A Reinforcement Learning Approach for Multi-edge Task Offloading Through Bi-level Optimization

Mohammed Dhyia Eddine Gouaouri*,
Miloud Bagaa*,Oussama Bekkouche§, Messaoud Ahmed Ouameur * and Adlen Ksentini¶
* Université du Québec à Trois-Rivières, Trois-Rivières, QC, Canada.
Emails: {mohammed.dhyia.eddine.gouaouri, miloud.bagaa, messaoud.ahmed.ouameur}@uqtr.ca
, §Aalto University, Otakaari 24, 02150 Espoo FINLAND (e-mail: oussama.bekkouche@aalto.fi),
¶EURECOM, Campus SophiaTech, France (e-mail: adlen.ksentini@eurecom.fr)

***Abstract*—The Internet of Things (IoT) is rapidly expanding globally, but the limited size of IoT devices restricts their battery capacity, computational resources, and wireless bandwidth, making it difficult to handle resource-intensive tasks. Edge Computing addresses these challenges by enabling task offloading to more capable edge servers. However, optimal task offloading in Edge-IoT networks is complex due to dynamic conditions, such as varying server loads and wireless fluctuations. Traditional and some machine learning-based offloading methods often fall short in adaptability or efficiency. This paper introduces a bi-level optimization approach using Deep Reinforcement Learning (DRL) agents for IoT-level offloading and a priority-aware greedy heuristic for resource allocation on edge servers. The proposed method effectively improves QoS by balancing task execution latency and power consumption, as demonstrated by simulation results.**

***Index Terms*—Edge-IoT Computing, Task Offloading, Bi-level Optimization, Reinforcement Learning**

I. Introduction

The rapid proliferation of Internet of Things (IoT) devices has led to an exponential increase in the generation of data, presenting both opportunities and challenges for modern computing architectures [1]. Traditional cloud computing, with its centralized data centers, offers scalable resources and powerful analytics capabilities, but it is often constrained by issues such as network latency, bandwidth limitations, and data privacy concerns. In response to these challenges, the Cloud-Edge-IoT Continuum has emerged as a comprehensive framework that integrates cloud computing, edge computing, and IoT devices into a cohesive and adaptive system.

The Cloud-Edge-IoT Continuum (CEIC) represents a paradigm shift in the way data is processed, stored, and analyzed. By distributing computing resources across the cloud and the edge, and by leveraging the capabilities of IoT devices, this continuum enables efficient data management and real-time decision-making. Edge computing, positioned between the cloud and IoT devices, plays a critical role in this architecture by bringing processing power closer to the data source. This proximity reduces latency, minimizes bandwidth usage, and allows for local processing of critical data, which is particularly valuable in applications requiring immediate responses, such as autonomous vehicles, industrial automation, and healthcare monitoring. The integration of cloud, edge, and IoT systems also enhances scalability and flexibility, enabling organizations to dynamically allocate resources based on real-time demands. This distributed approach not only optimizes the performance of IoT applications but also addresses the growing concerns around data security and privacy by processing sensitive information closer to the point of origin.

However, IoT devices are known to be resource-constrained systems with limited battery capacity and computational resources, making it impossible to run resource-intensive applications. By leveraging the capabilities ofeEdge computing and performing the offloading of tasks to more powerful edge servers as necessary, QoS requirements such as reduced execution latency and lower power consumption can be satisfied.

Traditional task-offloading methods are often based on heuristics or simple rules, which may result in sub-optimal and less satisfactory solutions, in terms of the cost induced by the offloading strategy, as they may not adapt well to the variable conditions of the Edge-IoT networks, such as varying server loads and fluctuating wireless channel conditions. However, Deep Reinforcement Learning (DRL) techniques offer a promising approach to address these challenges.

In this paper, we propose a novel approach for multi-edge IoT task offloading and resource allocation using bi-level optimization. A reinforcement learning agent based on DDPG is employed to determine offloading decisions on local IoT devices, while a fast heuristic scheduling algorithm is designed to schedule tasks on remote edge servers and allocate resources to the offloaded tasks, aiming to reduce task execution latency and energy consumption.

The remainder of the paper is organized as follows. We review the task-offloading related work in Section II. The task offloading problem is formulated in Section III. Meanwhile, section IV describes the proposed solution. Section V presents the simulation results and their analysis, and finally, Section VI concludes the paper.

