Internet of Things 31 (2025) 101534

Contents lists available at ScienceDirect

Internet of Things

journal homepage: www.elsevier.com/locate/iot

Research article

Extending battery lifespan in IoT extreme sensor networks through collaborative reinforcement learning-powered task offloading

Mateo Cumia, Gabriel Mujica *, Jorge Portilla

Centro de Electrónica Industrial, Universidad Politécnica de Madrid, 28006, Madrid, Spain

ARTICLE INFO

Keywords:
Internet of things
Wireless sensor network
Extreme edge
Computation offloading
Edge computing
Reinforcement learning
Deep reinforcement learning
Q-learning

ABSTRACT

The use of wireless sensor networks (WSN) is increasingly widespread in the Internet of Things domain. Additionally, since the onset of the edge computing paradigm that brings the cloud closer to devices, these networks have seen improvements in battery lifetime and processing time, particularly in extreme edge architectures where network resources are more limited. Meanwhile, AI and machine learning techniques have been expanding across various domains to optimize different decision-making processes, including the task assignment problem in computation offloading. This article employs reinforcement learning (RL) techniques to address the task offloading problem, aiming to extend the lifespan of a WSN. To achieve this, a distributed multi-agent Q-learning algorithm is proposed, where sensor nodes (SNs) make collaborative decisions towards a common goal, avoiding selfish decision-making. The proposed algorithm is compared with two other state-of-the-art solutions, that is, a well-known Q-learning algorithm that allows centralized estimation of the Q-table before distributing it to the network's sensor nodes (SNs), and a similar implementation of this algorithm but using Deep Q-learning, which theoretically should achieve the best results. The outcomes show that the multi-agent RL algorithm improves performance when it takes other nodes in the network into account in its decisions, being the most suitable solution to be embedded in resource-constrained devices. Although it still achieves worse results than the Deep Q-learning algorithm, the latter involves much greater difficulties for implementation in real devices.

1. Introduction

Internet of Things (IoT) is an emerging technological trend that looks forward to actively contributing to the digitalization of the society. It involves the efficient interconnection of various physical devices and their application across various domains including industry, business, agriculture, infrastructure, communication, household management, healthcare, and numerous other facets of human existence [1,2]. The IoT paradigm is usually organized into layers to facilitate hardware/system abstraction, network design, and implementation. Each layer plays a specific role in the operation and management of the network.

There are typically three different layers in IoT architectures: the device layer, where physical IoT devices equipped with sensors and actuators collect data from the environment; the edge layer, established between devices and the cloud, which processes data locally to reduce latency and optimize data transmission; and the cloud layer, where large-scale IoT data is stored, processed, and analyzed, providing computational resources and advanced analytics services for decision making based on insights derived from IoT data.

* Corresponding author.
E-mail addresses: mateo.cumiaa@alumnos.upm.es (M. Cumia), gabriel.mujica@upm.es (G. Mujica), jorge.portilla@upm.es (J. Portilla).

https://doi.org/10.1016/j.iot.2025.101534
Received 9 August 2024; Received in revised form 10 January 2025; Accepted 4 February 2025
Available online 11 February 2025

One of the main advances in IoT technology is what is known as computation offloading through edge computing [3]. Computation offloading allows devices with limited resources, such as energy storage and computation capacity, to compute their tasks in remote devices with higher computing abilities. Traditionally, when devices were overwhelmed by their computational requirements, they would shift to perform computation offloading to the cloud (Cloud Computing), but the cloud is usually far away from the offloading device, resulting in high latency and energy consumption. This is why edge computing technology, which consists of bringing the cloud closer to the edge of the network, is gaining popularity [4].

The edge computing paradigm brings the question of when or where it is necessary to offload, or when it is necessary to compute locally. Many different algorithms have been used to look for the optimal decision [5]. In recent years this question has come to the field of Artificial Intelligence (AI), where AI-based optimization algorithms such as Machine Learning (ML), and in particular Reinforcement Learning (RL), a branch of ML in which an agent learns how to make the decisions based on the experiences it obtains from the environment, have been extensively researched. While most of the works related to computation offloading in the literature focus on the Mobile Edge Computing paradigm (MEC) [6], this article focuses on computation offloading in the area of wireless sensor networks, specifically in extreme edge architectures, where task offloading schemes are unexplored. A Wireless Sensor Network (WSN) is a network of tiny devices called nodes, each equipped with sensors and low-power microcontrollers, that collaborate on one or more common tasks, with specific sensing and wireless communication capabilities. They are widely used for different applications, such as healthcare, security, environmental monitoring, and Industry 4.0, among others. Extreme edge refers to the lowest layers of the IoT, also known as Mist. It represents a WSN where the devices are extremely limited in computing and storage capabilities [7]. Despite their limited resources, these devices are now capable of processing small amounts of data [8], and it is necessary to implement computation offloading algorithms that allow balancing network energy consumption, achieving increased network lifespan.

In this article, the task assignment problem is viewed as a variant of the Generalized Assignment problem (GAP) [9], which is NP-hard [10,11], as the focus is not only on reducing the average battery consumption, but also on minimizing the difference in battery levels between devices. In this sense, the lifetime of a network can have different meanings depending on the network's needs and application requirements. Generally, lifetime is regarded as the duration until, due to the battery depletion of some nodes within the network, it can no longer adequately perform its functions [12]. Some authors have even defined lifetime as the period until a single node in the network runs out of battery [13]. Therefore, just attempting to minimize the average energy consumption might be less useful if some devices run out of battery much sooner than the rest.

This can happen in several ways. For example, in a typical network, where devices at the lower layer must share resources (such as bandwidth) with those at the upper layers, if devices make decisions independently, there will be competition for the edge node's resources, since the best individual strategy for each node would be to offload tasks whenever possible. The strategy of minimizing individual consumption might not be optimal for the overall network's lifespan. In this competitive environment, if some nodes have higher local power consumption or task loads [14], they will drain their batteries much faster. This could also occur with centralized decision making that aims to minimize the total network consumption without considering battery differences among devices. For instance, if some devices have significantly higher consumption, the best global strategy to minimize the entire network's energy consumption would be to allocate edge resources to these higher local consumption devices. However, if this strategy remains fixed, some devices would always need to process locally, leading to a much faster battery drain compared to those that always offload tasks to the edge node.

This article addresses this concern by proposing a collaborative decentralized scheme for computation offloading in resource-constrained wireless sensor networks and its effectiveness in extending the WSN lifetime. To evaluate this, the performance of three Q-learning-based algorithms in addressing this challenge has been compared. Reinforcement learning techniques, particularly Q-learning-based methods, are among the most widely used approaches in the analyzed studies for solving computation offloading problems in IoT systems [15]. The three algorithms under comparison are:

- A decentralized Q-learning optimization algorithm for WSN, called Multi-Agent Collaborative Q-learning, enables individual devices to autonomously learn their own decisions, but also receive certain information from the rest of the network, to make decisions that aim to increase the network's lifespan. This algorithm simplifies the implementation in real-world scenarios by its straightforward nature and the minimal network information that nodes need to function effectively.
- A centralized Q-learning, called Edge Q-learning, is used for comparing the results. This algorithm is very similar to the one proposed in [16] where the edge layer estimates the offloading cost for the WSN sensor nodes with its global information, enabling them to make independent decisions afterward.
- Deep Q-learning using artificial neural networks, in this algorithm the edge layer estimates the offloading cost for the WSN sensor nodes. It is based on the Edge Q-learning algorithm, but implemented with neural networks, and modified so that it also takes into account the battery difference between nodes. It will be used as an ideal example of optimizing the defined performance metrics, as its implementation on actual extreme edge nodes is somewhat more complicated. However, the superiority of this type of algorithm over more traditional Q-learning algorithms is well known.

The rest of the article is structured as follows. In Section 2 the related work in the context of computation offloading is explored. Sections 3 and 4 outlines the system model and problem formulation of task offloading in WSNs, whereas Section 5 describes the architecture of the edge-enabled sensor network environment and introduces the different proposed algorithms based on artificial intelligence. Section 6 examines the experimental setup and the simulation results. Finally, Section 7 provides the main conclusions and future lines of the work.

