Abstract

In the realm of mobile edge computing (MEC), efficient computation task offloading plays a pivotal role in ensuring a seamless quality of experience (QoE) for users. Maintaining a high QoE is paramount in today's interconnected world, where users demand reliable services. This challenge stands as one of the most primary key factors contributing to handling dynamic and uncertain mobile environment. In this study, we delve into computation offloading in MEC systems, where strict task processing deadlines and energy constraints can adversely affect the system performance. We formulate the computation task offloading problem as a Markov decision process (MDP) to maximize the long-term QoE of each user individually. We propose a distributed QoE-oriented computation offloading (QECO) algorithm based on deep reinforcement learning (DRL) that empowers mobile devices to make their offloading decisions without requiring knowledge of decisions made by other devices. Through numerical studies, we evaluate the performance of QECO. Simulation results validate that QECO efficiently exploits the computational resources of edge nodes. Consequently, it can complete 14% more tasks and reduce task delay and energy consumption by 9% and 6%, respectively. These together contribute to a significant improvement of at least 37% in average QoE compared to an existing algorithm.

Index Terms

Mobile edge computing, computation task offloading, quality of experience, deep reinforcement learning.

I. Introduction

Mobile edge computing (MEC) [1] has emerged as a promising technological solution to overcome the challenges faced by mobile devices (MDs) when performing high computational tasks, such as real-time data processing and artificial intelligence applications [2] [3]. In spite of the MDs' technological advancements, their limited computing power and battery may lead to task drops, processing delays, and an overall poor user experience. By offloading intensive tasks to nearby edge nodes (ENs), MEC effectively empowers computation capability and reduces the delay and energy consumption. This improvement enhances the users' QoE, especially for time-sensitive computation tasks [4] [5].

Efficient task offloading in MEC is a complex optimization challenge due to the dynamic nature of the network and the variety of MDs and servers involved [6] [7]. In particular, determining the optimal offloading strategy, scheduling the tasks, and selecting the most suitable EN for task offloading are the main challenges that demand careful consideration. Furthermore, the uncertain requirements and sensitive latency properties of computation tasks pose nontrivial challenges that can significantly impact the computation offloading performance in MEC systems with limited resources.

A. Related Work

To cope with the dynamic nature of the network, recent research has proposed several task offloading algorithms using machine learning methods. In particular, deep reinforcement learning (DRL) hold promises to determine optimal decision-making policies by capturing the dynamics of environments and learning strategies for accomplishing long-term objectives [8]. DRL can effectively tackle the challenges of MEC arising from the ever-changing nature of networks, MDs, and servers' heterogeneity. This ultimately improves the MD users' QoE. In [9], Huang *et al.* focused on a wireless-powered MEC. They proposed a

DRL-based approach, capable of attaining near-optimal decisions. This is achieved by selectively considering a compact subset of candidate actions in each iteration. In [10], the authors proposed an offloading algorithm using deep Q-learning for wireless-powered Internet of Things (IoT) devices in MEC systems. This algorithm aims to minimize the task drop rate while the devices solely rely on harvested energy for operation. In [11], Zhao *et al.* proposed a computation offloading algorithm based on DRL, which addresses the competition for wireless channels to optimize long-term downlink utility. In this approach, each MD requires quality-of-service information from other MDs. Tang *et al.* in [12] investigated the task offloading problem for indivisible and deadline-constrained computational tasks in MEC systems. The authors proposed a distributed DRL-based offloading algorithm designed to handle uncertain workload dynamics at the ENs. Sun *et al.* in [13] explored both computation offloading and service caching problems in MEC. They formulated an optimization problem that aims to minimize the long-term average service delay. They then proposed a hierarchical DRL framework, which effectively handles both problems under heterogeneous resources. Dai *et al.* in [14] introduced the integration of action refinement into DRL and designed an algorithm to concurrently optimize resource allocation and computation offloading. In [15], Huang *et al.* proposed a DRL-based method based on a partially observable MDP, which guarantees the deadlines of real-time tasks while minimizing the total energy consumption of MDs. Liu *et al.* in [16] investigated a two-timescale computing offloading and resource allocation problem and proposed a resource coordination algorithm based on multi-agent DRL, which can generate interactive information along with resource decisions. Zhou *et al.* in [17] used an MDP to study MEC and modeled the interactions of the environment. They proposed a Q-learning approach to achieve optimal resource allocation strategies and computation offloading. In [18], Gao *et al.* introduced an attention-based multi-agent algorithm designed for decentralized computation offloading. This algorithm effectively tackles the challenges of dynamic resource allocation in large-scale heterogeneous networks. Gong *et al.* in [19] proposed a DRL-based network structure in the industrial IoT systems to jointly optimize task offloading and resource allocation in order to achieve lower energy consumption and decreased task delay. Liao *et al.* in [20] introduced a double reinforcement learning algorithm for performing online computation offloading in MEC. This algorithm optimizes transmission power and scheduling of CPU frequency when minimizing both task computation delay and energy consumption.

B. Motivation and Contributions

Although DRL-based methods have demonstrated their effectiveness in handling network dynamics, task offloading still encounters several challenges that require further attention. QoE is a time-varying performance measure that reflects user satisfaction and is not affected only by delay, as assumed in [9]– [13], but also by energy consumption. Albeit some existing works such as [14]– [20], have investigated the trade-off between delay and energy consumption, they fail to properly address the user demands and fulfill QoE requirements. A more comprehensive approach is required to address the dynamic requirements of individual users in real-time scenarios with multiple MDs and ENs. In contrast to the aforementioned works [9]– [20], we propose a DRL-based distributed algorithm that provides users with an appropriate balance

among QoE factors based on their demands. We also explore a more realistic MEC scenario involving delay-sensitive tasks with processing deadlines, posing a more intricate challenge.

In this study, we delve into the computation task offloading problem in MEC systems, where strict task processing deadlines and energy constraints can adversely affect the system performance. We propose a distributed QoE-oriented computation offloading (QECO) algorithm that leverages DRL to efficiently handle task offloading in uncertain loads at ENs. This algorithm empowers MDs to make offloading decisions utilizing only locally observed information, such as task size, queue details, battery status, and historical workloads at the ENs. By adopting the appropriate policy based on each MD's specific requirements at any given time, the QECO algorithm significantly improves the QoE for individual users.

Our main contributions are summarized as follows:

- *Task Offloading Problem in the MEC System:* We formulate the task offloading problem as an MDP for time-sensitive tasks. This approach takes into account the dynamic nature of workloads at the ENs and concentrates on providing high performance in the MEC system while maximizing the long-term QoE.
- *DRL-based Offloading Algorithm:* To address the problem of long-term QoE maximization, we focus on task completion, task delay, and energy consumption to quantify the MDs' QoE. We propose QECO algorithm based on DRL that empowers each MD to make offloading decisions independently, without prior knowledge of the other MDs' tasks and offloading models. With a focus on the MD's battery level, our approach leverages deep Q-network (DQN) [21] and long short-term memory (LSTM) [22] to prioritize and strike an appropriate balance between QoE factors. We also analyze the training convergence and complexity of the proposed algorithm.
- *Performance Evaluation:* We conduct comprehensive experiments to evaluate the QECO's performance as well as its training convergence under different computation workloads. The results demonstrate that our algorithm quickly converges and effectively utilizes the processing capabilities of MDs and ENs, resulting in substantial improvement of at least 37% in average QoE. This advantage is achieved through a 14% increase in the number of completed tasks, along with 9% and 6% reductions in task delay and energy consumption, respectively, when compared to the potential game-based offloading algorithm (PGOA) [23] and several benchmark methods.