II. Related work

Task offloading in wireless networks, especially for resource-constrained devices, has been extensively studied. Traditional methods often rely on game theory [2], linear regression [3], and dynamic programming [4]. While these approaches can approximate optimal solutions, they are limited in dynamic task-offloading scenarios due to their reliance on prior knowledge of the environment. Many prior studies focus on optimizing either energy consumption or task execution latency [5], while some address both objectives [6]. Others aim to optimize additional goals, such as load balancing [7] and deployment cost [6]. DRL has proven effective for complex decision-making tasks like task offloading and resource allocation in wireless networks. For instance, [8] formulated task offloading for mobile users as a single-agent infinite-horizon

Markov Decision Process (MDP) to minimize monetary costs and energy consumption. In [9], a single-agent ϵ-greedy Q-learning algorithm was used to achieve a better trade-off between execution latency and power consumption for both the end device and the edge server. However, this solution assumes that only channel conditions, task queue status, and available computation resources need to be considered for solving the task offloading problem, neglecting the fact that power consumption of devices plays an important in the offloading decision.

Some recent efforts utilize federated learning (FL) and DRL in wireless networks. For example, [10] proposed a federated DDPG-based offloading method with power control in Vehicular Edge Computing (VEC), using DDPG to handle continuous actions representing transmission power allocation. Similarly, [11] introduced a federated-DRL (F-DRL) approach, where base stations collaborate by sharing model weights to allocate optimal transmit power. In [12], a federated reinforcement learning approach based on DQN was proposed for power allocation and task offloading. The method allocates discrete offloading power to a remote edge server for incoming tasks and has demonstrated superior performance compared to central DQN in terms of execution latency and power consumption.

However, most of these works do not consider the scenario of resource placement and allocation across multiple edge servers, where a wide range of tasks must be distributed. This scenario impacts both latency and energy consumption for local devices and remote edge servers. In this work, we propose a novel bi-level optimization approach for multi-edge task offloading and compute resource allocation in Edge-IoT environments. The outer-level optimization at the IoT device level is modeled as a Markov Decision Process (MDP) and solved using reinforcement learning techniques to manage task offloading and resource allocation decisions. The inner-level optimization at the edge server level is handled using a greedy algorithm to allocate resources efficiently. By integrating both approaches, we optimize task execution delay and energy consumption jointly.

III. Problem Formulation

A. System model

We consider an Edge-IoT environment as shown in Fig. 1, where IoT devices connect via a wireless cellular network to an EC. These devices have limited computation power and energy, making task offloading to the edge important for meeting QoS requirements, particularly reducing execution delay and power consumption. The edge cloud comprises multiple servers that process offloaded tasks. A task placement controller assigns tasks to the optimal server and allocates resources to minimize computation latency and energy consumption.

B. Communication model

In this study, we divide the time horizon into consecutive epochs of fixed duration, indexed by an integer t, where $0 < t \leq T$, with T representing the total number of epochs within the considered time frame.

We assume communication between IoT devices and edge servers is established via sub-gigahertz radio frequency technology. This ensures reliable, long-range communication with

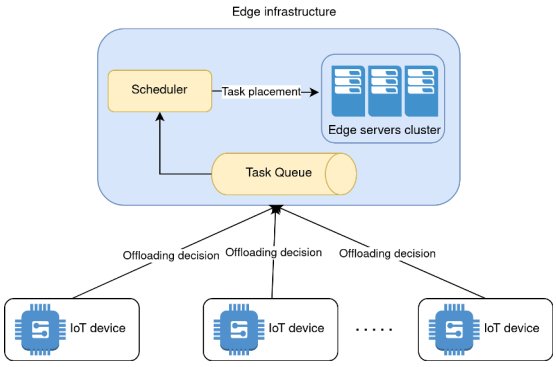

Fig. 1: Task offloading system architecture

low power consumption, making it ideal for IoT applications where devices are typically battery-powered and require extended operational lifespans. The communication bandwidth between each IoT device and the edge server is denoted as B_e.

We consider a set of stationary IoT devices, denoted by $D = \{d_1, d_2, ..., d_u\}$, each connected to the edge cloud access point through a wireless communication link with bandwidth B_e. The channel condition between an IoT device and the edge server varies over time, modeling realistic communication scenarios. This variation is captured by the channel gain, denoted G_t, which remains constant throughout a given offloading epoch. At any time, the channel gain is selected from a set $G = \{g_1, g_2, g_3, ..., g_n\}$.

Each IoT device makes autonomous decisions on task offloading based on multiple factors, including the number of queued tasks, the current channel gain, the device's remaining battery power, and its available computational resources. These decisions involve selecting the appropriate transmission power or determining the computation percentage allocated to each task.