2. Related work

Research on computation offloading for IoT devices is a highly active area within the community, especially when combined with machine learning techniques. Some of the most researched and utilized solutions in recent years consist of heuristic algorithms [17–19], especially GA (Genetic Algorithms), PSO (Particle Swarm Optimization), and ACO (Ant Colony Optimization) are some of the most studied nowadays. However, heuristic algorithms may encounter challenges in adapting to dynamic environments that can be found in a real network [20]. To address some of the issues encountered by these algorithms, solutions involving Artificial Intelligence (AI), and especially Reinforcement Learning (RL), a branch of Machine Learning (ML), or Deep Reinforcement Learning (DRL), are increasingly being investigated. Other techniques have also been used to address the problem of computation offloading, albeit to a lesser extent. For example, Mainak Adhikari et al. [21] introduced a delay-dependent priority-aware offloading (DPTO) strategy. This method assigns priorities to tasks according to their deadlines, ensuring that tasks sensitive to delays are prioritized while also addressing the starvation problem for tasks with lower priority. The research in [22] introduced a ‘‘Stackelberg’’ game to represent the interaction between the edge cloud and its users. In this scenario, the edge cloud determines prices based on its limited computing resources to maximize its revenue. In response to these prices, users make local offloading decisions to minimize their own expenses. Furthermore, a method for dynamic computation offloading and resource allocation leveraging Lyapunov optimization was introduced in [23].

Reinforcement learning is the foundation for various techniques, including Q-learning-based algorithms, commonly used by researchers in different optimization areas for MEC and WSN architectures. For instance, Youn et al. [24] applied the Q-learning technique and function approximation with linear regression to adjust the duty cycle of a WSN for a low latency and energy-saving schedule. Q-learning techniques have also been successfully applied to find energy-saving routing protocols in a WSN [25–27]. The research on the application of these RL techniques in computation offloading is also on the rise, and more authors are testing these solutions. For instance, Pan et al. [28] delved into dependency-aware task offloading in MEC, employing Q-learning to minimize execution time for mobile services with constrained battery power. Another task-dependency-aware Q-learning solution was studied in [29], where a new Q-learning computation offloading approach in MEC devices with task dependency is proposed. In another study by Robles-Encisco [30], an extended framework based on multi-layer reinforcement learning for task offloading in edge computing is introduced. This framework carefully balances energy consumption against execution time to devise strategies that minimize both factors. Numerical assessments highlight the superiority of this approach over a greedy algorithm, especially in intricate and uncertain scenarios.

Authors have also explored the application of these algorithms in vehicular edge computing [31] and sensor networks for smart healthcare. For instance, Rahul Yadav et al. [16] proposed a computation offloading scheme for Wireless Sensor Networks using reinforcement learning, emphasizing cloud-based estimation of overall offloading cost to reduce latency and energy usage. Moreover, there are combined approaches involving artificial neural network-based algorithms, particularly Deep Q-learning, to try to solve the problem of the restricted state space that exists with other techniques such as Q-learning [32–35]. In [33] a deep reinforcement learning-based offloading policy is suggested for collaborative computing in one such instance. Additionally, the authors compare the effectiveness of central Q-learning and Deep Q-learning algorithms. This comparison was also made in [32], where Huan et al. investigated computation offloading and resource allocation in dynamic multi-user MEC systems, utilizing the Double Deep Q-learning (DDQN) algorithm to target delay constraints and minimize energy consumption. Since the agent makes one decision for all the nodes in each time slot, these solutions can effectively function in small MEC networks. However, the neural network’s action space expands rapidly with the increasing number of IoT devices, so they might lack effectiveness when searching for an energy-efficient solution while dealing with a high number of sensor devices, such as those found in WSNs. Also, these algorithms try to reduce network energy total consumption. That, as previously mentioned, might not always be optimal for lifetime optimization.

Most of the proposed algorithms are designed for MEC networks. These algorithms can usually be applied to WSNs when decisions are made in a centralized manner, through edge nodes or the cloud layer. However, implementing these algorithms in WSNs becomes challenging when using distributed approaches, where each SN makes its own decisions. This challenge arises despite the many advantages of distributed algorithms due to the significantly reduced storage and processing capabilities of WSN devices. Although most WSN devices are now capable of processing their own tasks to a certain extent, they cannot execute advanced algorithms, such as deep learning ones. Some studies propose adapted solutions that enable WSN devices to decide independently but fail to consider potential collaboration between network nodes to extend their lifespan. In [16], the authors propose a Q-learning-based solution for a WSN, but the study does not account for potential battery level disparities among devices in heterogeneous networks. Similarly, in [30], a multi-agent Q-learning approach is considered for both sensor and edge devices, but it creates a competitive environment as it does not take into account possible battery differences between nodes.

3. System model

3.1. Network architecture

In the proposed scenario, there are N sensor nodes. These nodes are arranged in a single-hop star network, but the explained algorithms apply to multi-hop networks. These nodes are quite limited in their processing capacity but are still able to handle their own tasks. However, this will result in significantly higher energy and time consumption compared to offloading the tasks to the edge server, which has higher computational capacity and low latency. The set of SNs is denoted as $S = \{1, 2, \ldots, N\}$. A graphical representation of the network architecture is shown in Fig. 1.

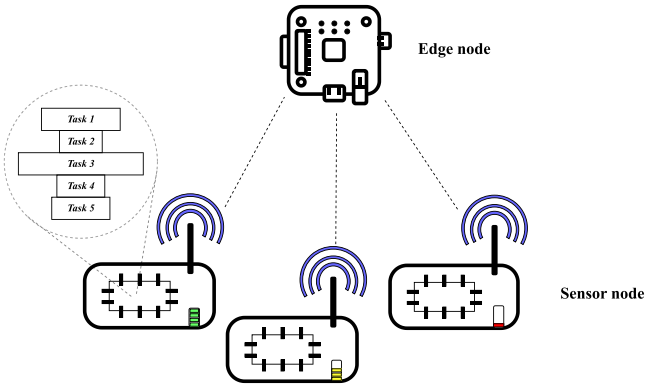


Fig. 1. Network model.

The sensors generate T tasks that need to be processed at regular intervals, referred to as the the duty cycle. Given that each SN handles more than one task within each duty cycle, it can be assumed that SNs generate these distinct tasks directly. Alternatively, this result can be viewed as the segmentation of a larger task generated by each SN. This segmentation results in smaller subtasks, managed independently, thereby aiming for a more efficient allocation of resources, reducing the fragmentation issue of leaving underutilized edge resources [36]. Such an approach maximizes the resource utilization at the edge layer (e.g., bandwidth). From now on, the first option explained, where independent tasks are generated, will be considered, and the term subtask will not be used to refer to them.

These tasks can be handled either through local computation or offloading to the edge layer. However, due to limitations in the edge capacity, not all SNs will be capable of offloading their tasks.

It is assumed that all sensors have the same initial battery, and power consumption during information transmission, denoted as P^t. However, they do not all have the same processing power consumption, denoted as P^p. Additionally, W is defined as the available edge layer resources for the entire SN network to offload tasks.

3.2. Task model

Each SN has a set of tasks, and each task has specific characteristics. The characteristics of task k on device n are denoted as:

$$T^{n,k} = (B, D, T_R).$$

Where B represents the size of incoming data in bits, and D denotes the total number of CPU cycles required to perform the computation task.

Computation complexity, represented by the variable C, is defined as the number of CPU cycles required to process a single bit of information. Thus D is directly proportional to B and C. It is assumed that regardless of whether the task is processed locally or offloaded, the value of C remains constant. This is because, in task processing, there is a processing chain in which several stages can be involved. This parameter captures an approximation that reflects this fact as an average value. Additionally, C is uniform across all tasks, while B fluctuates among different tasks. The transmission time is defined as T_R, which is required to send the task to the edge layer, and it will depend on the selected transmission rate T_t.