The structure of this paper is as follows. Section II presents the system model, followed by the problem formulation in Section III. In Section IV, we present the algorithm, while Section V provides an evaluation of its performance. Finally, we conclude in Section VI.

II. System Model

We investigate a MEC system consisting of a set of MDs denoted by $\mathcal{I} = \{1, 2, ..., I\}$, along with a set of ENs denoted by $\mathcal{J} = \{1, 2, ..., J\}$, where I and J represent the number of MDs and ENs, respectively. We regard time as a specific episode containing a series of T time slots denoted by $\mathcal{T} = \{1, 2, \ldots, T\}$, each representing a duration of τ seconds. As shown in Fig. 8, we consider two separate queues for each MD to organize tasks for local processing or dispatching to ENs, operating in a first-in-first-out (FIFO) manner. The MD's scheduler is responsible for assigning newly arrived tasks to each of the queues at the

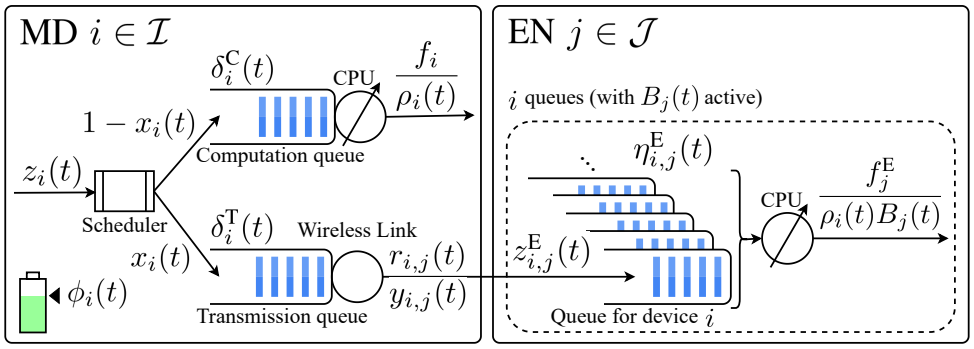


Fig. 1: An illustration of MD $i \in \mathcal{I}$ and EN $j \in \mathcal{J}$ in the MEC system.

beginning of the time slot. On the other hand, we assume that each EN $j \in \mathcal{J}$ consists of I FIFO queues, where each queue corresponds to an MD $i \in \mathcal{I}$. When each task arrives at an EN, it is enqueued in the corresponding MD's queue.

We define $z_i(t)$ as the index assigned to the computation task arriving at MD $i \in \mathcal{I}$ in time slot $t \in \mathcal{T}$. Let $\lambda_i(t)$ denote the size of this task in bits. The size of task $z_i(t)$ is selected from a discrete set $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_\theta\}$, where θ represents the number of these values. Hence, $\lambda_i(t) \in \Lambda \cup \{0\}$ to consider the case that no task has arrived. We also denote the task's processing density as $\rho_i(t)$ that indicates the number of CPU cycles required to complete the execution of a unit of the task. Furthermore, we denote the deadline of this task by $\Delta_i(t)$ which is the number of time slots that the task must be completed to avoid being dropped.

We define two binary variables, $x_i(t)$ and $y_{i,j}(t)$ for $i \in \mathcal{I}$ and $j \in \mathcal{J}$ to determine the offloading decision and offloading target, respectively. Specifically, $x_i(t)$ indicates whether task $z_i(t)$ is assigned to the computation queue ($x_i(t) = 0$) or to the transmission queue ($x_i(t) = 1$), and $y_{i,j}(t)$ indicates whether task $z_i(t)$ is offloaded to EN $j \in \mathcal{J}$. If the task is dispatched to EN j, we set $y_{i,j}(t) = 1$; otherwise, $y_{i,j}(t) = 0$.

A. *Communication Model*

We consider that the tasks in the transmission queue are dispatched to the appropriate ENs via the MD wireless interface. We denote the transmission rate of MD i's interface when communicating with EN $j \in \mathcal{J}$ in time t as $r_{i,j}(t)$. In time slot $t \in \mathcal{T}$, if task $z_i(t)$ is assigned to the transmission queue for computation offloading, we define $l_i^{\mathrm{T}}(t) \in \mathcal{T}$ to represent the time slot when the task is either dispatched to the EN or dropped. We also define $\delta_i^{\mathrm{T}}(t)$ as the number of time slots that task $z_i(t)$ should wait in the queue before transmission. It should be noted that MD i computes the value of $\delta_i^{\mathrm{T}}(t)$ before making a decision. The value of $\delta_i^{\mathrm{T}}(t)$ is computed as follows:

$$\delta_i^{\mathrm{T}}(t) = \left[\max_{t' \in \{0,1,\ldots,t-1\}} l_i^{\mathrm{T}}(t') - t + 1 \right]^+, \tag{1}$$

where $[\cdot]^+ = \max(0, \cdot)$ and $l_i^{\mathrm{T}}(0) = 0$ for the simplicity of presentation. Note that the value of $\delta_i^{\mathrm{T}}(t)$ only depends on $l_i^{\mathrm{T}}(t)$ for $t' < t$. If MD $i \in \mathcal{I}$ schedules task $z_i(t)$ for dispatching in time slot $t \in \mathcal{T}$, then it

will either be dispatched or dropped in time slot $l_i^{\mathrm{T}}(t)$, which is

$$l_i^{\mathrm{T}}(t) = \min\Big\{t + \delta_i^{\mathrm{T}}(t) + \lceil D_i^{\mathrm{T}}(t)\rceil - 1, t + \Delta_i(t) - 1\Big\}, \tag{2}$$

where $D_i^{\mathrm{T}}(t)$ refers to the number of time slots required for the transmission of task $z_i(t)$ from MD $i \in \mathcal{I}$ to EN $j \in \mathcal{J}$. We have

$$D_i^{\mathrm{T}}(t) = \sum_{\mathcal{J}} y_{i,j}(t)\frac{\lambda_i(t)}{r_{i,j}(t)\tau}. \tag{3}$$

Let $E_i^{\mathrm{T}}(t)$ denote the energy consumption of the transmission from MD $i \in \mathcal{I}$ to EN $j \in \mathcal{J}$. We have

$$E_i^{\mathrm{T}}(t) = D_i^{\mathrm{T}}(t)p_i^{\mathrm{T}}(t)\tau, \tag{4}$$

where $p_i^{\mathrm{T}}(t)$ represents the power consumption of the communication link of MD $i \in \mathcal{I}$ in time slot t.

B. Computation Model

The computation tasks can be executed either locally on the MD or on the EN. In this subsection, we provide a detailed explanation of these two cases.