C. Task model

We assume that each IoT device d_i maintains a task queue $Q^{(i)} = \{T_1^{(i)}, T_2^{(i)}, ..., T_M^{(i)}\}$, consisting of independent computational tasks. Each task in the queue requires a number of CPU cycles to process, depending on its size. At any given time t, the device receives a set of new user tasks. Each task $T_j^{(i)}$ has a size denoted by $s_j^{(i)} \in S$, measured in bytes, where S is the set of possible task sizes. Additionally, each task is assigned a priority value between 0 and 1 denoted by $U_j^{(i)}$, as well as a deadline $D_j^{(i)}$. If a task remains in the queue beyond its specified deadline, it is deemed to have exceeded its time limit and it is therefore canceled.

D. Computation model

1) Local execution: We consider a heterogeneous IoT environment where each device has its own computational capacity denoted by $C_d^{(i)}$ which measures the number of CPU cycles to process one byte of a computational task. Each CPU cycle consumes an amount of power that we denote $P_d^{(i)}$. Each device is attributed a task processing frequency denoted by $F_d^{(i)}$.

The power consumption required to process 1 byte of a task is given by $P_d^{(i)} \cdot C_d^{(i)}$. Hence, the power consumption for a task j is:

$$P_l^{(i,j)} = P_d^{(i)} \cdot C_d^{(i)} \cdot s_j^{(i)} \cdot r_j^{(i)} \quad (1)$$

where $r_j^{(i)}$ denotes the percentage of compute resources allocated to task $T_j^{(i)}$.

The local processing delay of a task $T_j^{(i)}$ is calculated by:

$$t_l^{(i,j)} = \frac{C_d^{(i)} \cdot s_j^{(i)} \cdot r_j^{(i)}}{F_d^{(i)}}. \tag{2}$$

We define the local processing cost function $L_l^{(i,j)}$ defined as:

$$L_l^{(i,j)} = (1-\alpha) \cdot P_l^{(i,j)} + \alpha \cdot t_l^{(i,j)} \tag{3}$$

where $\alpha \in [0,1]$ is mixing factor.

2) Offloading and remote execution:: The IoT devices are assumed to use the time division multiple access (TDMA) method to send their data to the edge server. This means that the IoT devices are given specific time slots to transmit data without the other devices interfering, allowing for efficient use of the channel and reducing collisions. The transmit power of the IoT device needed to offload the task to the remote servers is proportional to its size $s_j^{(i)}$. Hence, the resulting achievable transmission rate, expressed in bytes per second, is indicated as:

$$R_e^{(i,j)} = B_e \cdot \log(1 + \frac{s_j^{(i)} \cdot G}{\sigma^2}) \tag{4}$$

Where:

- $R_e^{(i,j)}$ represents the transmission rate between the end device and the edge cloud (measured in bytes per second).
- B_e represents the bandwidth of the channel between the IoT device and the edge servers (measured in Hertz).
- G represents the channel gain.
- σ^2 represents the variance of additive white Gaussian noise (AWGN).
- $s_j^{(i)}$ represents the size of task $T_j^{(i)}$.

We consider the edge cloud to be constituted of a set of heterogeneous edge servers $E = \{e_1, e_2, ..., e_M\}$ which are designed to handle relatively higher workload than IoT devices. Each edge server e_k has a compute capacity $C_e^{(k)}$ which denotes the number of CPU cycles to process one byte of a task computation. One CPU cycle of a server consumes $P_e^{(k)}$. The compute frequency is denoted by $F_e^{(k)}$.

If a task $T_j^{(i)}$ of size $s_j^{(i)}$ is assigned to server e_k then the processing time is calculated via the following formulas:

$$t_r^{(i,j,k)} = \frac{C_e^{(k)} \cdot s_j^{(i)} \cdot r_j^{(i,k)}}{F_e^{(k)}} \tag{5}$$

where $r_j^{(i,k)}$ is the compute percentage allocated to the task $T_j^{(i)}$ by the edge server e_k.

The consumed processing power is:

$$P_r^{(i,j,k)} = P_e^{(k)} \cdot C_e^{(k)} \cdot s_j^{(i)} \cdot r_j^{(i,k)} \tag{6}$$

We denote by $t_{trans}^{(i,j)}$ the transmission time when the task $T_j^{(i)}$ is offloaded:

$$t_{tran}^{(i,j)} = \frac{s_j^{(i)}}{R_e^{(i,j)}} \tag{7}$$

Hence, the remote processing cost is:

$$L_r^{(i,j,k)} = (1-\beta) \cdot P_r^{(i,j,k)} + \beta \cdot (t_{trans}^{(i,j)} + t_r^{(i,j,k)}) \tag{8}$$

Where β is a mixing factor.