It is assumed that tasks generated by each node in a working cycle cannot be further divided or subdivided, so each task must be fully processed locally or offloaded. The offloading decision for task k from device n is defined as $a_{n,k} = 0, 1$. A binary variable $K_{n,k}$ is also defined to indicate whether a task k of SN n is currently running.

$$K_{n,k} = \begin{cases} 1, & \text{if task } k \text{ is running on node } n, n \in \mathbb{N}, k \in T \\ 0, & \text{otherwise.} \end{cases}$$

3.3. Computing model

(1) Local Execution Model: for $a = 0$, the task will be executed locally using the resources of the SN. The execution time of task k on device n is denoted as:

$$L^0_{n,k} = \frac{D_{n,k}}{f_n}. \tag{1}$$

Where f_n represents the device's CPU frequency. The energy consumption of the device n to execute the task k is:

$$E^0_{n,k} = L^0_{n,k} P^p_n. \tag{2}$$

Therefore, the total cost for local processing can be expressed as:

$$C^0_{n,k} = E^0_{n,k}. \tag{3}$$

Thus, the cost of locally processing a task is established as a function that depends both on the time taken or employed for processing, and the energy expended in this.

(2) Edge Execution Model: for $a = 1$, the task will be executed at the edge server. The time spent in executing the task will depend on both the processing time spent in the edge layer and the transmission time.

The execution time of task k on device n, when the decision is made to send the task to the Edge layer, is denoted as:

$$L^1_{n,k} = \frac{D_{n,k}}{f_e} + T_R. \tag{4}$$

Where T_R is defined as:

$$T_R = \frac{B_{n,k}}{T_t}. \tag{5}$$

and the energy consumption of device n to execute task k, which only includes the consumption of the SN, i.e., the energy consumed during transmission, is:

$$E^1_{n,k} = T_R P^t_n. \tag{6}$$

Therefore, the total cost for processing at the edge can be expressed as:

$$C^1_{n,k} = E^1_{n,k}. \tag{7}$$

The total network cost can also be defined as:

$$C_all = \sum_{n=1}^{N} \sum_{k=1}^{T_n} K_{n,k}((1 - a_n)C^1_{n,k} + a_n C^0_{n,k}). \tag{8}$$

To address variations in battery levels among SNs, a new parameter is introduced:

$$V_{n,t} = \bar{E_{N,t}} - E_{n,t} \tag{9}$$

This parameter represents the difference in battery level between the average of all SNs in the network and the battery level of node n at time t. It is used to provide information about the overall network to each SN, so they can make decisions that benefit the rest of the WSN, always aiming to increase its lifetime.

4. Problem formulation

In this article, the target is to increase the lifespan of the sensor network, defined as the time elapsed until the first node runs out of battery. Additionally, it aims to ensure their completion before the subsequent duty cycle.

This translates to aiming to minimize the total energy consumption, while ensuring that the batteries of the nodes maintain similar levels so that nodes with higher battery levels can free edge resources, allowing those with less battery to process at the edge layer. The problem is formulated as follows:

$$\begin{aligned}
&\min \sum_{n=1}^{N} \sum_{k=1}^{T_n} K_{n,k}((1 - a_n)C^1_{n,k} + a_n C^0_{n,k}). \\
&\min \sqrt{\frac{1}{N} \sum_{n=1}^{N} (B_i - \overline{B})^2} \\
&\text{s.t.} \\
&C1 : a_n \in \{0, 1\}, \quad \forall n \in \mathbb{N} \\
&C2 : \sum_{n=1}^{N} \sum_{k=1}^{T_n} K_{n,k} B_n a_n \leq W, \quad \forall n \in \mathbb{N}, \forall k \in \mathbb{T} \\
&C3 : \sum_{n=1}^{N} K_{n,k} = 1, \forall k \in \mathbb{T}.
\end{aligned} \tag{10}$$

Where C1 indicates that the available actions are limited to local or offload, C2 expresses that the resources of the edge layer should not be exceeded. C3 indicates that a task cannot commence processing until the previous one has concluded, allowing each SN to process only one task at a time.

5. Proposed architecture

As mentioned before, the objective of this work is to implement a new multi-agent Q-learning algorithm to address the problem of extending the network's lifespan through task offloading and compare it to an existing Q-learning algorithm, and to an ideal Deep Q-learning algorithm. Q-learning is a classic RL algorithm that records the Q value, a fundamental measure representing the benefit of an agent for taking an action in a specific state. This value is considered a long-term reward. Each state–action pair contains a value $Q(s,a)$ that is stored in a Q-table. The formula for estimating $Q(s,a)$ is as follows:

$$Q(s_t,a_t) = Q(s_t,a_t) + \alpha[R(s_t,a_t) + \gamma \max Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)]. \tag{11}$$

In this process, each state–action pair is computed as a balance between the immediate reward gained by taking action a_t in state s_t and the highest Q-value achievable in state s_{t+1}, resulting from the earlier decision made in state s_t. This balance is adjusted by the discount factor γ, a constant that satisfies $0 \leq \gamma \leq 1$. The other factor α is a constant that satisfies the same constraints but serves to determine the proportion between the new learned value and the previous value. It is important to note that, in all three algorithms to be presented, independent decisions will be made for each of the tasks that each SN possesses. This means that the decision-making process will occur each time a new task appears, typically when an SN completes its previous task, in contrast to alternative solutions often observed in MEC networks, where algorithms decide periodically, making a single decision that applies to all devices at once. This approach leads to a significant increase in the action space as the number of network devices grows, by defining the action space as a vector $A = [a_1, a_2 ... a_N]$. Increasing the action space significantly complicates the task for RL algorithms to learn optimal solutions. This increases the time required for learning, particularly in Q-learning algorithms, where there exists a table dependent on the number of possible actions. Therefore, these approaches do not seem optimal when the number of network devices is very large. Expanding the action space in this way can hinder the ability of RL algorithms to find optimal solutions efficiently.

The proposed solution adapts to dynamic changes in the battery levels of the network nodes while remaining robust and adaptable to other forms of dynamics introduced in the network, such as the addition or removal of nodes. A decentralized approach, which relies on independent decision making, offers inherent advantages in managing these changes, as the algorithm, unlike centralized solutions, does not depend on a fixed number of nodes in the network. Although independent multi-agent Q-learning can face challenges of non-stationarity in highly dynamic environments, in practice, Independent Q-Learning (IQL) has proven to perform effectively, even in mixed and competitive scenarios [37].

This work evaluates three different architectures, centralized Edge Q-learning and Deep-Q-learning where the edge node is responsible for making decisions for the sensor nodes, and decentralized Q-learning, where each device takes its own decisions. The information flows required for decision-making for an SN in each of the proposed schemes are depicted in Fig. 2.

5.1. CMA Q-learning

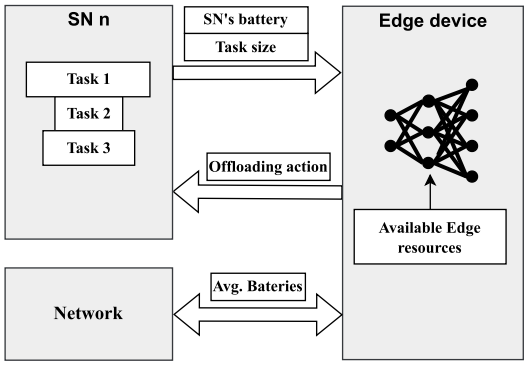
In this algorithm, each node represents an individual agent that independently decides which actions to execute, creating its own Q-table based on its specific state and characteristics. This design allows the algorithm to effectively account for heterogeneity among nodes, as each node makes decisions based on its own consumption, battery level, and computational capacity, learning the Q-values associated with its individual state transitions and estimating the associated costs in its particular situation. Additionally, nodes have access to information about the overall average battery levels in the network, fostering collaboration and enabling the creation of a Collaborative Multi-Agent (CMA) Q-learning approach. This model is simpler than the centralized one, as it does not require the edge to constantly collect information about the network state, consumption, and time, making it the easiest one to implement in constrained sensor devices, as well as the most suitable for real-life scenarios. This algorithm only requires nodes to periodically send data about their battery levels to the edge node. The edge node then periodically communicates the average battery level information to the entire network. Additionally, it is understood that the network consists of devices that have a minimal processing capability that allows them to execute the algorithm. The Q-learning used for the task offloading scheme is represented in Algorithm 1, where an epsilon greedy control policy is established to have a balance between exploration and exploitation. So when a random number e is greater than the exploration rate, the action for state S_t will be chosen from the Q-table, and when the opposite happens the action will be chosen randomly.