1) Local Execution: We model the local execution by a queuing system consisting the computation queue and the MD processor. Let f_i denote the MD i's processing power (in cycle per second). When task $z_i(t)$ is assigned to the computation queue at the beginning of time slot $t \in \mathcal{T}$, we define $l_i^{\mathrm{C}}(t) \in \mathcal{T}$ as the time slot during which task $z_i(t)$ will either be processed or dropped. If the computation queue is empty, $l_i^{\mathrm{C}}(t) = 0$. Let $\delta_i^{\mathrm{C}}(t)$ denote the number of remaining time slots before processing task $z_i(t)$ in the computation queue. We have:

$$\delta_i^{\mathrm{C}}(t) = \left[\max_{t' \in \{0,1,\ldots,t-1\}} l_i^{\mathrm{C}}(t') - t + 1\right]^+. \tag{5}$$

In the equation above, the term $\max_{t' \in \{0,1,\ldots,t-1\}} l_i^{\mathrm{C}}(t')$ denotes the time slot at which each existing task in the computation queue, which arrived before time slot t, is either processed or dropped. Consequently, $\delta_i^{\mathrm{C}}(t)$ denotes the number of time slots that task $z_i(t)$ should wait before being processed. We denote the time slot in which task $z_i(t)$ will be completely processed by $l_i^{\mathrm{C}}(t)$ if it is assigned to the computation queue for local processing in time slot t. We have

$$l_i^{\mathrm{C}}(t) = \min\Big\{t + \delta_i^{\mathrm{C}}(t) + \lceil D_i^{\mathrm{C}}(t)\rceil - 1, t + \Delta_i(t) - 1\Big\}. \tag{6}$$

The task $z_i(t)$ will be immediately dropped if its processing is not completed by the end of the time slot $t + \Delta_i(t) - 1$. In addition, we introduce $D_i^{\mathrm{C}}(t)$ as the number of time slots required to complete the processing of task $z_i(t)$ on MD $i \in \mathcal{I}$. It is given by:

$$D_i^{\mathrm{C}}(t) = \frac{\lambda_i(t)}{f_i\tau/\rho_i(t)}. \tag{7}$$

To compute the MD's energy consumption in the time slot $t \in \mathcal{T}$, we define $E_i^{\mathrm{L}}(t)$ as:

$$E_i^{\mathrm{L}}(t) = D_i^{\mathrm{C}}(t) p_i^{\mathrm{C}} \tau, \tag{8}$$

where $p_i^{\mathrm{C}} = 10^{-27}(f_i)^3$ represents the energy consumption of MD i's CPU frequency [24].

2) Edge Execution: We model the edge execution by the queues associated with MDs deployed at ENs. If computation task $z_i(t')$ is dispatched to EN j in time $t' < t$, we let $z_{i,j}^{\mathrm{E}}(t)$ and $\lambda_{i,j}^{\mathrm{E}}(t)$ (in bits) denote the unique index of the task and the size of the task in the i^{th} queue at EN j. We define $\eta_{i,j}^{\mathrm{E}}(t)$ (in bits) as the length of this queue at the end of time slot $t \in \mathcal{T}$. We refer to a queue as an active queue in a certain time slot if it is not empty. That being said, if at least one task is already in the queue from previous time slots or there is a task arriving at the queue, that queue is active. We define $\mathcal{B}_j(t)$ to denote the set of active queues at EN j in time slot t.

$$\mathcal{B}_j(t) = \left\{ i \;\middle|\; i \in \mathcal{I}, \lambda_{i,j}^{\mathrm{E}}(t) > 0 \;\text{ or }\; \eta_{i,j}^{\mathrm{E}}(t-1) > 0 \right\}. \tag{9}$$

We introduce $b_j(t) \triangleq |\mathcal{B}_j(t)|$ that represents the number of active queues in EN $j \in \mathcal{J}$ in time slot $t \in \mathcal{T}$. In each time slot $t \in \mathcal{T}$, the EN's processing power is divided among its active queues using a generalized processor sharing method [25]. Let variable f_j^{E} (in cycles per second) represent the computational capacity of EN j. Therefore, EN j can allocate computational capacity of $f_j^{\mathrm{E}}/(\rho_i(t) b_j(t))$ to each MD $i \in \mathcal{B}_j(t)$ during time slot t. To calculate the length of the computation queue for MD $i \in \mathcal{I}$ in EN $j \in \mathcal{J}$, we define $\omega_{i,j}(t)$ (in bits) to represent the number of bits from dropped tasks in that queue at the end of time slot $t \in \mathcal{T}$. The backlog of the queue, referred to as $\eta_{i,j}^{\mathrm{E}}(t)$ is given by:

$$\eta_{i,j}^{\mathrm{E}}(t) = \left[\eta_{i,j}^{\mathrm{E}}(t-1) + \lambda_{i,j}^{\mathrm{E}}(t) - \frac{f_j^{\mathrm{E}} \tau}{\rho_i(t) b_j(t)} - \omega_{i,j}(t) \right]^+. \tag{10}$$

We also define $l_{i,j}^{\mathrm{E}}(t) \in \mathcal{T}$ as the time slot during which the offloaded task $z_{i,j}^{\mathrm{E}}(t)$ is either processed or dropped by EN j. Given the uncertain workload ahead at EN j, neither MD i nor EN j has information about $l_{i,j}^{\mathrm{E}}(t)$ until the corresponding task $z_{i,j}^{\mathrm{E}}(t)$ is either processed or dropped. Let $\hat{l}_{i,j}^{\mathrm{E}}(t)$ represent the time slot at which the execution of task $z_{i,j}^{\mathrm{E}}(t)$ starts. In mathematical terms, for $i \in \mathcal{I}$, $j \in \mathcal{J}$, and $t \in \mathcal{T}$, we have:

$$\hat{l}_{i,j}^{\mathrm{E}}(t) = \max\{t, \max_{t' \in \{0,1,\ldots,t-1\}} l_{i,j}^{\mathrm{E}}(t') + 1\}, \tag{11}$$

where $l_{i,j}^{\mathrm{E}}(0) = 0$. Indeed, the initial processing time slot of task $z_{i,j}^{\mathrm{E}}(t)$ at EN should not precede the time slot when the task was enqueued or when the previously arrived tasks were processed or dropped.

Therefore, $l^{\mathrm{E}}_{i,j}(t)$ is the time slot that satisfies the following constraints.

$$\sum_{t'=\hat{l}^{\mathrm{E}}_{i,j}(t)}^{l^{\mathrm{E}}_{i,j}(t)} \frac{f^{\mathrm{E}}_j \tau}{\rho_i(t) b_j(t')} \mathbb{1}(i \in \mathcal{B}_j(t')) \geq \lambda^{\mathrm{E}}_{i,j}(t), \tag{12}$$

$$\sum_{t'=\hat{l}^{\mathrm{E}}_{i,j}(t)}^{l^{\mathrm{E}}_{i,j}(t)-1} \frac{f^{\mathrm{E}}_j \tau}{\rho_i(t) b_j(t')} \mathbb{1}(i \in \mathcal{B}_j(t')) < \lambda^{\mathrm{E}}_{i,j}(t), \tag{13}$$

where $\mathbb{1}(z \in \mathbb{Z})$ is the indicator function. In particular, the total processing capacity that EN j allocates to MD i from the time slot $\hat{l}^{\mathrm{E}}_{i,j}(t)$ to the time slot $l^{\mathrm{E}}_{i,j}(t)$ should exceed the size of task $z^{\mathrm{E}}_{i,j}(t)$. Conversely, the total allocated processing capacity from the time slot $l^{\mathrm{E}}_{i,j}(t)$ to the time slot $l^{\mathrm{E}}_{i,j}(t) - 1$ should be less than the task's size.

Additionally, we define $D^{\mathrm{E}}_{i,j}(t)$ to represent the quantity of processing time slots allocated to task $z^{\mathrm{E}}_{i,j}(t)$ when executed at EN j. This value is given by:

$$D^{\mathrm{E}}_{i,j}(t) = \frac{\lambda^{\mathrm{E}}_{i,j}(t)\rho_i(t)}{f^{\mathrm{E}}_j \tau / b_j(t)}. \tag{14}$$

We also define $E^{\mathrm{E}}_{i,j}(t)$ as the energy consumption of processing at EN j in time slot t by MD i. This can be calculated as:

$$E^{\mathrm{E}}_{i,j}(t) = \frac{D^{\mathrm{E}}_{i,j}(t) p^{\mathrm{E}}_j \tau}{b_j(t)}, \tag{15}$$

where p^{E}_j is a constant value which denotes the energy consumption of the EN j's processor when operating at full capacity.