A task scheduler deployed on the edge cloud makes decisions of placing offloaded tasks onto the proper edge servers. We define two binary variables $x_j^{(i)}$, $y^{(i,j,k)}$:

$$x_j^{(i)} = \begin{cases} 1, & \text{if task } T_j^{(i)} \text{ is offloaded to the edge} \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

$$y^{(i,j,k)} = \begin{cases} 1, & \text{if task } T_j^{(i)} \text{ is placed on edge server } e_k \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

We define the objective function:

$$\sum_i \sum_j \sum_k L_r^{(i,j,k)} \cdot y^{(i,j,k)} \cdot x_j^{(i)} + (1 - x_j^{(i)}) \cdot L_l^{(i,j)} \tag{11}$$

The goal is to minimize the objective defined in (11) as follows:

$$\begin{aligned} \text{minimize} \quad & \textstyle\sum_i \sum_j \sum_k \left[L_r^{(i,j,k)} \cdot y^{(i,j,k)} \cdot x_j^{(i)} + (1 - x_j^{(i)}) \cdot L_l^{(i,j)} \right] \\ \text{subject to} \quad & 1) \quad 0 \le r_j^{(i)} \le 1, \quad \forall i,j \\ & 2) \quad 0 \le r_j^{(i,k)} \le 1, \quad \forall k,j \\ & 3) \quad x_j^{(i)} \in \{0,1\}, \quad \forall i,j \\ & 4) \quad y^{(i,j,k)} \in \{0,1\}, \quad \forall i,j,k \\ & 5) \quad \textstyle\sum_k y^{(i,j,k)} = x_j^{(i)}, \quad \forall i,j \\ & 6) \quad \textstyle\sum_j r_j^{(i)} \cdot (1 - x_j^{(i)}) \le 1, \quad \forall i \\ & 7) \quad \textstyle\sum_i \sum_j r_j^{(i,k)} \cdot y^{(i,j,k)} \le 1, \quad \forall k \end{aligned} \tag{12}$$

The constraints in this optimization problem 12 ensure a feasible task offloading and resource allocation. The **local computation capacity constraint** (1) ensures that the percentage of computation resources allocated to each task on the device is between 0 and 1, representing the fraction of resources used locally. Similarly, the **remote computation capacity constraint** (2) restricts the resource allocation on the edge server, ensuring that tasks offloaded to a server do not exceed available resources. The **binary variables constraints** (3) and (4) define $x_j^{(i)}$ as a binary variable indicating whether a task is offloaded 1 or processed locally 0, and $y^{(i,j,k)}$, also a binary variable that indicates whether a task is assigned to a specific edge server. The **task assignment constraint** (5) ensures that each task can either be processed locally or offloaded, but not both, and that the sum of assignments to edge servers matches the offloading decision. Finally, the **computation capacity constraints** (6) and (7) ensure that the total resources allocated to tasks on both devices and edge servers do not exceed the available computation capacity, maintaining an overall balance of resource usage.

We notice that this problem is a mixed integer nonlinear programming (MINLP) problem that involves nonlinear objective functions and integer variables, making it difficult or even impossible to solve using conventional optimization techniques. Furthermore, traditional nonlinear programming optimization techniques take a considerable amount of time before giving the optimal solution, making them unfeasible for real-time, whereby the offloading decisions should be taken in real-time. Reinforcement Learning (RL) is a promising approach to tackle this problem, as it provides a framework

for devising optimized decision-making policies adapted to the dynamic of the environment.

IV. Reinforcement learning formulation and methodology

In this section, we propose an RL formulation for the task offloading and resource allocation problem described earlier and present an algorithm to solve it.

*1) **Offloading RL Formulation:** :* The task offloading problem is first modeled as a Markov Decision Process (MDP) $< S^{(i)}, A^{(i)}, P^{(i)}, R^{(i)}, \gamma^{(i)} >$ for each agent deployed on the device d_i.

a) Elements of the MDP:

- **State Space:** $S^{(i)} = \{s_1^{(i)}, s_2^{(i)}, \ldots, s_n^{(i)}\}$ represents the set of states for device i. Each state $s_t^{(i)}$ encapsulates information regarding the task offloading environment of device i at time slot t, including:
  - **Channel gain ($G_t^{(i)}$):** Reflects the quality of the communication channel between the device and the edge servers.
  - **Task queue ($Q_t^{(i)}$):** Represents the list of the local tasks of device i at time slot t, waiting to be offloaded or processed, characterized by the size and number of tasks.
  - **Remaining battery power ($B_t^{(i)}$):** Indicates the battery level od device i at time slot t.
  - **Available computation resources ($R_t^{(i)}$):** Represents the remaining CPU or computation resources on the device i at time slot t.

