Energy-Efficient Task Offloading using Reinforcement Learning for Dependent Tasks in Cloud-Edge-Device Systems

Seyedeh Negar Afrasiabi, Diala Naboulsi, Razvan Stanica

Energy-Efficient Task Offloading using Reinforcement Learning for Dependent Tasks in Cloud-Edge-Device Systems

Seyedeh Negar Afrasiabi
Software Engineering and IT Department
Ecole de technologie supérieure (ÉTS)
Montréal, Canada
seyedeh-negar.afrasiabi@etsmtl.ca

Diala Naboulsi
Software Engineering and IT Department
Ecole de technologie supérieure (ÉTS)
Montréal, Canada
diala.naboulsi@etsmtl.ca

Razvan Stanica
Telecommunications Department
INSA Lyon, Inria, CITI
Villeurbanne, France
razvan.stanica@insa-lyon.fr

***Abstract*—The advancement of 5G networks has enabled latency-sensitive, computation-intensive applications that demand high Quality of Service but result in significant energy consumption. Mobile devices, with limited computational and energy resources, struggle to meet these demands. While cloud computing offers powerful processing, its high latency makes it unsuitable for delay-sensitive applications. Mobile-Edge Computing (MEC) provides a solution by offloading tasks to nearby servers, but energy efficiency remains a critical challenge, especially in dynamic environments with mobile device mobility and task dependencies. This paper proposes an energy-efficient task offloading mechanism for MEC environments that leverages collaboration between cloud, edge, and mobile devices. We model the problem as a Markov Decision Process and apply Double Deep Reinforcement Learning to optimize task offloading while minimizing energy consumption. Simulation results demonstrate the effectiveness of our approach in achieving convergence and reducing energy consumption by 36% compared to other methods, while adapting to dynamic network conditions.**

***Index Terms*—Energy-efficiency, MEC, Task offloading, RL, Dependent Task.**

I. Introduction

The advent of 5G technology and the widespread adoption of mobile devices have significantly advanced delay-sensitive and computation-intensive applications such as online interactive games, autonomous driving, face recognition, and virtual/augmented reality. These applications impose stringent Quality of Service (QoS) standards, which often result in higher energy consumption compared to traditional applications. However, mobile devices face limitations in computational resources and energy capacity due to their physical size and economic constraints. This can become a bottleneck when handling extensive applications or sustaining a continuous energy supply. Although cloud computing offers substantial processing power, the latency introduced by long transmission distances makes it impractical for latency-sensitive applications. Consequently, Mobile-Edge Computing (MEC) has emerged as a promising paradigm within the 5G architecture, enabling mobile devices to offload computation and resource demands to nearby MEC servers [1].

Energy efficiency in the MEC environment is critical, given the limited processing power and battery capacity of mobile devices. Furthermore, MEC servers themselves can consume substantial amounts of energy, especially during peak demand periods, which contributes to increased operational costs [2], [3]. Moreover, the dynamic activity and mobility of end devices introduce additional challenges. As mobile devices move, their resource requirements fluctuate, necessitating frequent adjustments in resource allocation. This mobility may also lead to increased communication overhead and higher energy consumption as devices switch between MEC servers or between MEC and cloud resources. Furthermore, task dependencies, where one task relies on the output of another, complicate offloading decisions. For example, in a face recognition application, feature extraction cannot begin until face detection is completed. Offloading dependent tasks to different computing facilities can introduce data transmission delays and increase energy consumption [4]–[6].

Optimizing computation offloading under such conditions involves determining whether to execute tasks locally, offload them to a MEC server, or send them to a cloud server, while maintaining energy efficiency. The high dynamics of MEC environments, characterized by fluctuating computational resources and device mobility [7], make designing an optimal offloading strategy particularly challenging. In this context, Deep Reinforcement Learning (DRL), which integrates reinforcement learning (RL) with deep learning techniques, has shown promise in managing these time-varying environments [8], [9].

In recent years, research on energy efficiency in MEC environments has often overlooked critical factors, including mobile device mobility, the complexities of heterogeneous cloud-MEC infrastructures, and interdependencies among the application tasks of a mobile device. Our work addresses these gaps by developing an energy-efficient task offloading mechanism that facilitates collaboration across cloud, MEC, and mobile devices. Our primary objective is to minimize total energy consumption while dynamically adapting offloading decisions based on mobile device mobility patterns and effectively managing task dependencies. To address the dynamic nature of these environments, we employ DRL to learn the underlying network topology, optimize offloading strategies,

and ultimately enhance energy efficiency.

The main contributions of this work are summarized as follows:

- We formulate the task offloading problem in MEC environments with task dependencies, focusing on reducing total energy consumption.
- We model the problem using a Markov Decision Process (MDP) framework and propose a DRL approach to address it.
- We conduct extensive simulation experiments to evaluate the proposed solution, demonstrating its efficiency.

The remainder of this paper is organized as follows: Section II reviews related work. Section III describes the system model and problem formulation. Section IV details the proposed solution. Simulation results are presented in Section V, and Section VI concludes the paper.

II. Related Work

In recent years, MEC has garnered significant attention due to its ability to reduce processing latency and energy consumption for computationally intensive tasks. We review several works related to energy efficiency in MEC systems [2], [10]–[13].