There are three key elements in the reinforcement learning method, namely state, action, and reward:

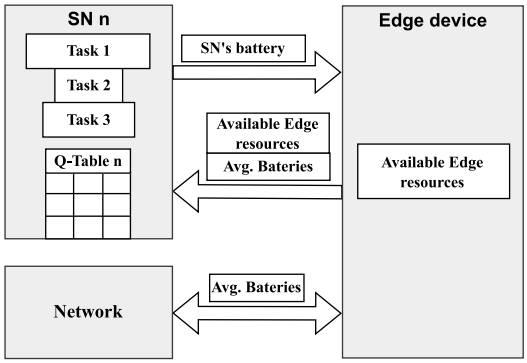
1. State: the state space S includes all resource characteristics (including battery difference and bandwidth) of the SN and the resource requirement of the task. The system state for nodes and tasks is defined as: $S_t = \left(V_{n,t}, W_t, B_{n,k}\right)$ Where $V_{n,t}$ indicates the battery difference of the SN with the network's average batteries, W_t indicates the available bandwidth, and $B_{n,k}$ indicates the incoming size or task t_k. For simplicity, it is assumed that the state space S consists of x levels of available bandwidth and y levels of task size, then the state space of an incoming SN task can be introduced as

$$S_t = \begin{bmatrix} s_{11}(\tau) & \cdots & s_{1x}(\tau) \\ \vdots & \ddots & \vdots \\ s_{y1}(\tau) & \cdots & s_{xy}(\tau) \end{bmatrix}$$

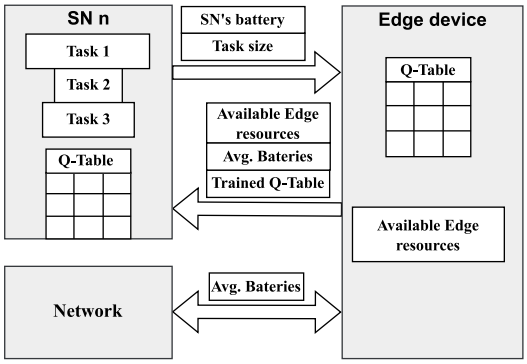
Considering $s_{x,y}(\tau) \in S$ where x and y signify the available bandwidth and task size levels, respectively.



(a) Deep-QL.



(b) CMA-QL.



(c) Edge-QL.

Fig. 2. Different network schemes for each algorithm.

2. Action: The sensor node will choose an action from the action space A, $a_{n,k}$, for dealing with task t_k depending on the state they are in, so that:

$$a_{n,k} = \begin{cases} 1, & \text{if task is executed locally.} \\ 0, & \text{if task is offloaded to the edge layer.} \end{cases}$$

3. Reward: An SN obtains a reward as a consequence of taking an action in a specific state. As each node decides independently in this scheme, it has to estimate the cost of actions based on its own characteristics, without considering the rest of the nodes. This leads to more selfish actions and results in poorer overall performance as they need to consider that if their battery level

is significantly higher, they should free up processing resources at the edge layer for nodes with lower battery levels. This issue is tackled by providing each node with an approximate average of the batteries in the rest of the network. In this way, they can make decisions considering whether covering more resources at each moment is beneficial for the other nodes (Eq. (12)).

$$R_t = \begin{cases} -C^0_{n,k} - \psi V_{n,t} & \text{if task is executed locally.} \\ -C^1_{n,k} + \psi V_{n,t} & \text{if task is offloaded.} \end{cases} \tag{12}$$

Here, a positive reward is obtained when choosing to process locally if its battery is higher than the average, and it becomes more positive the greater this difference is. Similarly, the reward is also positive when choosing to process in the edge layer if its battery is lower than the average, and it becomes more positive the greater this difference is. This effect weight is regulated by the parameter ψ. Also, in both cases, the reward is negatively related to the actual cost of the system.

5.2. Edge Q-learning

An adaptation of CORL from [16] named here as Edge Q-learning is used for comparison as it is one of the few studies focused on enabling networks of resource-constrained devices, such as WSNs, to perform task offloading independently, without relying on centralized decision-making. In this learning strategy, the agent, represented by the edge layer, is responsible for estimating the overall offloading cost from the possible state space and actions to create the Q-table using reinforcement learning. This table is then shared among the SNs to make the offloading decision of an incoming task, so that they can independently decide, reducing decision time. A more detailed description of the algorithm's functioning which is under comparison can be found in [38]. The state, action, and reward is as follows:

1. State: The system state for nodes and tasks is defined as: $S_t = \left(W_t, E_{n,k}, B_{n,k}\right)$ Where W_t indicates the available bandwidth, $E_{n,k}$ indicates the SN's battery level, and $B_{n,k}$ indicates the incoming size or task t_k.
2. Action: The action space is defined in the same way as in the CMA Q-learning:

$$a_{n,k} = \begin{cases} 1, & \text{if task is executed locally.} \\ 0, & \text{if task is offloaded to the edge layer.} \end{cases}$$

3. Reward: As estimating the cost of the actions for individual SNs in the whole network is targeted, the reward must be negatively related to the total cost Eqs. (8). (9) as follows:

$$R_t = -C_{\text{all}}. \tag{13}$$

Algorithm 1 Q-Learning Algorithm

Require: discount factor γ, learning rate α, exploration rate ϵ.

begin
Each device does:
for each step t **do**
  Observe actual State S_t
  $e \leftarrow$ random number from $[0, 1]$
  if $e < \varepsilon$ **then**
    $a_t \leftarrow$ randomly select an action from A
  else
    $a_t \leftarrow \arg\max_{a \in A} Q(S_t, a)$
  end if
  Execute offloading action a_t
  Calculate reward R_t by (12) or (13)
  Observe S_{t+1}
  Update $Q(S_t, a_t)$ according to (9)
  $S_t \leftarrow S_{t+1}$
end for
end

These two algorithms are now compared to a similar Deep Q-Learning solution that combines both approaches, estimating the offloading cost of individual SNs in the whole network, but also taking into account the possible problem of battery difference when just trying to optimize the network's energy consumption.

5.3. Deep Q-learning

By applying Q-learning, the corresponding Q value has to be calculated and stored in a table for each state–action pair. However, practical scenarios may involve an extensive number of possible states, exceeding tens of thousands. Storing all these Q values in the Q-table would result in an extremely large $Q(s, a)$ matrix. Consequently, obtaining enough samples to cover each state becomes challenging, which could lead to the algorithm's failure. Therefore, instead of calculating Q values for each state–action pair, a Deep Neural Network is used to estimate $Q(s, a)$ – in other words, a neural network is employed to approximate the previously presented equation used to obtain $Q(s, a)$. Thus, rather than looking up the Q value in a table, the current state is input into the neural network, which has as many outputs as there are possible actions, and it returns in each of these outputs the corresponding Q value for each action in that state. This constitutes the fundamental concept behind Deep Q-learning.