  Thus, the state for agent i at time step t can be defined as:

$$s_t^{(i)} = (G_t^{(i)}, Q_t^{(i)}, R_t^{(i)}, B_t^{(i)})$$

  This state space reflects the dynamic environment of IoT devices, where the channel conditions, task load, and resource availability fluctuate over time.

- **Action Space ($A^{(i)}$):** The action space includes the set of actions taken by the agent regarding task offloading and resource allocation. Each action can be represented as a tuple $a_t^{(i)} = (x_j^{(i)}, r_j^{(i)})$, where:
  - $x_j^{(i)}$: A binary variable indicating whether a task is offloaded or not.
  - $r_j^{(i)}$: The amount of computation resources allocated to the task, if executed locally.
  - Task $T_j^{(i)}$ is being treated at time step t.

  To reduce the action space complexity, we observe that $x_j^{(i)} = \mathbb{I}_{[r>0]}(r_j^{(i)})$ (where $\mathbb{I}_{[x>0]}(x)$ is the indicator function), meaning the offloading decision $x_j^{(i)}$ depends on whether $r_j^{(i)}$ is positive. Therefore, the agent's action is reduced to deciding the allocated computation resources $a_t^{(i)} = r_j^{(i)}$.
- **Transition Probability ($P^{(i)}$):** The state transition probability $P^{(i)}(s_{t+1}^{(i)}, R_t | s_t^{(i)}, a_t^{(i)})$ represents the probability distribution over the next state $s_{t+1}^{(i)}$ and the reward R_t given the current state $s_t^{(i)}$ and action $a_t^{(i)}$ taken by the agent. These transition probabilities are unknown due to the random dynamics of the environment, which makes the use dynamic programming infeasible. We use Reinforcement Learning (RL) to iteratively learn optimal policies by interacting with the environment, enabling the agent to discover the transition probabilities and rewards through experience.
- **Reward function ($R^{(i)}$):** The reward function $R^{(i)}$ is defined to encourage efficient resource allocation and energy consumption. The reward function is formulated rigorously in IV-2
- **Discount Factor ($\gamma^{(i)}$):** The discount factor $\gamma^{(i)}$ determines the importance of future rewards. The higher the value of $\gamma^{(i)}$ is, the more important future rewards are, encouraging long-term planning, while a lower $\gamma^{(i)}$ prioritizes immediate gains.

2) Bi-level optimization formulation: At each timestep t, each agent deployed on the device i takes an action $a_t^{(i)} = r_t^{(i)}$ from a local policy π_i. A local reward is received which is equal to:

$$R_t^{(i)}(s_t^i, a_t^i) = -L_l^{(i,j)} \cdot (1 - x_j^{(i)}) + (D_j^{(i)} - t) \cdot U_j^{(i)}$$

Given that at time step t, task $T_j^{(i)}$ is being treated.

The offloading decisions are communicated to the edge task scheduler that reads the placed tasks from the queue and assigns them to the edge servers using the algorithm described in IV-3. The scheduler, then, calculates the cost of remote processing of each task $T_j^{(i)}$ as $\sum_k L_r^{(i,j,k)} \cdot y^{(i,j,k)}$. The cost of remote processing of tasks is used for calculating the global reward as follows:

$$R_t(s_t, a_t, \lambda_t^*(s_t, a_t)) = \sum_i R_t^{(i)} + \sum_i \sum_j \sum_k -L_r^{(i,j,k)} \cdot y^{(i,j,k)}$$

Where:

- $s_t = (s_t^{(1)}, s_t^{(2)}, ..., s_t^{(i)}, ..., s_t^{(n)})$: The joint state of the multi-agent system.
- $a_t = (a_t^{(1)}, a_t^{(2)}, ..., a_t^{(i)}, ..., a_t^{(n)})$: The joint action of the multi-agent system.
- λ_t is the remote task placement decision on the edge server at time step t and λ_t^* is the optimal remote placement.