In [2], the energy efficiency of computation offloading strategies in MEC-enabled Internet-of-Vehicles (IoV) networks is investigated. A Multi-Agent Deep Reinforcement Learning (MADRL) based energy efficiency maximization algorithm is proposed to optimize computation offloading strategies and achieve maximum energy efficiency. In [10], the authors explore the joint optimization problem of computation offloading, service caching, and resource allocation in a coordinated MEC platform. The problem is formulated as a Mixed-Integer Non-Linear Programming (MINLP) model, and an algorithm based on Deep Deterministic Policy Gradient (DDPG) is proposed to minimize the long-term energy consumption of the collaborative MEC system. The joint optimization of computation offloading and resource allocation in a time-varying multi-user MEC system is discussed in [11]. The authors formulate the problem as an MINLP to minimize the energy consumption of the MEC system, considering delay constraints and uncertain resource requirements. They propose a centralized Double Deep Q-learning (DDQL)-based algorithm to determine the optimal policy for server selection, task offloading, and resource allocation. In [12], an energy-aware offloading method for MEC in 5G heterogeneous networks is proposed, where each mobile device decides whether to offload its task to MEC servers or compute it locally, aiming to reduce energy consumption while ensuring that latency constraints are met. An energy-efficient resource allocation algorithm for MEC, called Power Migration Expand (PowMigExpand), is presented in [13]. This algorithm assigns user requests to optimal servers and allocates resources to user equipment efficiently. Their approach migrates user requests to new servers when necessary due to user mobility. Additionally, a deep learning algorithm is proposed to solve the multidimensional optimization problem numerically, reducing the computational overhead through a pre-trained network.

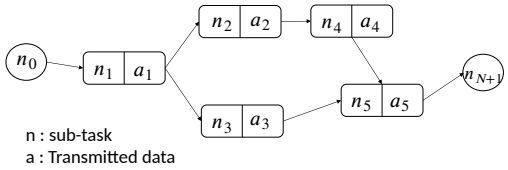

Fig. 1: An application dependencies graph with five dependent tasks.

The works [2], [10]–[13] generally assume the application to be atomic, meaning it consists of a single task with no dependent tasks. However, applications on mobile devices often consist of multiple interdependent tasks, which can be modeled as Directed Acyclic Graphs (DAGs). Furthermore, most of the existing works [10], [11], [13] assume static mobile devices, ignoring the mobility of devices and the heterogeneity of edge/cloud servers (which vary in processing and storage capabilities). To the best of our knowledge, none of these works address the energy consumption optimization problem for computation offloading in a multi-user MEC system, while considering both device mobility and task dependencies, where the output of one task serves as the input for another. This is the focus of our work.

III. System Model and Problem Formulation

In this section, we first present the system model of a multi-user end-edge-cloud heterogeneous network. We then formulate our problem, aiming to find a feasible offloading scheme that minimizes total energy consumption, including the energy required for computation, data transfer, and migration.

A. System Model

The system model comprises three types of computing facilities: mobile devices, MEC servers, and cloud servers. MEC servers are deployed at base stations, providing higher processing capacity and larger storage than mobile devices, which have limited capabilities. Mobile devices offload tasks required by applications to these nearby MEC servers, denoted by $P = \{1, 2, ..., |P|\}$. Lastly, the cloud servers, denoted by $Q = \{1, 2, ..., |Q|\}$, have relatively more computational resources and can receive all tasks for the required applications. There is a set of $K = P \cup Q$ servers in total. Hence, $|K| = |P| + |Q|$. The cloud and MEC servers are stationary, and their locations can be presented by (ℓ_p, φ_p) for MEC server p and (ℓ_q, φ_q) for cloud server q.

We assume that one or more mobile applications need to be executed at a given time. A group of mobile devices, denoted as $M = \{1, 2, ..., |M|\}$, is randomly distributed around their corresponding MEC servers (i.e. base stations), where $m \in M$ represents an individual mobile device within the group. Each mobile application can be divided into several tasks; for example, a face recognition application may consist of five dependent tasks: object acquisition, face detection, preprocessing, feature extraction, and classification. The mobile

application in this paper is represented by a DAG $G = (N, E)$, where N is the set of tasks and E is a set of edges that reflect dependency relationships between tasks. An example DAG illustrating this modelling approach is shown in Fig. 1. Each edge $e_{nn'}$ indicates that a task n must complete before the task n' begins. The term $a_{nn'}$ denotes the total data transferred from task n to task n', as an input for the latter. Tasks are generated by the mobile device, and the final execution results are returned to it. The entry node serves as the root of the spanning tree, while the exit node represents the task where the application terminates. For clarity, we introduce two virtual task nodes n_0 and n_{N+1} for a given DAG. n_0 marks the start of the application and n_{N+1} collects the execution results of all tasks. Thus, the set of tasks in the application is represented by $n \in N = \{0, 1, 2, ..., |N|, |N + 1|\}$.

Each task n for a certain mobile device m is modeled as a tuple $(i_{m,n}, c_{m,n}, \ddot{d}_{m,n})$, where: *i)* $i_{m,n}$ stands for the input data size of the computation task n of mobile device m, *ii)* $c_{m,n}$ reflects the amount of required computation resources to accomplish the task n of mobile device m, and *iii)* $\ddot{d}_{m,n}$ indicates the maximum tolerant delay for completing the task n of mobile device m, which means that the task n execution time should not exceed $\ddot{d}_{m,n}$.

Since there are dependencies between tasks, the precursor task may need to transmit its computation results to the successor task. Due to the uncertainty of the task execution location, all possible communication scenarios need to be analyzed. The communication delay per data unit from MEC server p to another MEC server p' is denoted by $d_{p,p'}$ (when $p = p'$, $d_{pp'} = 0$). The communication delay per data unit between the MEC server and the mobile device is denoted by $d_{p,m}$. The mobile device generates communication with the cloud server q through the base station hosting MEC server p, and the communication delay per data unit between the MEC server and the cloud server is denoted by $d_{p,q}$. Furthermore, we assume the system operates in fixed-length time slots $t \in \{1, 2, ...\tau\}$, where τ denotes the finite length of the time horizon. In each time slot, each mobile device generates one computation-intensive task and changes its location, where $(\ell_{m,n}(t), \varphi_{m,n}(t))$ are the longitude and latitude of mobile device m for task n in time slot t. Note that a task can arrive in a time slot only if its precursor task has completed execution.