Algorithm 2 Deep Q-Learning Algorithm

Initialize replay memory G, network parameters θ, target network parameters θ'
begin
for each step t **do**
  A request from one of the SN n is chosen
  Observe actual State S_t^n
  $e \leftarrow$ random number from $[0, 1]$
  if $e < \varepsilon$ **then**
    $a_t \leftarrow$ randomly select an action from A
  else
    $a_t \leftarrow \max_a Q^*(s_t^n, a^n; \theta)$
  end if
  Execute offloading action a_t^n
  Calculate reward R_t^n by (12)
  If a task of SN m has been finished, continue, if not, take another request:
  Observe S_{t+1}^m
  Store transition $(S_t^m, a_t^m, R_t^m, S_{t+1}^m)$ in G
  Sample random mini-batch of N transitions (S_j, a_j, R_j, S_{j+1})
  Construct target values for N transitions
  Set $a_{\max} = \arg\max_a Q(s_{j+1}, a_{j+1}, \theta_j)$
  Calculate the target Q-value as: $y_j = R_{j+1} + \gamma Q(s_{j+1}, a_{\max}, \theta_t')$
  Perform gradient descent step on $(y_t - Q(s_j, a_j; \theta_e))^2$ w.r.t. the network parameter θ_e
  $S_t \leftarrow S_{t+1}$
end for
end

To update the weights of the network during learning, it is necessary to obtain the Q values for the current state and the Q values for the state S_{t+1}. For this purpose, it is common to use two different neural networks. One is called the main network, used to estimate the Q values for the current state, and the other is called the target network, which estimates the Q values for the next state. The target network updates its weights by copying those of the main network every certain number of episodes, ensuring a delay compared to the main network to prevent overestimation of the Q values.

Another problem that may arise in this algorithm is that the correlation between the samples used to train the network can degrade the results. Therefore, a technique known as Experience Replay is used, which involves using a buffer, which is called G, to store obtained experiences. Random samples are then drawn from this buffer when training the main network.

Since the neural network's goal is to approximate the Bellman equation used in Q-learning, the objective function for the network is defined as follows:

$$y_t = R_{t+1} + \gamma Q\left(s_{t+1}, \arg\max_a Q\left(s_{t+1}, a; \theta\right); \theta'\right). \tag{14}$$

Where γ is the discount factor, S_t is the current state, S_{t+1} is the next state, and R_t is the reward obtained when taking action a_t in state S_t. θ and θ' are the weights of the main network and the target network, respectively. Thus, the loss function for training the main neural network is:

$$Loss = (y_t - Q\left(s_t, a; \theta\right))^2. \tag{15}$$

The algorithm was designed following a similar approach to the Edge Q-learning algorithm, where the edge layer estimates the overall cost of individual WSN node decisions based on their states, using the same action space, but using Deep Q-learning, which has been proved to work better than Q-learning solutions. As mentioned earlier, in this algorithm the state space and reward function were modified so that the algorithm also takes into account the battery difference.

- State: The system state is defined as: $S_t = \left(W_t, V_{n,k}, B_{n,k}\right)$ Where W_t and $B_{n,k}$ are the same parameters as in the Edge Q-learning algorithm, and the available bandwidth, $E_{n,k}$ indicates the SN's battery level, and $V_{n,k}$ indicate the battery difference between the node and the network.

Table 1
Experimental setup.

Parameters	Value
Number of tasks per SN	7–10
Tasks size	200–1500 bytes
Number of SNs in the network	10–80 SN
CPU frequency in SNs	16 MHz
CPU frequency in Edge device	500 MHz
Computational complexity	100 cycles/bit
E_t	0.1 W
E_p for 40% of the SNs	0.5 W
E_p for 20% of the SNs	[0.6, 0.75, 1] W
T_R	2.4 GHz/250 Kbps

Table 2
Parameters used for the Q-learning algorithms.

Parameter	Value
Learning rate α	0.1
Discount rate γ	0.9
Exploration rate ϵ	0.01
Collaboration tradeoff (Centralized) ψ	25
Collaboration tradeoff (Decentralized) ψ	20

- Reward: the reward function is a combination of the ones obtained in the previous algorithms. It considers the overall cost of the individual actions in the network, and also the battery-difference parameter which is used in CMA Q-learning.

$$\begin{cases} -C_{\text{all}} - \psi V_{n,t} & \text{if task is executed locally.} \\ -C_{\text{all}} + \psi V_{n,t} & \text{if task is offloaded.} \end{cases} \tag{16}$$

A simplified view of the implemented Deep Q-learning algorithm is represented in Algorithm 2. The main problem with this solution lies in the run-time. Unlike the Q-learning algorithm, where the table can be distributed among nodes after the learning process, allowing each node to make decisions independently without involving the edge, due to the low processing capacity of SNs, the weights of the neural network cannot be distributed among nodes once the learning process is complete. Consequently, they always have to depend on the edge layer and its decision-making capability, consistently waiting for the inference time.

6. Simulation results and analysis

To assess the performance of the proposed computation offloading scheme, simulation experiments were conducted on sensor networks with a variable number of SNs, in the range of [10, 80], with a CPU frequency of 16 MHz, and an edge frequency of 500 MHz.The available resources were set at the edge layer as the maximum number of bytes it can receive, and these vary, increasing as the number of SNs increases. The computational complexity is set at 100 cycles/bit for all tasks [39]. The power consumption during transmission, denoted as P^t, is 0.1 W. The processing power consumption, P^p, is distributed among the nodes as follows: 40% of the nodes use 0.5 W, while 20% of the nodes consume 0.6 W, 0.75 W, and 1 W, respectively. These values correspond to consumption levels zero, one, two, and three, listed in ascending order of power usage. This heterogeneity in energy consumption was intentionally introduced into the network to reflect the diverse tasks or operational requirements that nodes in a network may encounter, which, as noted earlier, can result in increased power consumption and consequently faster battery depletion. Each SN generates between seven and ten tasks, with each episode containing a variable number of bytes ranging from 200 to 1500. Both generative processes follow a stochastic pattern. These tasks must be processed before the next episode, which is considered to start at the beginning of the next duty cycle. The data transmission T_R rate has been established according to the IEEE standard for low-speed wireless networks (IEEE 802.15.4) in the 2.4 GHz band. The proposed strategies were implemented in Python on an HP computing system with a ninth-generation Intel Core i7 processor and a GTX 1650 graphics card. These setup parameters are summarized in Table 1.

In Tables 2 and 3 the parameters used in both Q-learning algorithms and Deep Q-learning algorithm are shown, respectively.

The three Q-learning algorithms described above were compared with a version of the decentralized Q-learning algorithm with ψ set to 0 which is called in this work Multi-Agent Q-learning (MAQL). In this way, each node seeks its own benefit, which results in every SN trying to offload tasks to the edge layer whenever the conditions allow it. Three different metrics were used to measure the efficiency of the algorithm:

- Total energy consumed by the network in each episode.
- Standard deviation of all the network batteries, to monitor whether an increase occurs in the difference between the batteries of different nodes due to their varying consumption and condition, or if the effect is mitigated.
- The progression of the batteries over the episodes.

Table 3
Parameters used for the Deep Q-learning algorithm.

Parameter	Value
Discount rate γ	0.9
Exploration rate ϵ	0.01
Collaboration tradeoff ψ	40
Batch size	32
Buffer size	10 000
Number of dense layers	3
Activation function	Relu

- Number of locally processed tasks on average in the last episodes.
- Total time spent on decision-making and processing of all network tasks in each episode.

Simulation experiments were conducted for 2000 episodes with 10, 20, 40, and 80 SNs. The exploration rate starts at 1 for all simulations and then gradually decreases until it reaches the rate shown in Tables 2 and 3 which marks the end of the learning phase. This strategy is implemented to speed up algorithm learning by encouraging more exploration at the outset.