The global reward is then returned to the devices' agents to perform the Bellman update. Hence, we can formulate this joint problem as a bi-level optimization problem as follows:

$$\begin{aligned} \max_{\pi_i} \quad & \mathbb{E}_{a_t^{(i)} \sim \pi_i(\cdot|s_t)} \left[\sum_{t=0}^{T} \gamma^t R_t(s_t, a_t, \lambda_t^*(s_t, a_t)) \right] \\ \text{subject to} \quad & \lambda_t^*(s_t, a_t) = \arg\max_{\lambda_t} R_t(s_t, a_t, \lambda_t(s_t, a_t)) \end{aligned} \tag{13}$$

The equation above aims to maximize the expected discounted cumulative reward for each IoT agent i (the outer level objective), taking into account an optimal placement and resource allocation on the edge servers (the inner level objective).

We employ Deep Deterministic Policy Gradient (DDPG) algorithm [13] to learn the optimal offloading and resource allocation policy, the choice of DDPG is driven by the fact that our state and action spaces are continuous which limits the set of algorithms to be employed. On the other hand, DDPG is considered a robust learning algorithm and has demonstrated superiority in various research problems [14], [15]. The DDPG

algorithm that we use leverages two neural networks: an *actor network* that outputs the optimal action for a given state and a *critic network* that evaluates the quality of the action by predicting the expected future rewards. By using experience replay and soft target updates, DDPG effectively stabilizes training and improves convergence.

*3) **Remote task placement and resource allocation formulation:** :* For remote task placement and resource allocation, we adopt a priority-aware greedy approach that balances resource usage across edge servers, aiming to minimize latency, reduce energy consumption, and ensure fairness across tasks of varying sizes and priorities, thus avoiding resource starvation. Algorithm 1 outlines the proposed method, which begins by sorting the tasks in the queue in non-increasing order, prioritizing first high-priority tasks and then those with larger sizes. This ensures that urgent or resource-intensive tasks are processed first, enhancing system responsiveness to critical tasks. For each task, the least loaded server is selected for task offloading, with ties resolved randomly to avoid biases. This strategy distributes the load evenly across servers, reducing the risk of overloading any single server. Once a task is assigned to a server, a temperature-softmax score is calculated to estimate resource allocation for the remaining tasks. The softmax function transforms task sizes into probabilities, ensuring a smooth resource distribution based on task sizes, and is applied pessimistically by assuming that all remaining, smaller tasks will also be assigned to the same server. The temperature parameter τ controls the balance between fairness and task size preference, with smaller τ values allocating more resources to larger tasks and larger τ values ensuring more equitable resource distribution. Typically, τ is set to the median task size to strike a balance between allocating resources to large and small tasks. Finally, the allocated computation resources are determined by multiplying the softmax score by the fraction of remaining server resources, ensuring that no server is over-allocated while efficiently utilizing available resources. This dynamic resource allocation balances optimal resource usage, task fairness, and energy efficiency.

Algorithm 1: Greedy Approach for Remote Task Placement and Resource Allocation

Input: Set of tasks $\mathcal{T}$, Set of servers $\mathcal{S}$
Output: Task placements and allocated resources

Sort tasks $\mathcal{T}$ in non-increasing order based on priority and size;
Initialize placement array $p \leftarrow []$;
for *each task* $T_j \in \mathcal{T}$ **do**
    // Find the least loaded server
    $y_j \leftarrow \text{LeastLoaded}(\mathcal{S})$;
    // Softmax-based allocation factor
    $f_j \leftarrow \frac{\exp\left(\frac{s_j}{\tau}\right)}{\sum_{k \geq j} \exp\left(\frac{s_k}{\tau}\right)}$;
    // Assign server and resources to task
    $r_j \leftarrow f_j \cdot \text{RemainingResources}(\mathcal{S}[y_j])$;
    $p.\text{append}((y_j, r_j))$;
return p;

As discussed previously After performing the remote task placement and resource allocation for all of the tasks in the remote queue, the reward $R_t(s_t, a_t, \lambda_t(s_t, a_t))$ is calculated and sent back to the IoT agent to perform their Reinforcement Learning update.

V. Simulation and results

We conducted a simulation study to assess the performance of the proposed RL-based Multi-Edge task offloading solution. We consider an environment with 8 devices and 4 edge servers with the parameters shown in Table I. Some parameter values for simulating sub-gigahertz communication were taken from the experiments in [12]

The DDPG RL agent are developed using the PyTorch framework and trained over 1000 episodes, with each episode comprising 40 epochs (simulation time steps). The task generation process was modeled stochastically following Poisson distribution, with an arrival rate of 5 tasks per second. The initial power of the IoT device was set to the maximum value of 30 dB.