In addition to the energy consumed by EN j for task processing, we also take into account the energy consumed by the MD i's user interface in the standby state while waiting for task completion at the EN j. We define $E^{\mathrm{I}}_{i,j}(t)$ as the energy consumption associated with the user interface of MD $i \in \mathcal{I}$, which is given by

$$E^{\mathrm{I}}_i(t) = D^{\mathrm{E}}_{i,j}(t) p^{\mathrm{I}}_i \tau, \tag{16}$$

where p^{I}_i is the standby energy consumption of MD $i \in \mathcal{I}$.

$$E^{\mathrm{O}}_i(t) = E^{\mathrm{T}}_i(t) + \sum_{\mathcal{J}} E^{\mathrm{E}}_{i,j}(t) + E^{\mathrm{I}}_i(t). \tag{17}$$

III. Task Offloading problem Formulation

Based on the introduced system model, we present the computation task offloading problem in this section. Our primary goal is to enhance each MD's QoE individually by taking the dynamic demands of MDs into account. To achieve this, we approach the optimization problem as an MDP, aiming to maximize the MD's QoE by striking a balance among key QoE factors, including task completion, task delay, and energy consumption. To prioritize QoE factors, we utilize the MD's battery level, which plays a crucial role

in decision-making. Specifically, when an MD observes its state (e.g. task size, queue details, and battery level) and encounters a newly arrived task, it selects an appropriate action for that task. The selected action, based on the observed state, will result in enhanced QoE. Each MD strives to maximize its long-term QoE by optimizing the policy mapping from states to actions. In what follows, we first present the state space, action space, and QoE function, respectively. We then formulate the QoE maximization problem for each MD.

A. State Space

A state in our MDP represents a conceptual space that comprehensively describes the state of an MD facing the environment. We represent the MD i's state in time slot t as vector $\boldsymbol{s}_i(t)$ that includes the newly arrived task size, the queues information, the MD's battery level, and the workload history at the ENs. The MD observers this vector at the beginning of each time slot. The vector $\boldsymbol{s}_i(t)$ is defined as follows:

$$\boldsymbol{s}_i(t) = \Big(\lambda_i(t), \delta_i^{\mathrm{C}}(t), \delta_i^{\mathrm{T}}(t), \boldsymbol{\eta}_i^{\mathrm{E}}(t-1), \phi_i(t), \mathcal{H}(t)\Big), \tag{18}$$

where vector $\boldsymbol{\eta}_i^{\mathrm{E}}(t-1) = (\eta_{i,j}^{\mathrm{E}}(t-1))_{j\in\mathcal{J}}$ represents the queues length of MD i in ENs at the previous time slot and is computed by the MD according to (10). Let $\phi_i(t)$ denote the battery level of MD i in time slot t. Considering the power modes of a real mobile device, $\phi_i(t)$ is derived from the discrete set $\Phi = \{\phi_1, \phi_2, \phi_3\}$, corresponding to ultra power-saving, power-saving, and performance modes, respectively.

In addition, to predict future EN workloads, we define the matrix $\mathcal{H}(t)$ as historical data, indicating the number of active queues for all ENs. This data is recorded over T^{s} time slots, ranging from $t-T^{\mathrm{s}}$ to $t-1$, in $T^{\mathrm{s}} \times J$ matrix. For EN j workload history at i^{th} time slot from $T^{\mathrm{s}} - t$, we define $h_{i,j}(t)$ as:

$$h_{i,j}(t) = b_j(t - T^{\mathrm{s}} + i - 1). \tag{19}$$

EN $j \in \mathcal{J}$ broadcasts $b_j(t)$ at the end of each time slot.

We define vector $\mathcal{S}$ as the discrete and finite state space for each MD. The size of the set $\mathcal{S}$ is given by $\Lambda \times T^2 \times \mathcal{U} \times 3 \times I^{T^{\mathrm{s}}\times J}$, where $\mathcal{U}$ is the set of available queue length values at an EN over T time slots.

B. Action Space

The action space represents the agent's behavior and the decisions. In this context, we define $\boldsymbol{a}_i(t)$ to denote the action taken by MD $i \in \mathcal{I}$ in time slot $t \in \mathcal{T}$. These actions involve two decisions, (a) Offloading decision to determine whether or not to offload the task, and (b) Offloading target to determine the EN to send the offloaded tasks. Thus, the action of MD i in time slot t can be concisely expressed as the following action tuple:

$$\boldsymbol{a}_i(t) = (x_i(t), \boldsymbol{y}_i(t)), \tag{20}$$

where vector $\boldsymbol{y}_i(t) = (y_{i,j}(t))_{j\in\mathcal{J}}$ represents the selected EN for offloading this task. In Section IV-B, we will discuss about the size of this action space.

C. QoE Function

The QoE function evaluates the influence of agent's actions by taking several key performance factors into account. Given the selected action $\boldsymbol{a}_i(t)$ in the observed state $\boldsymbol{s}_i(t)$, we represent $\mathcal{D}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t))$ as the delay of task $z_i(t)$, which indicates the number of time slots from time slot t to the time slot in which task $z_i(t)$ is processed. It is calculated by:

$$\mathcal{D}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)) = (1 - x_i(t))\Big(l_i^{\mathrm{C}}(t) - t + 1\Big) + x_i(t)\bigg(\sum_{\mathcal{J}}\sum_{t'=t}^{T} \mathbb{1}\big(z_{i,j}^{\mathrm{E}}(t') = z_i(t)\big) l_{i,j}^{\mathrm{E}}(t') - t + 1\bigg), \tag{21}$$

where $\mathcal{D}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)) = 0$ when task $z_i(t)$ is dropped. Correspondingly, we denote the energy consumption of task $z_i(t)$ when taking action $\boldsymbol{a}_i(t)$ in the observed state $\boldsymbol{s}_i(t)$ as $\mathcal{E}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t))$, which is:

$$\mathcal{E}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)) = (1 - x_i(t))E_i^{\mathrm{L}}(t) + x_i(t)\bigg(\sum_{\mathcal{J}}\sum_{t'=t}^{T} \mathbb{1}\big(z_{i,j}^{\mathrm{E}}(t') = z_i(t)\big) E_i^{\mathrm{O}}(t)\bigg). \tag{22}$$

Given the delay and energy consumtion of task $z_i(t)$, we also define $\mathcal{C}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t))$ that denotes the assosiate cost of task $z_i(t)$ given the action $\boldsymbol{a}_i(t)$ in the state $\boldsymbol{s}_i(t)$.

$$\mathcal{C}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)) = \phi_i(t)\,\mathcal{D}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)) + (1 - \phi_i(t))\,\mathcal{E}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)), \tag{23}$$

where $\phi_i(t)$ represents the MD i's battery level. When the MD is operating in performance mode, the primary focus is on minimizing task delays, thus the delay contributes more to the cost. On the other hand, when the MD switches to ultra power-saving mode, the main attention is directed toward reducing power consumption.

Finally, we define $\boldsymbol{q}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t))$ as the QoE associated with task $z_i(t)$ given the selected action $\boldsymbol{a}_i(t)$ and the observed state $\boldsymbol{s}_i(t)$. The QoE function is defined as follows:

$$\boldsymbol{q}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)) = \begin{cases} \mathcal{R} - \mathcal{C}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)) & \text{if task } z_i(t) \text{ is processed,} \\ -\mathcal{E}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)) & \text{if task } z_i(t) \text{ is dropped,} \end{cases} \tag{24}$$

where $\mathcal{R} > 0$ represents a constant reward for task completion. If $z_i(t) = 0$, then $\boldsymbol{q}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)) = 0$. Throughout the rest of this paper, we adopt the shortened notation $\boldsymbol{q}_i(t)$ to represent $\boldsymbol{q}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t))$.