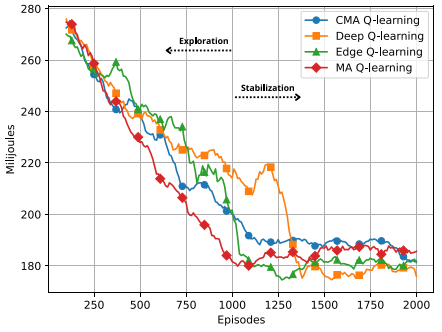
Regarding the reduction of overall network consumption in Figs. 3(a)–3(d), it can be observed that, generally, the algorithm achieving the best results at the end of the learning process is, as expected, the DQL algorithm. Although all algorithms exhibit similar consumption levels, CMA Q-learning has a slightly higher consumption, followed by MA Q-learning and Edge Q-learning. It is also noticeable that there is a slight decline in the total energy-saving performance in CMA-QL versus MA-QL due to the lack of perfect coordination among nodes deciding not to offload, thus not allowing those with less battery to offload their tasks. Before DQL, the algorithm that generally achieved better results was Edge Q-learning. However, Edge Q-learning's performance in this metric is noted to deteriorate compared to the other algorithms as the network size increases. This decline could be due to the increased complexity of estimating the cost for the entire network based on the individual actions of a single SN. Also, as the network size grows, there are many other actions from other network devices outside the control of the estimated SN, which influence the reward it obtains. This does not happen in DQL because estimating the cost of individual actions for the entire network becomes overly complex for Q-Learning, but not for a neural network. Notably, the two algorithms using a global reward exhibit higher consumption during the learning process until the exploration rate is set to 0.1, with a less linear behavior. This again makes sense, as the task of estimating the rewards is more complex.

Regarding the standard deviation shown in Figs. 5(a)–5(d), no notable differences are observed when increasing the number of SNs, and once again, the best results are achieved by DQL. This algorithm stabilizes the batteries at very close levels with a very low standard deviation. The CMA Q-learning algorithm also stabilizes the network's batteries, although with higher differences. On the other hand, MA Q-learning and Edge Q-learning, despite achieving similar overall consumption, do not include any incentive in their reward to encourage nodes with more battery to free up edge resources. This would allow nodes with higher consumption to better utilize these resources and consume less. Instead, they create a competitive environment for these resources, causing the battery differences to increase as the episodes progress continuously. Moreover, it is anticipated that both MA algorithms exhibit a similar trend in standard deviation as the first episodes progress. However, due to the MA-QL algorithm's decentralized competitive strategy, where each SN optimizes its benefits without regard for nodes with higher consumption rates, a continuous increase in standard deviation is witnessed across all network sizes. Conversely, in CMA-QL where there is some global knowledge of the network allowing agents to collaborate, the typical deviation between batteries stabilizes as the number of episodes increases.

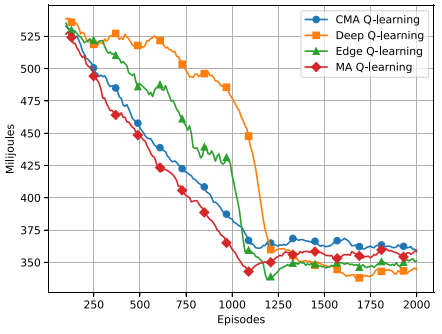
Fig. 4 shows the average number of tasks processed locally for each consumption group over the last 10 episodes of various simulations. The collaborative effect of the CMA Q-learning approach is clearly evident, as there is a distinct relationship between node consumption and the number of tasks processed locally. Specifically, nodes that consume more resources tend to process fewer tasks locally, while nodes that consume less choose to process more tasks locally to reduce the load on edge resources.

Additionally, it is observed that the standard deviation in Edge Q-learning is higher. This is because devices compete for resources and try to prevent other nodes from accessing edge resources, causing the likelihood of each node accessing these resources to depend more on the probability of finding them available at a given time compared to the CMA Q-learning algorithm. However, these variations tend to balance out across the total set of episodes. In contrast, the deviation between episodes is lower in CMA Q-learning, where a certain level of coordination and a deliberate release of edge resources by nodes with lower consumption results in less variability between episodes. It is also noted that the deviation appears to decrease slightly more in the higher consumption groups, as whether they process tasks locally or not has a greater impact compared to other groups.

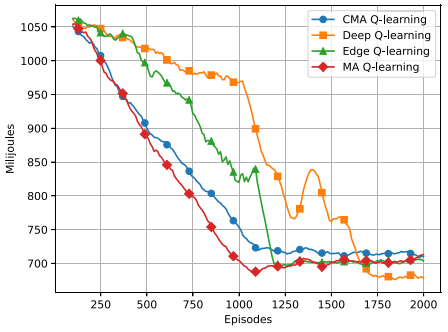
The previous results become more visible in Fig. 6, which shows the evolution of the batteries of all the nodes in the 10-SN network over the episodes. Comparing both evolutions, the expected positive effect of the previously studied factors can be observed. In the algorithm that does not consider collaboration, the batteries of the group of SNs with the highest consumption deplete much more rapidly, whereas, in CMA Q-learning, all the batteries are maintained at more similar mid-levels. The first scenario would lead to a situation where the network's lifespan is compromised according to the previously explained definition, thus affecting the useful life of the network. In contrast, the second scenario manages to extend this lifespan simply by adjusting the classical algorithm to achieve collaboration among the different agents. Despite this, they maintain a similar average energy level across all network nodes, with an average consumption of 6618.19 $\pm$0.18 Joules for CMA Q-learning and 6617.93 $\pm$0.56 Joules for Edge Q-learning.



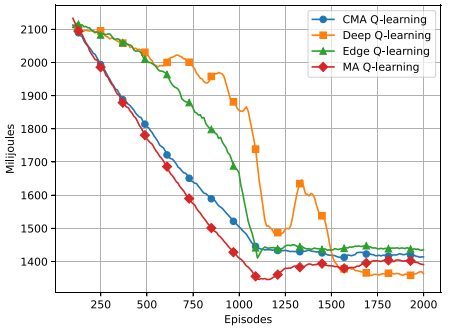
(a) Total energy consumed for 10 SNs.



(b) Total energy consumed for 20 SNs.

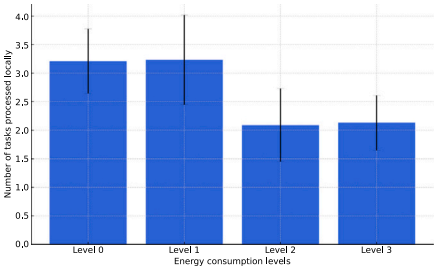


(c) Total energy consumed for 40 SNs.

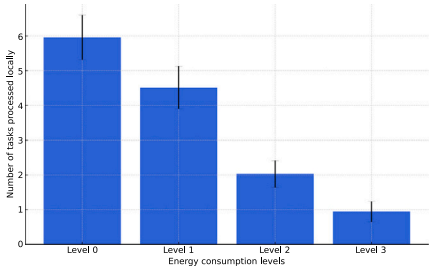


(d) Total energy consumed for 80 SNs.

Fig. 3. Total energy consumed for different sizes of SNs.



(a) Edge Q-learning



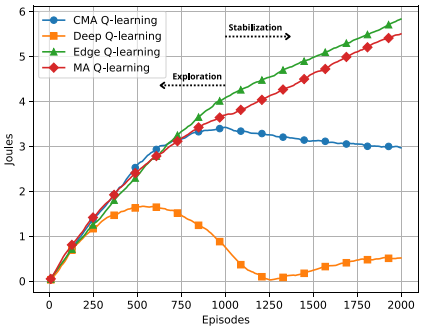
(b) CMA Q-learning

Fig. 4. Locally executed tasks in the last episodes for each consumption level.

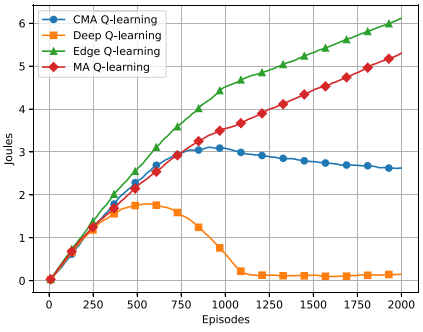
Table 4
Runtime of tasks in the final episodes.

SNs	CMAQL	MAQL	EQL	DQL
10	0.031s	0.021s	0.032s	0.094s
20	0.034s	0.021s	0.033s	0.16s
40	0.036s	0.025s	0.040s	0.31s
80	0.042s	0.026s	0.035s	1.65s

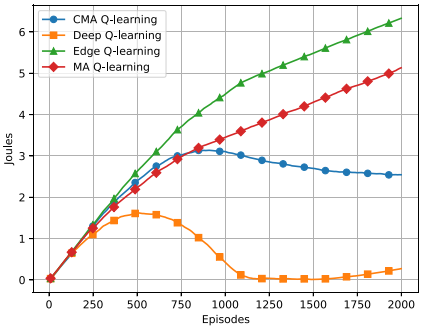
Although the computation offloading technique with Deep Q-learning seems to be the most effective in increasing the network's lifetime, it also requires a much higher duty cycle of SNs because the decision-making process takes much longer than in the Q-learning algorithms. In Table 4, the average execution times per episode achieved by the different algorithms were collected for each number of sensors, once the learning phase has been surpassed.