B. Problem Formulation

Mobile device m has three execution options for task n: it can execute it locally on the mobile device, offload it to a MEC server, or offload it to a cloud server. Let $x^l_{m,n}(t) \in \{0, 1\}$, $y^e_{m,n,p}(t) \in \{0, 1\}$, and $z^c_{m,n,q}(t) \in \{0, 1\}$ be binary offloading variables at time slot t, representing the computation offloading strategies locally, to the MEC server, and to the cloud server, respectively, where $p \in P$ and $q \in Q$. Particularly, $x^l_{m,n}(t) = 1$ indicates that mobile device m chose to execute task n locally at time slot t. $y^e_{m,n,p}(t) = 1$ means that mobile device m chooses to offload task n to the pth MEC server in time slot t while $y^e_{m,n,p}(t) = 0$ means otherwise. Likewise, $z^c_{m,n,q} = 1$ means that mobile device m decides to offload task n to the qth remote cloud server in time slot t, and $z^c_{m,n,q} = 0$ means otherwise. We set $x^l_{m,n_0}(t) = 1$ and $x^l_{m,n_{N+1}}(t) = 1$ to ensure that tasks n_0 and n_{N+1} can only be executed on the mobile device. In each time slot t, we need to make a decision regarding the computation offloading of task n of mobile device m.

1) Computation Model: The computation task n of mobile device m can be executed either locally, using the computation resources of the mobile device, or on a remote server (MEC or cloud), through computation offloading.

Local Computing: If the mobile device m chooses to compute task n locally, the entire computation resource of this device will be utilized for computing the task. We denote f^l_m as the computation resource (in CPU cycle per second) of mobile device m. The local task completion delay in time slot t only includes the task computing delay, which can be expressed as:

$$T^l_{m,n}(t) = x^l_{m,n}(t)\frac{c_{m,n}}{f^l_m} \tag{1}$$

The energy consumption of unit computation resources per CPU cycle is $\zeta(f^l_m)^2$, where ζ represents the effective switched capacities depending on the chip architecture [3]. We denote the local energy consumption for computing task n of mobile device m in time slot t as $E^l_{m,n}(t)$ which can be defined as:

$$E^l_{m,n}(t) = x^l_{m,n}(t)c_{m,n}\zeta(f^l_m)^2 \tag{2}$$

Offloading to MEC Server: Several mobile devices may be associated with the same MEC server and offload their tasks simultaneously, with each mobile device assigned to a portion of the MEC server's computation resources. Let F^e_p denote the total computation resources of MEC p. In time slot t, if the mobile device m decides to offload its computation task n to the MEC server p, the computation resource allocation capacity of MEC server p to mobile device m for task n at time slot t is denoted by $f^e_{p,m,n}(t)$. After processing the task, the MEC server returns the results to mobile device m. We neglect the transmission delay and energy consumption for sending results back to the mobile device due to the small data size and typically higher downlink rates. The total processing time for task n consists of two components: the uplink transmission delay and the computation execution time on the MEC server, which is based on the allocated resources $f^e_{p,m,n}(t)$. The task completion delay for offloading task n of mobile device m to MEC server p in time slot t can be written as:

$$T^e_{m,n}(t) = y^e_{m,n,p}(t)\left(\frac{i_{m,n}}{R^e_{m,n,p}(t)} + \frac{c_{m,n}}{f^e_{p,m,n}(t)}\right) \tag{3}$$

where $R^e_{m,n,p}(t)$ is the wireless communication data rate between mobile devices m and MEC server p, which varies based on transmission power, interference, bandwidth, and the location of the mobile device. We assume mobile devices communicate with the nearest MEC server for computation offloading. Let ω_p represent the coverage radius of MEC server p. If the distance between mobile device m and MEC server p in time slot t is less than ω_p (i.e.,$r_{m,n,p}(t) < \omega_p$), the mobile device m can communicate with MEC server p. The maximum

achievable wireless communication data rate for transmitting task n in time slot t is given by:

$$R^e_{m,n,p}(t) = W^e log_2\left(1 + \frac{P_{m,n,p} h_{m,n,p} (r_{m,n,p}(t))^{-\alpha}}{\phi^2 + \mathcal{I}}\right) \quad (4)$$

where W^e denotes the channel bandwidth, $P_{m,n,p}$ is the transmission power of mobile device m to offload task n to MEC server p, $h_{m,n,p}$ is the channel gain, α is the path loss exponent, ϕ^2 is the noise power, and $\mathcal{I}$ represents the interference from other MEC servers.

The total energy consumption consists of the energy used to transmit the task n to MEC server p and the energy consumed by MEC server p to process the task. The total energy consumption for completing the task n of mobile device m in time slot t in the MEC server is:

$$E^e_{m,n}(t) = y^e_{m,n,p}(t)\left(\frac{P_{m,n,p} i_{m,n}}{R^e_{m,n,p}(t)} + c_{m,n} \cdot e_p\right) \quad (5)$$

where e_p is the energy consumed per CPU cycle by MEC server p.

Offloading to Cloud Server: A sub-task n of mobile device m can be offloaded to a cloud server for its execution. In this case, the nearest base station to the mobile device will relay the task to the cloud server. The total processing time of task n of mobile device m in the cloud server consists of three parts: *i)* the transmission time from mobile device m to the MEC server p, *ii)* the expected propagation delay for transmitting the task n from MEC server p to the cloud server q, and *iii)* the computation execution time on the cloud server. The task completion delay for offloading the task n from the mobile device m to the cloud server in time slot t is given by:

$$T^c_{m,n}(t) = z^c_{m,n,p}(t)\left(\frac{i_{m,n}}{R^e_{m,n,p}(t)} + E(T_p) + \frac{c_{m,n}}{f^c_{q,m,n}(t)}\right) \quad (6)$$

where $E(T_p)$ is the expected propagation delay between MEC server p and the cloud server q, and $f^c_{q,m,n}(t)$ is the computation resource capacity of the cloud server q assigned to task n of mobile device m in time slot t, with the total allocation not exceeding the cloud server's total capacity F^c_q. The energy consumption for completing task n of mobile device m in time slot t is:

$$E^c_{m,n}(t) = z^c_{m,n,p}(t)\left(\frac{P_{m,n,p} i_{m,n}}{R^e_{m,n,p}(t)} + \frac{P_{p,n,q} i_{m,n}}{b_{p,q}} + c_{m,n} \cdot e_q\right) \quad (7)$$

where $P_{p,n,q}$ is the transmission power from MEC server p to offload task n to cloud server q, $b_{p,q}$ is the bandwidth between MEC server and the corresponding remote cloud server, and e_q is the energy consumed per CPU cycle by cloud server q.