D. Problem Formulation

We define the task offloading policy for MD $i \in \mathcal{I}$ as a mapping from its state to its corresponding action, denoted by i.e., $\pi_i : \mathcal{S} \to \mathcal{A}$. Especially, MD i determines an action $\boldsymbol{a}_i(t) \in \mathcal{A}$, according to policy

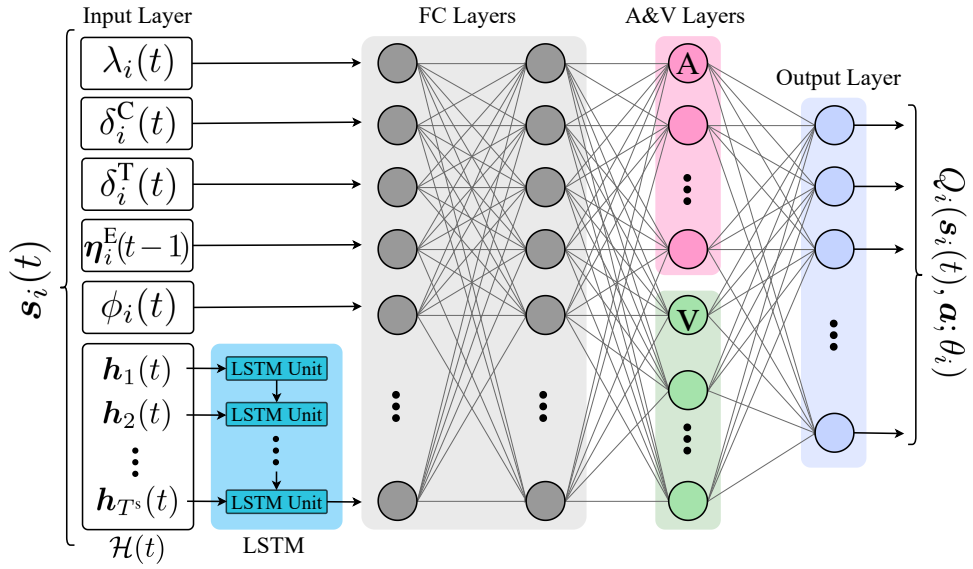


Fig. 2: The neural network of MD $i \in \mathcal{I}$, which characterize the Q-value of each action $\boldsymbol{a} \in \mathcal{A}$ under state $\boldsymbol{s}_i(t) \in \mathcal{S}$.

π_i given the observed environment state $\boldsymbol{s}_i(t) \in \mathcal{S}$. The MD aims to find its optimal policy π_i^* which maximizes the long-term QoE,

$$\pi_i^* = \arg \max_{\pi_i} \mathbb{E}\left[\sum_{t \in \mathcal{T}} \gamma^{t-1} \boldsymbol{q}_i(t) \middle| \pi_i\right], \tag{25}$$

where $\gamma \in (0, 1]$ is a discount factor and determines the balance between instant QoE and long-term QoE. As γ approaches 0, the MD prioritizes QoE within the current time slot exclusively. Conversely, as γ approaches 1, the MD increasingly factors in the cumulative long-term QoE. The expectation $\mathbb{E}[\cdot]$ is taken into consideration of the time-varying system environments. Solving the optimization problem in (25) is particularly challenging due to the dynamic nature of the network. To address this challenge, we introduce a DRL- based offloading algorithm to learn the mapping between each state-action pair and their long-term QoE.

IV. DRL-Based Offloading Algorithm

We now present QECO algorithm so as to address the distributed offloading decision-making of MDs. The aim is to empower MDs to identify the most efficient action that maximizes their long-term QoE. In the following, we introduce a neural network that characterizes the MD's state-action Q-values mapping, followed by a description of the information exchange between the MDs and ENs.

A. DQN-based Approach

We utilize the DQN technique to find the mapping between each state-action pair to Q-values in the formulated MDP. As shown in Fig. 2, each MD $i \in \mathcal{I}$ is equipped with a neural network comprising six layers. These layers include an input layer, an LSTM layer, two dense layers, an advantage-value (A&V)

layer, and an output layer. The parameter vector θ_i of MD i's neural network is defined to maintain the connection weights and neuron biases across all layers. For MD $i \in \mathcal{I}$, we utilize the state information as the input of neural network. The state information $\lambda_i(t)$, $\delta_i^{\mathrm{C}}(t)$, $\delta_i^{\mathrm{T}}(t)$, $\phi_i(t)$, and $\boldsymbol{\eta}_i^{\mathrm{E}}(t-1)$ are directly passed to the dense layer, while the state information $\mathcal{H}(t)$ is first supplied to the LSTM layer and then the resulting output is sent to the dense layer. The role and responsibilities of each layer are detailed as follows.

1) Predicting Workloads at ENs: In order to capture the dynamic behavior of workloads at the ENs, we employ the LSTM network [22]. This network maintains a memory state $\mathcal{H}(t)$ that evolves over time, enabling the neural network to predict future workloads at the ENs based on historical data. By taking the matrix $\mathcal{H}(t)$ as an input, the LSTM network learns the patterns of workload dynamics. The architecture of the LSTM consists of T^{s} units, each equipped with a set of hidden neurons, and it processes individual rows of the matrix $\mathcal{H}(t)$ sequentially. Through this interconnected design, MD tracks the variations in sequences from $\boldsymbol{h}_1(t)$ to $\boldsymbol{h}_{T^{\mathrm{s}}}(t)$, where vector $\boldsymbol{h}_i(t) = (h_{i,j}(t))_{j\in\mathcal{J}}$, thereby revealing workload fluctuations at the ENs across different time slots. The final LSTM unit produces an output that encapsulates the anticipated workload dynamics, and is then connected to the subsequent layer neurons for further learning.

2) State-Action Q-Value Mapping: The pair of dual dense layers plays a crucial role in learning the mapping of Q-values from the current state and the learned load dynamics to the corresponding actions. The dense layers consist of a cluster of neurons that employ rectified linear units (ReLUs) as their activation functions. In the initial dense layer, connections are established from the neurons in the input layer and the LSTM layer to each neuron in the dense layer. The resulting output of a neuron in the dense layer is connected to each neuron in the subsequent dense layer. In the second layer, the outputs from each neuron establish connections with all neurons in the A&V layers.

3) Dueling-DQN Approach for Q-Value Estimation: In the neural network architecture, the A&V layer and the output layer incorporate the principles of the dueling-DQN [26] to compute action Q-values. The fundamental concept of dueling-DQN involves two separate learning components: one for action-advantage values and another for state-value. This approach enhances Q-value estimation by separately evaluating the long-term QoE attributed to states and actions.

The A&V layer consists of two distinct dense networks referred to as network A and network V. Network A's role is to learn the action-advantage value for each action, while network V focuses on learning the state-value. For an MD $i \in \mathcal{I}$, we define $V_i(\boldsymbol{s}_i(t);\theta_i)$ and $A_i(\boldsymbol{s}_i(t),\boldsymbol{a};\theta_i)$ to denote the state-value and the action-advantage value of action $\boldsymbol{a} \in \mathcal{A}$ under state $\boldsymbol{s}_i(t) \in \mathcal{S}$, respectively. The parameter θ_i is responsible for determining these values, and it can be adjusted when training the QECO algorithm.