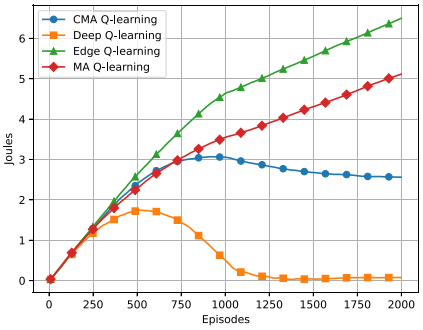
(a) Standard deviation for 10 SNs.



(b) Standard deviation for 20 SNs.

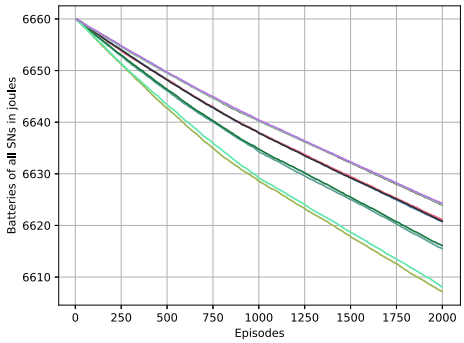


(c) Standard deviation for 40 SNs.

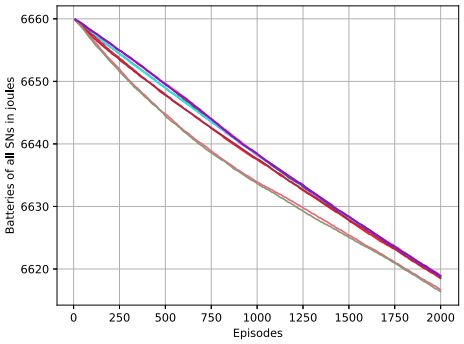


(d) Standard deviation for 80 SNs.

Fig. 5. Standard deviation for different sizes of SNs.



(a) Edge Q-learning



(b) CMA Q-learning

Fig. 6. Network batteries evolution over the episodes.

Despite achieving the best results in terms of overall energy consumption, with a 5% lower consumption for the 80-node network compared to CMA-QL, the Deep Q-learning algorithm may not be universally suitable due to the inference time required for each decision. As the number of SNs increases, the time for task processing significantly rises, becoming 39 times slower for the 80-node network compared to the CMA-QL solution. Consequently, Deep Q-learning proves optimal for applications not demanding high-speed data acquisition or low frequencies, in contrast to the Q-learning algorithms, which exhibit minimal decision-making time and are well-suited for high-speed networks with limited node capacity, where this time remains consistent regardless of the SN count, as each node independently makes decisions.

7. Conclusion and future work

This work shows how the classical Q-learning algorithm can be adapted to function in a distributed manner within Wireless Sensor Networks. This adaptation is straightforward enough to allow network devices to collaborate effectively, making it possible to maintain the battery values of the WSN close together, which could be of great importance in a sensor network depending on the energy and autonomy requirements. Additionally, it has been observed that, depending on the WSN application and network size requirements, a different algorithm should be chosen to obtain better results in terms of lifetime expansion, decision-making strategy, and SN resources.

The results demonstrate the advantages of achieving collaboration among the different agents quantitatively. While CMA achieves overall energy consumption results similar to other algorithms, it exhibits a stabilized standard deviation over time. In contrast to other decentralized Q-learning algorithms such as Edge Q-learning, where some devices end up with battery levels close to 6608 joules, CMA Q-learning produces very consistent results across all nodes, approximately 6618 joules, which roughly coincides with the average battery level in the Edge Q-learning results.

It is also clear that more research is needed to develop algorithms suitable for environments with limited computational capabilities, even at the edge node. This limitation makes it difficult to implement centralized algorithms such as Deep Q-learning, which are known to produce better results. Due to the research on the application of AI techniques in task offloading for WSN not being extensively explored, there are several further lines to be addressed. One is the comparison and benefits of these algorithms with other widely used heuristic techniques. Additionally, testing the proposed Q-Learning-based approaches in real IoT sensor devices will be another important avenue for exploration. Also, an increasingly popular alternative in this field is the Deep Deterministic Policy Gradient (DDPG) algorithm, which offers advantages over DQL by supporting continuous action spaces [40–42].

CRediT authorship contribution statement

Mateo Cumia: Writing – review & editing, Writing – original draft, Validation, Methodology, Investigation, Formal analysis, Data curation. **Gabriel Mujica:** Writing – review & editing, Writing – original draft, Validation, Supervision, Resources, Methodology, Investigation, Formal analysis, Conceptualization. **Jorge Portilla:** Writing – review & editing, Supervision, Resources, Project administration, Methodology, Funding acquisition.

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.

Acknowledgments

The publication is part of the TED2021-132475B-I00 project, funded by MCIN/AEI/10.13039/501100011033 and by the European Union “NextGenerationEU”/PRTR.

Data availability

Data will be made available on request.

References

[1] K. Shafique, B.A. Khawaja, F. Sabir, S. Qazi, M. Mustaqim, Internet of things (IoT) for next-generation smart systems: A review of current challenges, future trends and prospects for emerging 5G-IoT scenarios, IEEE Access 8 (2020) 23022–23040, http://dx.doi.org/10.1109/ACCESS.2020.2970118.
[2] S. Madakam, V. Lake, V. Lake, V. Lake, et al., Internet of things (IoT): A literature review, J. Comput. Commun. 3 (05) (2015) 164.
[3] L. Lin, X. Liao, H. Jin, P. Li, Computation offloading toward edge computing, Proc. IEEE 107 (8) (2019) 1584–1607.
[4] K. Cao, Y. Liu, G. Meng, Q. Sun, An overview on edge computing research, IEEE Access 8 (2020) 85714–85728, http://dx.doi.org/10.1109/ACCESS.2020.2991734.
[5] K. Kumar, J. Liu, Y.-H. Lu, B. Bhargava, A survey of computation offloading for mobile systems, Mob. Networks Appl. 18 (2013) 129–140.
[6] P. Mach, Z. Becvar, Mobile edge computing: A survey on architecture and computation offloading, IEEE Commun. Surv. Tutorials 19 (3) (2017) 1628–1656, http://dx.doi.org/10.1109/COMST.2017.2682318.
[7] J. Portilla, G. Mujica, J.-S. Lee, T. Riesgo, The extreme edge at the bottom of the internet of things: A review, IEEE Sensors J. 19 (9) (2019) 3179–3190.
[8] P. Merino, G. Mujica, J. Señor, J. Portilla, A modular IoT hardware platform for distributed and secured extreme edge computing, Electronics 9 (3) (2020) http://dx.doi.org/10.3390/electronics9030538, URL: https://www.mdpi.com/2079-9292/9/3/538.
[9] D.R. Morales, H.E. Romeijn, The generalized assignment problem and extensions, Handb. Comb. Optim. (2005) 259–311.
[10] D.S. Hochba, Approximation algorithms for NP-hard problems, ACM Sigact News 28 (2) (1997) 40–52.
[11] D.S. Johnson, The NP-completeness column: An ongoing guide, J. Algorithms 8 (2) (1987) 285–303.
[12] H. Yetgin, K.T.K. Cheung, M. El-Hajjar, L.H. Hanzo, A survey of network lifetime maximization techniques in wireless sensor networks, IEEE Commun. Surv. Tutorials 19 (2) (2017) 828–854, http://dx.doi.org/10.1109/COMST.2017.2650979.
[13] C.G. Cassandras, T. Wang, S. Pourazarm, Optimal routing and energy allocation for lifetime maximization of wireless sensor networks with nonideal batteries, IEEE Trans. Control. Netw. Syst. 1 (1) (2014) 86–98.