2) Data Transferring: Let $T^{tr}_{nn'}$ and $E^{tr}_{nn'}$ denote the time and energy consumed in data transfer between two dependent tasks n and n', respectively. As shown in Figure 1, a mobile application with five dependent tasks is illustrated, where each task receives data from its predecessor. For example, tasks n_1 and n_2 are dependent, and after completion of n_1, a data amount of a_1 is transferred from n_1 to task n_2. The transfer occurs in four modes: edge-to-edge, mobile-to-edge, edge-to-cloud, and mobile-to-cloud.

Transfer between MEC Servers: When both tasks are offloaded to the same MEC server, the data transfer time is zero. For tasks offloaded to different MEC servers, assuming that MEC servers are neighboring and directly connected, the communication delay in time slot t is given by:

$$T^{tr}_{p,p'}(t) = y^e_{m,n,p}(t) y^e_{m,n',p'}(t) d_{p,p'} a_{n,n'} \quad (8)$$

The energy consumption for this transfer is calculated as:

$$E^{tr}_{p,p'}(t) = T^{tr}_{p,p'}(t) P_{p,n,p'} \quad (9)$$

where $P_{p,n,p'}$ is the transmission power between the MEC server p and MEC server p' for the data transfer of task n.

Transfer between MEC Server and Mobile Device: When the task n is executed on mobile device m and n' on MEC server p, data must be transmitted between them, causing a communication delay of $x^l_{m,n}(t) y^e_{m,n',p}(t) d_{p,m} a_{nn'}$. Similarly, when the task n is executed on MEC server p and n' on the mobile device m, the delay between the MEC server p and the mobile device m is $x^l_{m,n'}(t) y^e_{m,n,p}(t) d_{p,m} a_{nn'}$. Therefore, the communication delay between the MEC server p and the mobile device m in time slot t is:

$$\begin{aligned} T^{tr}_{p,m}(t) = & x^l_{m,n}(t)\, y^e_{m,n',p}(t)\, d_{p,m}\, a_{n,n'} \\ & + x^l_{m,n'}(t)\, y^e_{m,n,p}(t)\, d_{p,m}\, a_{n,n'} \end{aligned} \quad (10)$$

The corresponding energy consumption for this data transfer is given by:

$$E^{tr}_{p,m}(t) = T^{tr}_{p,m}(t) P_{p,n,m} \quad (11)$$

where $P_{p,n,m}$ is transmission power of the MEC server p for transmitting task n to the mobile device m.

Transfer between MEC Server and the Cloud server: When n is executed on MEC server p and n' on cloud server q, data transfer between them incurs a communication delay of $y^e_{m,n,p}(t) z^c_{m,n',q}(t) d_{p,q} a_{n,n'}$. Similarly, in the reverse direction, the delay is $y^e_{m,n',p}(t) z^c_{m,n,q}(t) d_{p,q} a_{n,n'}$. Therefore, the communication delay between MEC server p and cloud server q in time slot t is expressed as:

$$\begin{aligned} T^{tr}_{p,q}(t) = & y^e_{m,n,p}(t) z^c_{m,n',q}(t) d_{p,q} a_{n,n'} + \\ & y^e_{m,n',p}(t) z^c_{m,n,q}(t) d_{p,q} a_{n,n'} \end{aligned} \quad (12)$$

The corresponding energy consumption for transmitting data between MEC server p and cloud server q is given by:

$$E^{tr}_{p,q}(t) = T^{tr}_{p,q}(t) P_{p,n,q} \quad (13)$$

Transfer between Mobile Device and the Cloud Server: When a task n is executed on mobile device m and n' on cloud server q, the data transfer between them incurs a communication delay of $x^l_{m,n}(t) z^c_{m,n',q}(t) d_{m,q} a_{n,n'}$. In the reverse direction, the delay is $x^l_{m,n'}(t) z^c_{m,n,q}(t) d_{m,q} a_{n,n'}$. Thus, the communication delay between the mobile device m and cloud server q in time slot t can be expressed as:

$$\begin{aligned} T^{tr}_{m,q}(t) = & x^l_{m,n}(t) z^c_{m,n',q}(t) d_{m,q} a_{nn'} + \\ & x^l_{m,n'}(t) z^c_{m,n,q}(t) d_{m,q} a_{nn'} \end{aligned} \quad (14)$$

where $d_{m,q} = d_{p,q} + d_{p,m}$ is the communication delay per unit of data transferred between the mobile device m and the cloud server q with data first transferred from the mobile device (cloud server) to a MEC server and then forwarded to the cloud server (mobile device). The energy consumption for data

transmission between mobile device m and cloud server q, given the communication delay $T^{tr}_{m,q}(t)$, is:

$$E^{tr}_{m,q}(t) = T^{tr}_{m,q}(t) P_{m,n,q} \tag{15}$$

where $P_{m,n,q} = P_{m,n,p} + P_{p,n,q}$ is the transmission power of the mobile device m for transmitting task n to cloud server q.