For an MD $i \in \mathcal{I}$, the A&V layer and the output layer collectively determine $Q_i(\boldsymbol{s}_i(t),\boldsymbol{a};\theta_i)$, representing the resulting Q-value under action $\boldsymbol{a} \in \mathcal{A}$ and state $\boldsymbol{s}_i(t) \in \mathcal{S}$, as follows:

$$Q_i(\boldsymbol{s}_i(t),\boldsymbol{a};\theta_i) = V_i(\boldsymbol{s}_i(t);\theta_i) + \left(A_i(\boldsymbol{s}_i(t),\boldsymbol{a};\theta_i) - \frac{1}{|\mathcal{A}|}\sum_{\boldsymbol{a}'\in\mathcal{A}} (A_i(\boldsymbol{s}_i(t),\boldsymbol{a}';\theta_i) \right), \tag{26}$$

where θ_i establishes a functional relationship that maps Q-values to pairs of state-action.

B. QoE-Oriented DRL-Based Algorithm

The QECO algorithm is meticulously designed to optimize the allocation of computational tasks between MDs and ENs. Since the training of neural networks imposes an extensive computational workload on MDs, we enable MDs to utilize ENs for training their neural networks, effectively reducing their computational workload. For each MD $i \in \mathcal{I}$, there is an associated EN, denoted as EN $j_i \in \mathcal{J}$, which assists in the training process. This EN possesses the highest transmission capacity among all ENs. We define $\mathcal{I}_j \subset \mathcal{I}$ as the set of MDs for which training is executed by EN $j \in \mathcal{J}$, i.e. $\mathcal{I}_j = \{i \in \mathcal{I} | j_i = j\}$. This approach is feasible due to the minimal information exchange and processing requirements for training compared to MD's tasks. The algorithms to be executed at MD $i \in \mathcal{I}$ and EN $j \in \mathcal{J}$ are given in Algorithms **??** and 4, respectively. The core concept involves training neural networks with MD experiences (i.e., state, action, QoE, next state) to map Q-values to each state-action pair. This mapping allows MD to identify the action in the observed state with the highest Q-value and maximize its long-term QoE.

In detail, EN $j \in \mathcal{J}$ maintains a replay buffer denotes as $\mathcal{M}_i$ with two neural networks for MD i: Net_i^{E}, denoting the evaluation network, and Net_i^{T}, denoting the target network, which have the same neural network architecture. However, they possess distinct parameter vectors θ_i^{E} and θ_i^{T}, respectively. Their Q-values are represented by $Q_i^{\mathrm{E}}(\boldsymbol{s}_i(t), \boldsymbol{a}; \theta_i^{\mathrm{E}})$ and $Q_i^{\mathrm{T}}(\boldsymbol{s}_i(t), \boldsymbol{a}; \theta_i^{\mathrm{T}})$ for MD $i \in \mathcal{I}_j$, respectively, associating the action $\boldsymbol{a} \in \mathcal{A}$ under the state $\boldsymbol{s}_i(t) \in \mathcal{S}$. The replay buffer records the observed experience $(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t), \boldsymbol{q}_i(t), \boldsymbol{s}_i(t+1))$ of MD i. Moreover, Net_i^{E} is responsible for action selection, while Net_i^{T} characterizes the target Q-values, which represent the estimated long-term QoE resulting from an action in the observed state. The target Q-value serves as the reference for updating the network parameter vector θ_i^{E}. This update occurs through the minimization of disparities between the Q-values under Net_i^{E} and Net_i^{T}. In the following, we introduce the offloading decision algorithm of MD $i \in \mathcal{I}$ and the training process algorithm running in EN $j \in \mathcal{J}$.

1) Offloading Decision Algorithm at MD $i \in \mathcal{I}$: We analyze a series of episodes, where N^{ep} denotes the number of them. At the beginning of each episode, if MD $i \in \mathcal{I}$ receives a new task $z_i(t)$, it initializes the state $\mathcal{S}_i(1)$ and sends an *UpdateRequest* to EN j_i. After receiving the requested vector θ_i^{E} of Net_i^{E} from EN j_i, MD i chooses the following action for task $z_i(t)$.

$$\boldsymbol{a}_i(t) = \begin{cases} \arg\ \max_{\boldsymbol{a} \in \mathcal{A}} Q_i^{\mathrm{E}}(\boldsymbol{s}_i(t), \boldsymbol{a}; \theta_i^{\mathrm{E}}), & \text{w.p. } 1-\boldsymbol{\epsilon}, \\ \text{pick a random action from } \mathcal{A}, & \text{w.p. } \boldsymbol{\epsilon}, \end{cases} \tag{27}$$

where w.p. stands for with probability, and $\boldsymbol{\epsilon}$ represents the random exploration probability. The value of $Q_i^{\mathrm{E}}(\boldsymbol{s}_i(t), \boldsymbol{a}; \theta_i^{\mathrm{E}})$ indicates the Q-value under the parameter θ_i^{E} of the neural network Net_i^{E}. Specifically, the MD with a probability of $1-\boldsymbol{\epsilon}$ selects the action associated with the highest Q-value under Net_i^{E} in the observed state $\boldsymbol{s}_i(t)$.

In the next time slot $t+1$, MD i observes the state $\mathcal{S}_i(t+1)$. However, due to the potential for tasks to extend across multiple time slots, QoE $\boldsymbol{q}_i(t)$ associated with task $z_i(t)$ may not be observable in time slot $t+1$. On the other hand, MD i may observe a group of QoEs associated with some tasks $z_i(t')$ in time slots $t' \leq t$. For each MD i, we define the set $\mathcal{F}_i^t \subset \mathcal{T}$ to denote the time slots during which each arriving task $z_i(t')$ is either processed or dropped in time slot t, as given by:

$$\mathcal{F}_i^t = \Bigg\{ t' \Bigg| \; t' \leq t,\; \lambda_i(t') > 0,\; (1 - x_i(t'))\; l_i^{\mathrm{C}}(t') + x_i(t') \sum_{\mathcal{J}} \sum_{n=t'}^{t} \mathbb{1}(z_{i,j}^{\mathrm{E}}(n) = z_i(t'))\; l_{i,j}^{\mathrm{E}}(n) = t \Bigg\}.$$

Therefore, MD i observes a set of QoEs $\{\boldsymbol{q}_i(t') \mid t' \in \mathcal{F}_i^t\}$ at the beginning of time slot $t+1$, where the set $\mathcal{F}_i^t$ for some $i \in \mathcal{I}$ can be empty. Subsequently, MD i sends its experience $(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t), \boldsymbol{q}_i(t), \boldsymbol{s}_i(t+1))$ to EN j_i for each task $z_i(t')$ in $t' \in \mathcal{F}_i^t$.

2) Training Process Algorithm at EN $j \in \mathcal{J}$: Upon initializing the replay buffer $\mathcal{M}_i$ with the neural networks $\mathit{Net}_i^{\mathrm{E}}$ and $\mathit{Net}_i^{\mathrm{T}}$ for each MD $i \in \mathcal{I}_j$, EN $j \in \mathcal{J}$ waits for messages from the MDs in the set $\mathcal{I}_j$. When EN j receives an *UpdateRequest* signal from an MD $i \in \mathcal{I}_j$, it responds by transmitting the updated parameter vector θ_i^{E}, obtained from $\mathit{Net}_i^{\mathrm{E}}$, back to MD i. On the other side, if EN j receives an experience $(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t), \boldsymbol{q}_i(t), \boldsymbol{s}_i(t+1))$ from MD $i \in \mathcal{I}_j$, the EN stores this experience in the replay buffer $\mathcal{M}_i$ associated with that MD.