[14] Y. Liu, Y. Wang, H. Long, H. Yang, Lifetime-aware battery allocation for wireless sensor network under cost constraints, IEICE Trans. Commun. 95 (5) (2012) 1651–1660.
[15] D. Hortelano, I. de Miguel, R.J.D. Barroso, J.C. Aguado, N. Merayo, L. Ruiz, A. Asensio, X. Masip-Bruin, P. Fernández, R.M. Lorenzo, et al., A comprehensive survey on reinforcement-learning-based computation offloading techniques in edge computing systems, J. Netw. Comput. Appl. 216 (2023) 103669.
[16] R. Yadav, W. Zhang, I.A. Elgendy, G. Dong, M. Shafiq, A.A. Laghari, S. Prakash, Smart healthcare: RL-based task offloading scheme for edge-enable sensor networks, IEEE Sensors J. 21 (22) (2021) 24910–24918.
[17] C. Li, J. Bai, Y. Chen, Y. Luo, Resource and replica management strategy for optimizing financial cost and user experience in edge cloud computing system, Inform. Sci. 516 (2020) 33–55, http://dx.doi.org/10.1016/j.ins.2019.12.049, URL: https://www.sciencedirect.com/science/article/pii/S0020025519311600.
[18] N.I.M. Enzai, M. Tang, A heuristic algorithm for multi-site computation offloading in mobile cloud computing, Procedia Comput. Sci. 80 (2016) 1232–1241.
[19] R. Singh, S. Armour, A. Khan, M. Sooriyabandara, G. Oikonomou, Heuristic approaches for computational offloading in multi-access edge computing networks, in: 2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications, 2020, pp. 1–7, http://dx.doi.org/10.1109/PIMRC48278.2020.9217181.
[20] Z. Zabihi, A.M. Eftekhari Moghadam, M.H. Rezvani, Reinforcement learning methods for computation offloading: A systematic review, ACM Comput. Surv. 56 (1) (2023) 1–41.
[21] M. Adhikari, M. Mukherjee, S.N. Srirama, DPTO: A deadline and priority-aware task offloading in fog computing framework leveraging multilevel feedback queueing, IEEE Internet Things J. 7 (7) (2020) 5773–5782, http://dx.doi.org/10.1109/JIOT.2019.2946426.
[22] M. Liu, Y. Liu, Price-based distributed offloading for mobile-edge computing with computation capacity constraints, IEEE Wirel. Commun. Lett. 7 (3) (2017) 420–423.
[23] Z. Chang, L. Liu, X. Guo, Q. Sheng, Dynamic resource allocation and computation offloading for IoT fog computing system, IEEE Trans. Ind. Informatics 17 (5) (2020) 3348–3357.
[24] H.Y. Huang, K.T. Kim, H.Y. Youn, Determining node duty cycle using Q-learning and linear regression for WSN, Front. Comput. Sci. 15 (2021) 1–7.
[25] W.-K. Yun, S.-J. Yoo, Q-learning-based data-aggregation-aware energy-efficient routing protocol for wireless sensor networks, IEEE Access 9 (2021) 10737–10750, http://dx.doi.org/10.1109/ACCESS.2021.3051360.
[26] Y. Zhou, T. Cao, W. Xiang, Anypath routing protocol design via Q-learning for underwater sensor networks, IEEE Internet Things J. 8 (10) (2021) 8173–8190, http://dx.doi.org/10.1109/JIOT.2020.3042901.
[27] G. Künzel, L.S. Indrusiak, C.E. Pereira, Latency and lifetime enhancements in industrial wireless sensor networks: A Q-learning approach for graph routing, IEEE Trans. Ind. Informatics 16 (8) (2020) 5617–5625, http://dx.doi.org/10.1109/TII.2019.2941771.
[28] S. Pan, Z. Zhang, Z. Zhang, D. Zeng, Dependency-aware computation offloading in mobile edge computing: A reinforcement learning approach, IEEE Access 7 (2019) 134742–134753, http://dx.doi.org/10.1109/ACCESS.2019.2942052.
[29] B. Dab, N. Aitsaadi, R. Langar, Q-learning algorithm for joint computation offloading and resource allocation in edge cloud, in: 2019 IFIP/IEEE Symposium on Integrated Network and Service Management, IM, IEEE, 2019, pp. 45–52.
[30] A. Robles-Enciso, A.F. Skarmeta, A multi-layer guided reinforcement learning-based tasks offloading in edge computing, Comput. Netw. 220 (2023) 109476.
[31] H. Maleki, M. Başaran, L. Durak-Ata, Reinforcement learning-based decision-making for vehicular edge computing, in: 2021 29th Signal Processing and Communications Applications Conference, SIU, 2021, pp. 1–4, http://dx.doi.org/10.1109/SIU53274.2021.9478026.
[32] H. Zhou, K. Jiang, X. Liu, X. Li, V.C.M. Leung, Deep reinforcement learning for energy-efficient computation offloading in mobile-edge computing, IEEE Internet Things J. 9 (2) (2022) 1517–1530, http://dx.doi.org/10.1109/JIOT.2021.3091142.
[33] J. Li, H. Gao, T. Lv, Y. Lu, Deep reinforcement learning based computation offloading and resource allocation for MEC, in: 2018 IEEE Wireless Communications and Networking Conference, WCNC, 2018, pp. 1–6, http://dx.doi.org/10.1109/WCNC.2018.8377343.
[34] Y. Li, F. Qi, Z. Wang, X. Yu, S. Shao, Distributed edge computing offloading algorithm based on deep reinforcement learning, IEEE Access 8 (2020) 85204–85215, http://dx.doi.org/10.1109/ACCESS.2020.2991773.
[35] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, W. Zhuang, Learning-based computation offloading for IoT devices with energy harvesting, IEEE Trans. Veh. Technol. 68 (2) (2019) 1930–1941, http://dx.doi.org/10.1109/TVT.2018.2890685.
[36] F. Samie, V. Tsoutsouras, L. Bauer, S. Xydis, D. Soudris, J. Henkel, Computation offloading and resource allocation for low-power IoT edge devices, in: 2016 IEEE 3rd World Forum on Internet of Things, WF-IoT, 2016, pp. 7–12, http://dx.doi.org/10.1109/WF-IoT.2016.7845499.
[37] T. Rashid, M. Samvelyan, C.S. De Witt, G. Farquhar, J. Foerster, S. Whiteson, Monotonic value function factorisation for deep multi-agent reinforcement learning, J. Mach. Learn. Res. 21 (178) (2020) 1–51.
[38] 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, IEEE, 2019, pp. 1–10.
[39] H. Li, K. Xiong, Y. Lu, B. Gao, P. Fan, K. Letaief, Distributed design of wireless powered fog computing networks with binary computation offloading, IEEE Trans. Mob. Comput. (2021).
[40] H. Gao, X. Wang, W. Wei, A. Al-Dulaimi, Y. Xu, Com-DDPG: Task offloading based on multiagent reinforcement learning for information-communication-enhanced mobile edge computing in the internet of vehicles, IEEE Trans. Veh. Technol. 73 (1) (2024) 348–361, http://dx.doi.org/10.1109/TVT.2023.3309321.
[41] H. Hu, D. Wu, F. Zhou, S. Jin, R.Q. Hu, Dynamic task offloading in MEC-enabled IoT networks: A hybrid DDPG-D3QN approach, in: 2021 IEEE Global Communications Conference, GLOBECOM, 2021, pp. 1–6, http://dx.doi.org/10.1109/GLOBECOM46510.2021.9685906.
[42] L. Ale, S.A. King, N. Zhang, A.R. Sattar, J. Skandaraniyam, D3PG: Dirichlet DDPG for task partitioning and offloading with constrained hybrid action space in mobile-edge computing, IEEE Internet Things J. 9 (19) (2022) 19260–19272, http://dx.doi.org/10.1109/JIOT.2022.3166110.