To evaluate the method, we measured the following key metrics:

- **Running Average Reward**: Tracks the agents' cumulative performance over time, averaged over 100 episodes, to assess adaptation in task arrival, resource allocation, and offloading.
- **Power Consumed by Devices**: Measures IoT device energy use for local task execution, including computation and communication costs, to ensure energy efficiency.
- **Remote Power Consumed by Edge Servers**: Monitors energy used by edge servers for offloaded tasks, covering both computation and data transmission, to evaluate offloading efficiency.

A. Running average of local IoT agents reward

Figure 2 illustrates the moving average reward (The learning curve) with a window size of 100 for the agents. Initially, the reward function exhibits significant noise during the early training stages, indicating the exploration behavior of the agents. However, after approximately 200 episodes, the reward function increases more smoothly, reflecting the agents' gradual improvement in task offloading and resource allocation strategies as they learn more efficient policies.

B. Running average of local IoT agents power consumption

Figure 3 illustrates the moving average local power consumption with a window size of 100 for the agents. In the initial stages of training, the power consumption shows significant noise, reflecting the exploration process. However, after around 200 episodes, the power allocation starts to decrease smoothly, indicating that the agents are learning more energy-efficient resource allocation strategies over time.

C. Running Average of Remote Edge Servers Power Consumption

We measure the remote power consumption of the edge servers, as depicted in figure 4. Initially, we observed relatively high power consumption due to the significant amount of tasks being offloaded. As the agents' policies converge to an optimal strategy, the power consumption stabilizes and the tasks are distributed more evenly across the edge servers within the edge-cloud. This reflects improved load balancing and more efficient resource utilization over time.

Parameters	Description	Value
S	Task sizes set	[5, 10, 15, ..., 1000] Kbytes (step 5)
B_e	Communication bandwidth	10^5
σ	Variance of AWGN	$-160 + 10 \cdot \log_{10}$
α, β	Mixing factors for costs	$\alpha = 0.8$, $\beta = 0.8$
G	Channel Gain	[0.5, 0.6, ..., 2] (step 0.1)
γ	Discount factor	0.95

Parameters	Description	Value
$C_d^{(i)}$	Device processing capacity	400 cycles/byte
$C_e^{(k)}$	Server processing capacity	4000 cycles/byte
$P_d^{(i)}$	Device power consumption	6×10^{-3} W
$P_e^{(k)}$	Server power consumption	1×10^{-6} W
$F_d^{(k)}$	Device frequency	500 MHz
$F_e^{(k)}$	Server frequency	4 GHz

TABLE I: Simulation Parameters For Multi-Edge Task Offloading

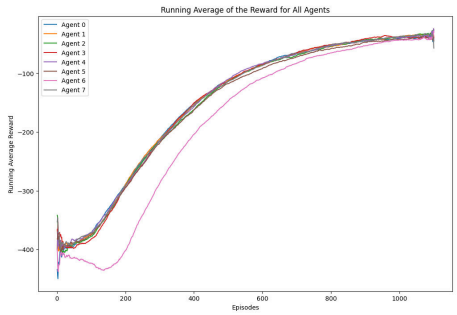

Fig. 2: Running average reward of the previous 100 episodes

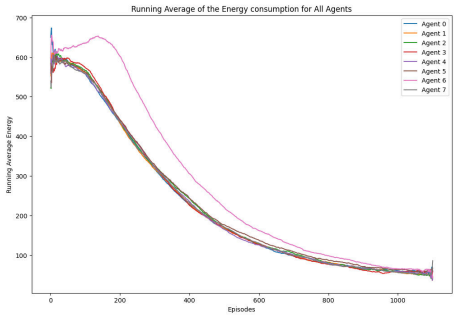

Fig. 3: Running average of local power consumption over the previous 100 episodes

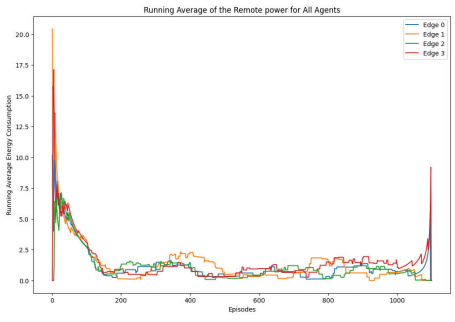

Fig. 4: Running average of remote power consumption over the previous 100 episodes

D. Analysis summary

The previous analysis confirms that the proposed multi-edge task offloading and resource allocation strategy is effective and robust. The DDPG-based reinforcement learning agent, coupled with the greedy allocation strategy, demonstrates significant improvements in reward, power consumption, and load balancing. The system's performance under various synthetic scenarios indicates its suitability for real-world applications, with potential for further optimization and refinement.