In summary, if data transmission is performed between two tasks n and n' with dependencies, the time and energy required for their transmission in time slot t are given by Eq. 16 and Eq. 17, respectively:

$$T^{tr}_{n,n'}(t) = T^{tr}_{p,p'}(t) + T^{tr}_{p,m}(t) + T_{m,q}(t) + T^{tr}_{p,q}(t) \tag{16}$$

$$E^{tr}_{n,n'}(t) = E^{tr}_{p,p'}(t) + E^{tr}_{p,m}(t) + E_{m,q}(t) + E^{tr}_{p,q}(t) \tag{17}$$

3) Migration Model: In a MEC environment, task migration is essential for maintaining service continuity as mobile devices move. Each time slot requires a decision on whether to migrate tasks between multiple servers to meet user demands [14]. We assume that the mobile device m migrates all offloaded tasks from the source node to the destination node. In this case, $y^e_{m,n,p}(t-1) \neq y^e_{m,n,p}(t)$ or $z^c_{m,n,q}(t-1) \neq z^c_{m,n,q}(t)$. The migration delay, defined as the time to transfer task n from node k to node k' in time slot t, is given by:

$$T^{mig}_{m,n}(t) = \frac{i_{m,n}}{b_{kk'}} \tag{18}$$

where $b_{kk'}$ represents the bandwidth used for the communication. If no migration occurs, indicated by $y^e_{m,n,p}(t-1) = y^e_{m,n,p}(t)$ or $z^c_{m,n,q}(t-1) = z^c_{m,n,q}(t)$, then $T^{mig}_{m,n}(t) = 0$.

The energy consumption for transferring the task n of mobile device m between source and destination servers in time slot t is expressed as:

$$E^{mig}_{m,n}(t) = j_k T^{mig}_{m,n}(t) \tag{19}$$

where j_k denotes the power consumption per unit time for data transmission at the source server k.

Our objective is to find an optimal offloading solution that minimizes energy consumption over all time slots, considering the computation tasks of all mobile devices. We can consider the following objective function:

$$E_{total} = \sum_{t=1}^{t=\tau} \sum_{p\in P, q\in Q} \sum_{m\in M} \sum_{n\in N} E^l_{m,n}(t) + E^e_{m,n}(t) + E^c_{m,n}(t) + E^{tr}_{n,n'}(t) + E^{mig}_{m,n}(t) \tag{20}$$

A feasible offloading solution should satisfy the following constraints:

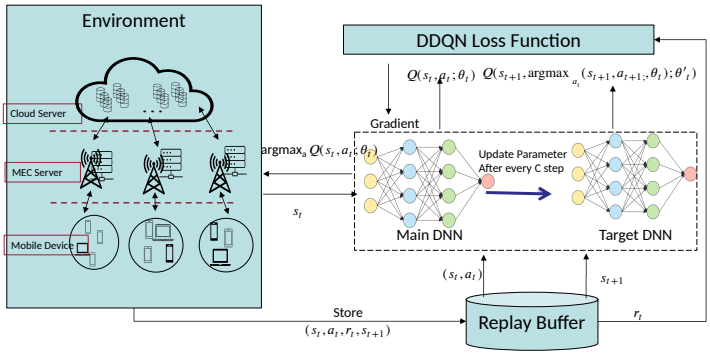

Fig. 2: Training process of DDQN.

$$\sum_{p\in P, q\in Q} T^l_{m,n}(t) + T^e_{m,n}(t) + T^c_{m,n}(t) + T^{tr}_{n,n'}(t) + T^{mig}_{m,n}(t) \leq \ddot{d}_{m,n}, \quad \forall n \in N, m \in M \tag{21}$$

$$x^l_{m,n}(t) + \sum_{p\in P} y^e_{m,n,p}(t) + \sum_{q\in Q} z^c_{m,n,q}(t) = 1, \forall n \in N, m \in M \tag{22}$$

$$x^l_{m,n}(t), y^e_{m,n,p}(t), z^c_{m,n,q}(t) \in \{0,1\} \tag{23}$$

$$\sum_{m\in M} \sum_{n\in N} z^c_{m,n,q}(t) f^c_{q,m,n}(t) \leq F^c_q, \forall q \in Q \tag{24}$$

$$\sum_{m\in M} \sum_{n\in N} y^e_{m,n,p}(t) f^e_{p,m,n}(t) \leq F^e_p, \forall p \in P \tag{25}$$

$$TR_{m,n} + T^l_{m,n}(t) + T^e_{m,n}(t) + T^c_{m,n}(t) + T^{mig}_{m,n}(t) \leq TR_{m,n'}, \quad \forall n \in N, m \in M \tag{26}$$

Constraint 21 guarantees that the total delay of the task cannot exceed the upper limit $\ddot{d}_{m,n}$. Constraints 22 and 23 ensure that each mobile device only chooses one way to compute its task, i.e., executing locally or offloading to a MEC server or offloading to a cloud server. Constraints 24 and 25 ensure that the sum of the computation resources of each server (MEC or cloud server) allocated to all tasks does not exceed the total amount of computation resources available on that server. Constraint 26 is a dependency constraint. For any edge $(e, e') \in E$ in the DAG, task n' cannot be started until all the associated precursor tasks have been executed, and the required data have been transferred to the server where task n' is offloaded. Here, $TR_{m,n}$ represents the time when task n of mobile device m is ready to be processed in time slot t. This means that the task n' can be executed only after task n has been executed.

Since the problem is a MINLP problem and involves dynamic factors such as the mobility of mobile devices and the changing computation capacity of servers over time, a traditional model-based approach may not be suitable for dynamic decision-making scenarios. Therefore, we leverage a DRL-based approach to address the optimization problem.

IV. DRL-Based Task Offloading Method

In this section, we incorporate a Q-learning-based method to minimize total energy consumption. We first define the

problem as an MDP. Then, we present a Double Deep Q-Network (DDQN), which leverages Deep Neural Networks (DNNs). A typical MDP is defined by a 3-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R})$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, and $\mathcal{R}$ is an immediate reward function for taking an action, which is defined as follows.

