han, 5G wireless backhaul networks: challenges and research advances, *IEEE Netw.* 28 (6) (2014) 6–11, <http://dx.doi.org/10.1109/MNET.2014.6963798>.
- [28] C. Quadri, M. Premoli, A. Ceselli, S. Gaito, G.P. Rossi, Optimal assignment plan in sliced backhaul networks, *IEEE Access* 8 (2020) 68983–69002, <http://dx.doi.org/10.1109/ACCESS.2020.2986535>.
- [29] J. Zhang, X. Hu, Z. Ning, E.C.-H. Ngai, L. Zhou, J. Wei, J. Cheng, B. Hu, Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks, *IEEE Internet Things J.* 5 (4) (2018) 2633–2645, <http://dx.doi.org/10.1109/JIOT.2017.2786343>.
- [30] K. Zhang, X. Gui, D. Ren, D. Li, Energy-latency tradeoff for computation offloading in UAV-assisted multiaccess edge computing system, *IEEE Internet Things J.* 8 (8) (2021) 6709–6719, <http://dx.doi.org/10.1109/JIOT.2020.2999063>.
- [31] O.-K. Shahryari, H. Pedram, V. Khajehvand, M.D. Takht Fooladi, Energy and task completion time trade-off for task offloading in fog-enabled IoT networks, *Pervasive Mob. Comput.* 74 (2021) 101395, <http://dx.doi.org/10.1016/j.pmcj.2021.101395>.
- [32] A. Turab, W. Sintunavarat, On the existence and uniqueness of the solution of a probabilistic functional equation approached by the Banach fixed point theorem, 2020.
- [33] V. Heidrich-Meisner, M. Lauer, C. Igel, M.A. Riedmiller, Reinforcement learning in a nutshell, in: ESANN, 2007.

- [34] R.S. Sutton, A.G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 2018.
- [35] C. Jin, Z. Allen-Zhu, S. Bubeck, M.I. Jordan, Is Q-learning provably efficient? in: *NeurIPS*, 2018.
- [36] A. Robles-Enciso, ML-RL Simulations results, 2022, URL <https://github.com/alb1183/ML-RL-simulations>.
- [37] Universidad de Murcia - Gaia 5G. URL <https://ants.inf.um.es/en/gaialab>.
- [38] C. Mechalikh, H. Taktak, F. Moussa, PureEdgeSim: A simulation toolkit for performance evaluation of cloud, fog, and pure edge computing environments, in: *2019 International Conference on HPCS*, 2019, pp. 700–707, <http://dx.doi.org/10.1109/HPCS48598.2019.9188059>.
- [39] C. Mechalikh, H. Taktak, Moussa, PureEdgeSim: A simulation framework for performance evaluation of cloud, edge and mist computing environments, *Comput. Sci. Inf. Syst.* 18 (2020) 42, <http://dx.doi.org/10.2298/CSIS200301042M>.
- [40] A. Robles-Enciso, PureEdgeSim RL extension, 2021, URL <https://github.com/alb1183/ML-RL-PureEdgeSim>.
- [41] L.F. Bittencourt, R. Immich, R. Sakellariou, N.L.S. da Fonseca, E.R.M. Madeira, M. Curado, L.A. Villas, L. da Silva, C.A. Lee, O.F. Rana, The internet of things, fog and cloud continuum: Integration and challenges, *Internet Things* 3–4 (2018) 134–155.
- [42] D. Balouek-Thomert, E.G. Renart, A.R. Zamani, A. Simonet, M. Parashar, Towards a computing continuum: Enabling edge-to-cloud integration for data-driven workflows, *IJHPCA* 33 (2019) 1159–1174.



**Alberto Robles-Enciso** received the B.S. and M.S. degrees in computer science from the University of Murcia, in 2019 and 2020, respectively, where he is currently pursuing the Ph.D. degree. Since 2021, he has been a Seneca Pre-Doctoral Researcher with the Department of Information and Communications Engineering, University of Murcia. His main research interests are related to orchestration, Internet of Things, Edge Computing and energy optimization.



**Antonio F. Skarmeta** is a full professor and has been the head of the Research Group ANTS at the University of Murcia, Murcia, 31100, Spain, since its creation in 1995. His research interests include the integration of security services, identity, the Internet of Things (IoT), and smart cities. Skarmeta received a Ph.D. in computer science from the University of Murcia, Spain. Since 2014, he has been the Spanish National Representative for the Marie Skłodowska-Curie Actions within H2020. He has worked on and coordinated different European Union research projects in the IoT area, such as SMARTIE, SOCIOTAL, IoT6, and IoTCrawler.