The EN randomly selects a sample collection of experiences from the replay buffer, denoted as $\mathcal{N}$. For each experience $n \in \mathcal{N}$, it calculates the value of $\hat{Q}_{i,n}^{\mathrm{T}}$. This value represents the QoE in experience n and includes a discounted Q-value of the action anticipated to be taken in the subsequent state of experience n, according to the network $\mathit{Net}_i^{\mathrm{T}}$, given by

$$\hat{Q}_{i,n}^{\mathrm{T}} = \boldsymbol{q}_i(n) + \gamma Q_i^{\mathrm{T}}(\boldsymbol{s}_i(n+1)), \tilde{\boldsymbol{a}}_n; \theta_i^{\mathrm{T}}), \tag{28}$$

where $\tilde{\boldsymbol{a}}_n$ denotes the optimal action for the state $\boldsymbol{s}_i(n+1)$ based on its highest Q-value under $\mathit{Net}_i^{\mathrm{E}}$, as given by:

$$\tilde{\boldsymbol{a}}_n = \arg \max_{\boldsymbol{a} \in \mathcal{A}} Q_i^{\mathrm{E}}(\boldsymbol{s}_i(n+1), \boldsymbol{a}; \theta_i^{\mathrm{E}}). \tag{29}$$

In particular, regarding experience n, the target-Q value $\hat{Q}_{i,n}^{\mathrm{T}}$ represents the long-term QoE for action $\boldsymbol{a}_i(n)$ under state $\boldsymbol{s}_i(n)$. This value corresponds to the QoE observed in experience n, as well as the approximate expected upcoming QoE. Based on the set $\mathcal{N}$, the EN trains the MD's neural network using previous sample experiences. Simultaneously, it updates θ_i^{E} in $\mathit{Net}_i^{\mathrm{E}}$ and computes vector $\hat{\mathbf{Q}}_i^{\mathrm{T}} = (\hat{Q}_{i,n}^{\mathrm{T}})_{n \in \mathcal{N}}$. The key idea of updating $\mathit{Net}_i^{\mathrm{E}}$ is to minimize the disparity in Q-values between $\mathit{Net}_i^{\mathrm{E}}$ and $\mathit{Net}_i^{\mathrm{T}}$, as indicated by the following loss function:

$$L(\theta_i^{\mathrm{E}}, \hat{\mathbf{Q}}_i^{\mathrm{T}}) = \frac{1}{|\mathcal{N}|} \sum_{n \in \mathcal{N}} \left(Q_i^{\mathrm{E}}(\boldsymbol{s}_i(n), \boldsymbol{a}_i(n); \theta_i^{\mathrm{E}}) - \hat{Q}_{i,n}^{\mathrm{T}} \right)^2. \tag{30}$$

In every *ReplaceThreshold* iterations, the update of $\mathit{Net}_i^{\mathrm{T}}$ will involve duplicating the parameters from $\mathit{Net}_i^{\mathrm{E}}$ ($\theta_i^T = \theta_i^E$). The objective is to consistently update the network parameter θ_i^T in $\mathit{Net}_i^{\mathrm{T}}$, which enhances the approximation of the long-term QoE when computing the target Q-values in (28).

3) Computational Complexity: The computational complexity of the QECO algorithm is determined by the number of experiences required to discover the optimal offloading policy. Each experience involves backpropagation for training, which has a computational complexity of $\mathcal{O}(C)$, where C represents the

number of multiplication operations in the neural network. During each training round triggered by the arrival of a new task, a sample collection of experiences of size $|\mathcal{N}|$ is utilized from the replay buffer. Since the training process encompasses N^{ep} episodes and there are K expected tasks in each episode, the computational complexity of the proposed algorithm is $\mathcal{O}(N^{\text{ep}}K|\mathcal{N}|C)$, which is polynomial. Given the integration of neural networks for function approximation, the convergence guarantee of the DRL algorithm remains an open problem. In this work, we will empirically evaluate the convergence of the proposed algorithm in Section V-B.

V. Performance Evaluation

In this section, we first present the simulation setup and training configuration. We then illustrate the convergence of the proposed DRL-based QECO algorithm and evaluate its performance in comparison to three baseline schemes in addition to the existing work [23].

A. Simulation Setup

We consider a MEC environment with 50 MDs and 5 ENs, similar to [12]. We also follow the model presented in [17] to determine the energy consumption. All the parameters are given in Table III. To train the MDs' neural networks, we adopt a scenario comprising 1000 episodes. Each episode contains 100 time slots, each of length 0.1 second. The QECO algorithm incorporates real-time experience into its training process to continuously enhance the offloading strategy. Specifically, we employ a batch size of 16, maintain a fixed learning rate of 0.001, and set the discount factor γ to 0.9. The probability of random exploration gradually decreases from an initial value 1, progressively approaching 0.01, all of which is facilitated by an RMSProp optimizer.

We use the following methods as benchmarks.

1) **Local Computing (LC):** The MDs execute all of their computation tasks using their own computing capacity.
2) **Full Offloading (FO):** Each MD dispatches all of its computation tasks while choosing the offloading target randomly.
3) **Random Decision (RD):** In this approach, when an MD receives a new task, it randomly makes the offloading decisions and selects the offloading target if it decides to dispatch the task.
4) **PGOA [23]:** This existing method is a distributed optimization algorithm designed for delay-sensitive tasks in an environment where MDs interact strategically with multiple ENs. We select PGOA as a benchmark method due to its similarity to our work.

B. Performance Comparison and Convergence

We first evaluate the number of completed tasks when comparing our proposed QECO algorithm with the other four schemes. As illustrated in Fig. 3 (a), the QECO algorithm consistently outperforms the benchmark methods when we vary the task arrival rate. At a lower task arrival rate (i.e., 50), most of the methods demonstrate similar proficiency in completing tasks. However, as the task arrival rate increases, the efficiency of QECO becomes more evident. Specifically, when the task arrival rate increases

TABLE I: Simulation Parameters

Parameter	**Value**
Computation capacity of MD f_i	2.6 GHz
Computation capacity of EN f_j^{E}	42.8 GHz
Transmission capacity of MD $r_{i,j}(t)$	14 Mbps
Task arrival rate	150 Task/sec
Size of task $\lambda_i(t)$	$\{1.0, 1.1, \ldots, 7.0\}$ Mbits
Required CPU cycles of task $\rho_i(t)$	$\{0.197,0.297,0.397\} \times 10^3$
Deadline of task Δ_i	10 time slots (1 Sec)
Battery level percentage of MD $\phi_i(t)$	$\{25, 50, 75\}$
Computation power of EN p_j^{E}	5 W
Transmission power of MD p_i^{T}	2.3 W
Standby power of MD p_i^{I}	0.1 W

to 250, our algorithm can increase the number of completed tasks by 73% and 47% compared to RD and PGOA, respectively. Similarly, in Fig. 3 (b), as the number of MDs increases, QECO shows significant improvements in the number of completed tasks compared to other methods, especially when faced with a large number of MDs. When there are 110 MDs, our proposed algorithm can effectively increase the number of completed tasks by at least 34% comparing with other methods. This achievement is attributed to the QECO's ability to effectively handle unknown workloads and prevent congestion at the ENs.

Figs. 4 (a) and 4 (b) illustrate the overall energy consumption for different values of task arrival rate and the number of MDs, respectively. At the lower task arrival rate, the total energy consumption of all methods is close to each other. The total energy consumption increases when we have a higher task arrival rate. As can be observed from Fig. 4 (a), at task arrival rate 450, QECO effectively reduces overall energy consumption by 18% and 15% compared to RD and PGOA, respectively, as it takes into account the battery level of the MD in its decision-making process. However, it consumes more energy compared to LC and FO because they do not utilize all computing resources. In particular, LC only uses the MD's computational resources, while FO utilizes the allocated EN computing resources.