1) System State: The state $s_t \in \mathcal{S}$ at time slot t is composed of five parts: the input data sizes of tasks $i(t) = [i_{1,1}(t), ..., i_{|M|,|N|}(t)]$; the resources required to complete the tasks $c(t) = [c_{1,1}(t), ..., c_{|M|,|N|}(t)]$; the wireless data rates between mobile devices and MEC servers $R^e(t) = [R^e_{1,1}(t), ..., R^e_{|M|,|P|}(t)]$; the available computational resources of the MEC and cloud servers $F(t) = [F_1(t), ..., F_{|K|}(t)]$; and the dependency matrix $A(t) = [A_{n,n'}(t)]$, where each element $A_{n,n'}(t)$ represents the amount of data to be transferred from task n to task n'. If $A_{n,n'}(t) > 0$, task n must be completed before task n'; otherwise, $A_{n,n'}(t) = 0$.

2) System Action: The action vector consists of two main components: offloading decision-making and computation resource allocation. The offloading decision-making component, $\mathbf{x}(t) = [x_1(t), \ldots, x_N(t)]$, represents the offloading decisions for tasks, where each $x_i(t)$ indicates the index of the server to which task i is offloaded. This index ranges from 0 (indicating that the task is executed locally) to the total number of servers. The second component is computation resource allocation $f(t) = [f_1(t), \ldots, f_N(t)]$, with $f_i(t)$ denoting the computation resources assigned to task i at time slot t.

3) Reward Function: The reward function is associated with the optimization objective function which is minimizing total energy consumption, by giving a higher positive reward to actions that lead to a lower energy consumption. After an action a_t is taken, the system receives a reward $r(s_t, a_t)$, defined as:

$$r(s_t, a_t) = \kappa - \mathbb{E}\Bigg[\sum_{p\in P, q\in Q}\sum_{m\in M}\sum_{n\in N} E^l_{m,n}(t) + E^e_{m,n}(t) + E^c_{m,n}(t) + E^{tr}_{n,n'}(t) + E^{mig}_{m,n}(t)\Bigg], \tag{27}$$

where κ is a constant value that shifts the reward into a positive range, determined based on the maximum expected negative reward from the system.

$$r(s_t, a_t) = \kappa - \mathbb{E}\Bigg[\sum_{p\in P, q\in Q}\sum_{m\in M}\sum_{n\in N} E^l_{m,n}(t) + E^e_{m,n}(t) + E^c_{m,n}(t) + E^{tr}_{n,n'}(t) + E^{mig}_{m,n}(t)\Bigg], \tag{28}$$

Q-learning is an effective model-free RL algorithm that enables the agent to learn optimal behavior by observing the state s_t in the environment at time slot t, selecting an action a_t to execute according to the probability distribution or Q-table, transitioning to the next state s_{t+1}, and receiving a reward r_t. The Q-table and the current state are updated based on the observed reward and the estimated future rewards, which guide the agent toward selecting actions that maximize long-term rewards. The learning process is repeated until the algorithm converges to the optimal Q-function Q^*, and the optimal policy is obtained. While an optimal policy can be derived by updating Q-values in a Q-table, this approach becomes impractical due to the curse of dimensionality in large state-action spaces. Additionally, searching through a large Q-table can be time-consuming and memory-intensive. To overcome these challenges, we propose using DDQN. The structure of DDQN, as shown in Fig. 2, consists of two neural networks: the Main DNN, which identifies the best action, and the Target DNN, which evaluates the Q-value of the selected action, helping avoid overestimation of the action-value function. The DDQN algorithm utilizes two Q-functions, allowing each network to use the other's Q-value to update the next state. During the update process, DDQN first identifies the action with the maximum Q-value from the Q-network and then incorporates the chosen action with the maximum Q-value into the target network, thereby calculating the target Q-value, which is computed by:

$$y'_j = r_j + \gamma Q(s_{j+1}, \arg\max_{a_{j+1}} Q(s_{j+1}, a_{j+1}; \theta_j); \theta'_j) \tag{29}$$

j represents the index of the learning step within an episode.

Algorithm 1: Training Process of DDQN for Dependent Task Offloading Decision

Input: Experience replay buffer, main DNN with random weights θ, target DNN with weights $\theta' \leftarrow \theta$

for *each episode* **do**
  Obtain initial observation s_t;
  for *each step of the episode* **do**
    if *with probability* ϵ **then**
      Select a random action a_t from action set $\mathcal{A}$;
    else
      Select action $a_t \leftarrow \arg\max_a Q(s_t, a; \theta)$;
    Execute action a_t, observe reward r_t and next observation s_{t+1};
    Store transition (s_t, a_t, r_t, s_{t+1}) in replay buffer;
    Sample a minibatch of size $\mathcal{D}$ from the replay buffer;
    for *each transition* (s_j, a_j, r_j, s_{j+1}) *in the minibatch* **do**
      Compute the target Q-value as Eq. 29;
      Compute loss $\text{Loss}(\theta) \leftarrow \frac{1}{|\mathcal{D}|}\sum_j (y_j - Q(s_j, a_j; \theta))^2$;
      Update main DNN parameters θ by minimizing $\text{Loss}(\theta)$;
      Compute gradient $\nabla_\theta \text{Loss}(\theta)$;
      Perform a gradient descent step to update θ;
  Update target DNN weights $\theta' \leftarrow \theta$;

The process of the proposed DDQN algorithm is shown in Algorithm 1. First, the experience replay buffer and the weights of the DNN are initialized. Notably, the target DNN is initialized with the same weights as the main DNN, i.e., $\theta' = \theta$. The DNN is then trained to capture the correlation between each state-action pair (s, a) and its value function $Q(s, a)$. Initially, the offloading decisions for mobile devices are processed using a random policy for a sufficiently long time. The corresponding estimated $Q(s, a)$ values, along with state transition profiles, are stored in the experience replay buffer. The DNN is then trained with input state-action pairs (s, a) and their outcomes $Q(s, a)$. During each episode, the DDQN agent starts by obtaining the initial observation of the environment and processing it as the starting state s_t. The ϵ-greedy strategy is then used to select the execution action a_t. With probability ϵ, an action is randomly chosen from the action set $\mathcal{A}$. Otherwise, the action with the highest estimated