VI. Conclusion

In this paper, we proposed a novel bi-level optimization approach for multi-edge task offloading and resource allocation in Edge-IoT environments. The outer-level optimization at the IoT device level is addressed using a reinforcement learning-based optimizer, specifically DDPG, chosen for its robustness and ability to handle the complexity of our environment. The inner-level optimization at the edge server level is addressed with a greedy algorithm. The simulation results demonstrated promising performance, effectively balancing task processing latency and energy consumption. The greedy allocation algorithm on the remote edge servers also achieved acceptable fairness among tasks, improved load balancing, reduced energy consumption, and provided faster placement and allocation. Future work could explore broader problem setups, including mobile IoT devices, and investigate alternative scheduling algorithms, such as meta-heuristics, and other reinforcement learning-based approaches.

References

[1] A. Ghasempour, "Internet of things in smart grid: Architecture, applications, services, key technologies, and challenges," *Inventions*, vol. 4, no. 1, p. 22, 2019.

[2] S. Chen, S. Sun, H. Chen, J. Ruan, and Z. Wang, "A game theoretic approach to task offloading for multi-data-source tasks in mobile edge computing," in *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. IEEE, 2021, pp. 776–784.

[3] K.-H. Kim, J. Lynskey, S. Kang, and C. S. Hong, "Prediction based subtask offloading in mobile edge computing," in *Proceedings of the 2019 International Conference on Information Networking (ICOIN)*. IEEE, 2019, pp. 448–452.

[4] T. Zhao, S. Zhou, L. Song, Z. Jiang, X. Guo, and Z. Niu, "Energy-optimal and delay-bounded computation offloading in mobile edge computing with heterogeneous clouds," in *China Commun*, vol. 17, 2020, pp. 191–210.

[5] X. He, H. Xing, Y. Chen, and A. Nallanathan, "Energy-efficient mobile-edge computation offloading for applications with shared data," in *arXiv preprint arXiv:1809.00966*, 2018.

[6] Y. Nan, W. Li, W. Bao, F. Delicato, P. Pires, Y. Dou, and A. Zomaya, "Adaptive energy-aware computation offloading for cloud of things systems," *IEEE Access*, vol. 5, pp. 23 947–23 957, 2017.

[7] M. Avgeris, D. Dechouniotis, N. Athanasopoulos, and S. Papavassiliou, "Adaptive resource allocation for computation offloading: A control-theoretic approach," *ACM Transactions on Internet Technology (TOIT)*, vol. 19, no. 2, pp. 1–20, 2019.

[8] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Performance optimization in mobile-edge computing via deep reinforcement learning," *arXiv preprint arXiv:1804.00514*, 2018.

[9] X. Liu, Z. Qin, and Y. Gao, "Resource allocation for edge computing in iot networks via reinforcement learning," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, May 2019, pp. 1–6.

[10] S. Moon and Y. Lim, "Federated deep reinforcement learning based task offloading with power control in vehicular edge computing," *Sensors (Basel)*, vol. 22, no. 24, p. 9595, Dec. 2022.

[11] P. Tehrani, F. Restuccia, and M. Levorato, "Federated deep reinforcement learning for the distributed control of nextg wireless networks," in *2021 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, Dec. 2021, pp. 248–253.

[12] H. Merakchi, M. Bagaa, A. O. Messaoud, A. Ksentini, and A. Sehad, "Federated deep reinforcement learning-based task offloading system in edge computing environment," in *GLOBECOM 2023-2023 IEEE Global Communications Conference*. IEEE, 2023, pp. 5580–5586.

[13] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, and Y. Tassa, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015. [Online]. Available: https://arxiv.org/abs/1509.02971

[14] S. Gu, E. Holly, T. P. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous data centers," in *Robotics: Science and Systems (RSS)*, 2017. [Online]. Available: https://arxiv.org/abs/1705.08009

[15] T. Haarnoja, A. Zhou, S. Agarwal, and S. Levine, "Soft actor-critic: Off-policy actor-critic for continuous control," in *International Conference on Machine Learning (ICML)*, 2018. [Online]. Available: https://arxiv.org/abs/1812.05905