Q-value, derived from the main neural network using the state-action pair (s_t, a_t), is selected. The action a_t is executed, and the resulting reward r_t and next observation s_{t+1} are obtained from the environment. The transition memory tuple (s_t, a_t, r_t, s_{t+1}) is stored in the replay buffer to train the DNN parameters at each time step t. Throughout the training process, a minibatch of $\mathcal{D}$ samples is randomly selected by the experience replay mechanism in DDQN. The target network uses r_t and s_{t+1} to calculate the target Q-value using Eq. 29. The DDQN agent updates the parameters θ of the main DNN by minimizing Loss(θ). The gradient guiding the updates of θ is computed as $\left[\frac{\partial \text{Loss}(\theta)}{\partial \theta}\right]$. Stochastic gradient descent is then performed until the state-action Q-function converges to its optimal value.

TABLE I: Simulation Parameters

Parameter	Value	Description
$P_{m,n,p}$	100 mW	Transmission power of mobile device m for task n to MEC server p
ϕ^2	10^{-11} mW	Noise power
α	3	Path loss exponent
f_m^l	2 GHz	Computation resource of mobile device
F_p^e	10 GHz	Total computation resources of MEC server p
F_q^c	50 GHz	Total computation resources of cloud server q
W^e	100 MHz	Channel bandwidth between a mobile device and MEC server
$b_{kk'}$	10 Gbps	Bandwidth between servers
e_p	0.5 W/GHz	Energy consumed per CPU cycle by MEC server p
e_q	5 W/GHz	Energy consumed per CPU cycle by cloud server q
$P_{p,n,q}$	100 mW	Transmission power of the MEC server p to offload task n to the cloud server q
$c_{m,n}$	$U[0.5, 5]$ MHz	Number of CPU cycles required to process the task
$i_{m,n}$	$U[5, 50]$ MB	Size of task n from mobile device m
$d_{m,n}$	3 sec	Maximum allowable delay for task completion
T_p	50 ms	Propagation delay between MEC and cloud servers
$d_{p,p'}$	[0.45, 0.8] ms	Communication delay between MEC servers
$d_{p,m}$	[1, 10] ms	Communication delay between MEC server p and mobile device m
$a_{n,n'}$	$U[500$ kB, 2 MB]	Data size of dependency between two tasks
$h_{m,n,p}$	$127 + 20\log(L)$	Channel gain between mobile device m and MEC server p

V. Numerical Results

We consider a network environment with one cloud server and three MEC servers covering an area of $1000\,\text{m} \times 1000\,\text{m}$. Ten mobile devices are randomly scattered across this area, and their positions change over time, following a random walk model. The tasks for each mobile device are generated based on popular mainstream applications, such as face recognition (two tasks), pose recognition benchmarks (four tasks), and gesture recognition (two tasks) [15]. Table I summarizes the simulation parameters. The proposed algorithm was implemented in Python 3.10 and TensorFlow 2.17, using the OpenAI Gym toolkit [1], and deployed on the Graham cluster of the Digital Research Alliance of Canada [2].

[1]https://www.gymlibrary.dev/index.html

[2]https://docs.alliancecan.ca/wiki/Graham

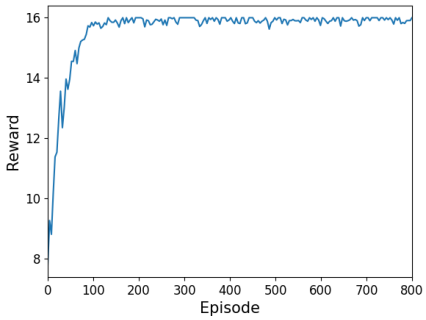

Fig. 3: Convergence of the proposed DDQN-based method

The number of hidden units per layer is set to 128. During the training process, we set the experience replay buffer size to 5000, the mini-batch size for sample transfer tuples to 64, and the learning rate to 0.001. The discount factor γ is set to 0.9, and the activation function used is ReLU. To verify the performance of our proposed algorithm, we introduce the following two benchmark policies:

- **Cloud-Only (CO):** In CO, all tasks are offloaded to the cloud server, and thus the resources of the MEC servers are not utilized.
- **Non-Collaboration (NC):** MEC servers do not cooperate with each other, and all tasks of a mobile device are randomly offloaded to a MEC server until the server's capacity is full. If a MEC server does not have enough capacity, the tasks will be offloaded to the cloud server.

Figure 3 shows the convergence behavior of the DDQN-based method in a system with six mobile devices. At the beginning, the agent follows a suboptimal policy that leads to high energy consumption, resulting in a lower cumulative reward. As training progresses, the agent learns to select actions that reduce energy use, causing a rapid increase in cumulative reward as energy efficiency improves. Around episode 200, the reward stabilizes, indicating that the agent has converged to an energy-efficient offloading strategy. The final stabilized reward represents the minimum energy consumption achievable under the learned policy, highlighting the effectiveness of the DDQN-based approach in optimizing energy usage.

Fig. 4 illustrates the performance comparison of the DDQN, NC, and CO methods in terms of total energy consumption as the number of mobile devices increases from 6 to 10. As expected, with the increase in the number of mobile devices, more computational tasks are generated, leading to a rise in energy consumption across all methods. The energy consumption of the CO method is consistently higher compared to the others, as it consumes more energy due to communication. The NC method consumes more energy than the proposed DDQN, as it randomly offloads tasks to servers without considering the energy consumption associated with computation, transmission, and migration. The NC method performs better than CO but still exhibits higher energy consumption compared to the proposed DDQN algorithm. The DDQN-based approach

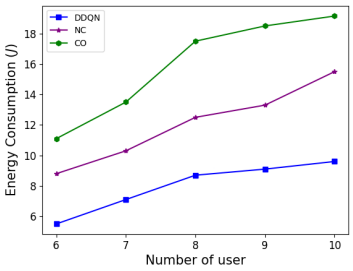

Fig. 4: Energy consumption versus the number of mobile devices.

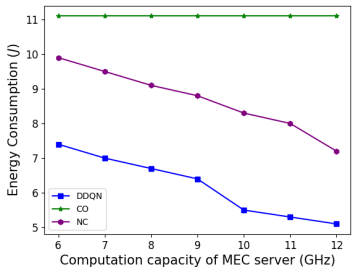

Fig. 5: Energy consumption versus the computing resources of the MEC server.

significantly outperforms both NC and CO methods. This improvement is achieved as the DDQN algorithm effectively integrates computation offloading, task migration, and edge collaboration into its decision-making process. The proposed DDQN method reduces energy consumption by approximately 36% compared to the NC method and 49% compared to the CO method.

Fig. 5 presents a performance comparison of the DDQN, NC, and CO methods under varying computation capacities of MEC servers, which range from 6 to 12 GHz, while the number of mobile devices is fixed at 6. The CO method exhibits the highest energy consumption throughout, as it does not utilize MEC server resources. As the computational capacity of the MEC server increases, the energy consumption of both the DDQN and NC methods decreases steadily. This reduction happens because more tasks can be processed closer to the mobile devices by the MEC servers, reducing the need for high-energy consumption associated with distant task migration and processing. The DDQN-based method consistently achieves the lowest energy consumption compared to the other approaches. On average, the DDQN algorithm reduces energy consumption by approximately 28% compared to NC.

VI. Conclusion

In this paper, we investigated energy-efficient task offloading in MEC environments, particularly in the context of dynamic end-device mobility and task dependencies. We mathematically modeled energy consumption by considering computation, data transmission, and task migration. By framing the problem as an MDP and leveraging a DRL approach, we developed a mechanism that dynamically adapts offloading decisions based on real-time changes in the network environment. Simulation results demonstrated the efficiency and effectiveness of the proposed algorithm. For future work, we plan to consider the mobility of MEC servers and the potential incorporation of a federated learning approach.

Acknowledgment

This work was supported by the CHIST-ERA ECOMOME project, through the Fonds de Recherche du Quebec – Nature et Technologies (FRQNT).

References

[1] I. Sitton-Candanedo, R. S. Alonso, O. Garcia, A. B. Gil, and S. Rodriguez-Gonzalez, "A review on edge computing in smart energy by means of a systematic mapping study," *Electronics*, vol. 9, no. 1, 2020.

[2] T. Z. H. Ernest and A. S. Madhukumar, "Computation offloading in MEC-enabled IoV networks: Average energy efficiency analysis and learning-based maximization," *IEEE Transactions on Mobile Computing*, pp. 1–14, 2023.

[3] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint computation offloading and user association in multi-task mobile edge computing," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 12, pp. 12 313–12 325, 2018.

[4] H. Hu, Q. Wang, R. Q. Hu, and H. Zhu, "Mobility-aware offloading and resource allocation in a MEC-enabled IoT network with energy harvesting," *IEEE Internet of Things Journal*, vol. 8, no. 24, pp. 17 541–17 556, 2021.

[5] L. Chen, J. Wu, J. Zhang, H.-N. Dai, X. Long, and M. Yao, "Dependency-aware computation offloading for mobile edge computing with edge-cloud cooperation," *IEEE Transactions on Cloud Computing*, vol. 10, no. 4, pp. 2451–2468, 2022.

[6] J. Zhang, J. Chen, X. Bao, C. Liu, P. Yuan, X. Zhang, and S. Wang, "Dependent task offloading mechanism for cloud-edge-device collaboration," *Journal of Network and Computer Applications*, vol. 216, p. 103656, 2023.

[7] S. N. Afrasiabi, A. Ebrahimzadeh, C. Mouradian, S. Malektaji, and R. H. Glitho, "Reinforcement learning-based optimization framework for application component migration in nfv cloud-fog environments," *IEEE Transactions on Network and Service Management*, vol. 20, no. 2, pp. 1866–1883, 2023.

[8] "A comprehensive survey on reinforcement-learning-based computation offloading techniques in edge computing systems," *Journal of Network and Computer Applications*, vol. 216, p. 103669, 2023.

[9] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[10] H. Zhou, Z. Zhang, Y. Wu, M. Dong, and V. C. M. Leung, "Energy efficient joint computation offloading and service caching for mobile edge computing: A deep reinforcement learning approach," *IEEE Transactions on Green Communications and Networking*, vol. 7, no. 2, pp. 950–961, 2023.

[11] H. Zhou, K. Jiang, X. Liu, X. Li, and V. C. M. Leung, "Deep reinforcement learning for energy-efficient computation offloading in mobile-edge computing," *IEEE Internet of Things Journal*, vol. 9, no. 2, pp. 1517–1530, 2022.

[12] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, "Energy-efficient offloading for mobile edge computing in 5G heterogeneous networks," *IEEE Access*, vol. 4, pp. 5896–5907, 2016.

[13] Z. Ali, S. Khaf, Z. H. Abbas, G. Abbas, F. Muhammad, and S. Kim, "A deep learning approach for mobility-aware and energy-efficient resource allocation in MEC," *IEEE Access*, vol. 8, pp. 179 530–179 546, 2020.

[14] S. N. Afrasiabi, A. Ebrahimzadeh, N. Promwongsa, C. Mouradian, W. Li, A. Recse, R. Szabo, and R. H. Glitho, "Cost-efficient cluster migration of vnfs for service function chain embedding," *IEEE Transactions on Network and Service Management*, vol. 21, no. 1, pp. 979–993, 2024.

[15] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: enabling interactive perception applications on mobile devices," ser. ACM MobiSys, 2011, p. 43–56.