In Fig. 4 (b), an increasing trend in overall energy consumption is observed as the number of MDs increases since the number of resources available in the system increases, which leads to higher energy consumption. The QECO algorithm consistently outperforms RD and PGOA methods in overall energy consumption, especially when there are a large number of MDs. Specifically, QECO demonstrates a 27% and 16% reduction in overall energy consumption compared to RD and PGOA, respectively, when the number of MDs increases to 110.

As shown in Fig. 5 (a), the QECO algorithm maintains a lower average delay compared to other methods as the task arrival rate increases from 50 to 350. Specifically, when the task arrival rate is 200, it reduces the average delay by at least 12% compared to other methods. However, for task arrival rates exceeding 350, QECO may experience a higher average delay compared to some of the other methods. This can be attributed to the fact that the other algorithms drop more tasks while our proposed algorithm is capable of completing a higher number of tasks, potentially leading to an increase in average delay. In Fig. 5 (b), as

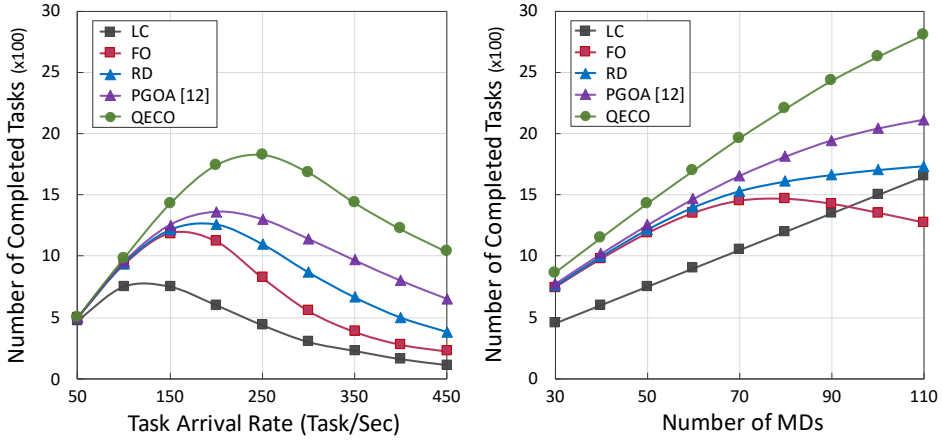


Fig. 3: The number of completed tasks under different computation workloads: (a) task arrival rate; (b) the number of MDs.

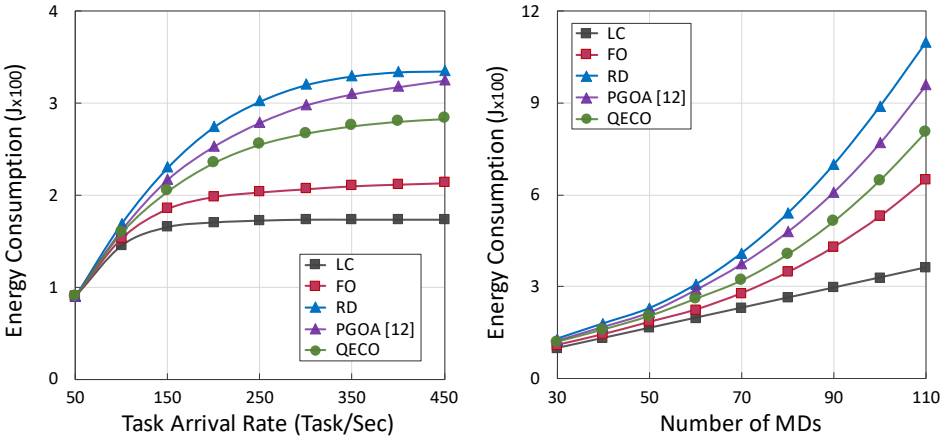
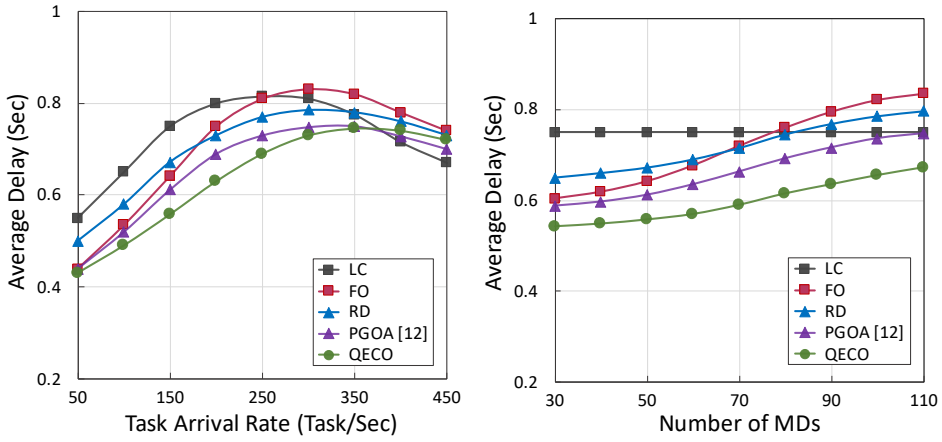


Fig. 4: The overall energy consumption under different computation workloads: (a) task arrival rate; (b) the number of MDs.

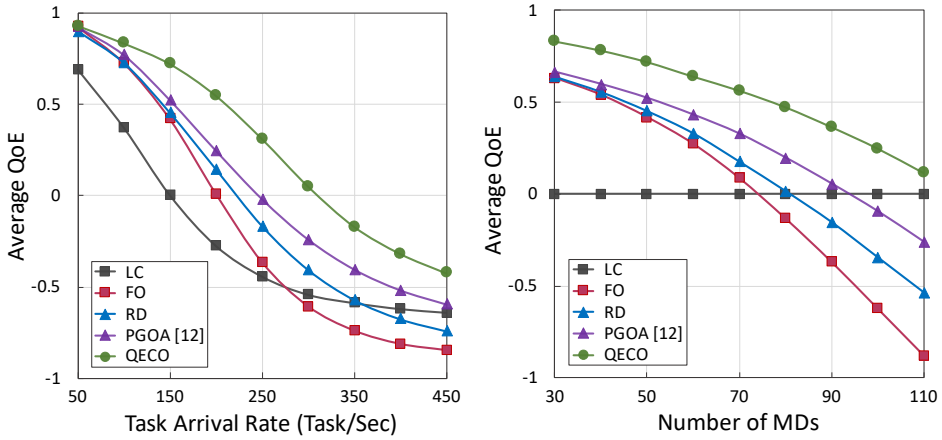
the number of MDs increases, we observe a rising trend in the average delay. It can be inferred that an increase in computational load in the system can lead to higher queuing delays and computations at ENs. Considering the QECO's ability to schedule workloads, when the number of MDs increases from 30 to 110, it consistently maintains a lower average delay which is at least 8% less than the other methods.

We further investigate the overall improvement achieved by the QECO algorithm in comparison to other methods in terms of the average QoE. This metric signifies the advantages MDs obtain by utilizing different algorithms. Fig. 6 (a) shows the average QoE for different values of task arrival rate. This figure indicates the superiority of the QECO algorithm in providing MDs with an enhanced experience. Specifically, when the task arrival rate is moderate (i.e., 250), QECO improves the average QoE by 57% and 33% compared



(a) (b)

Fig. 5: The average delay under different computation workloads: (a) task arrival rate; (b) the number of MDs.



(a) (b)

Fig. 6: The average QoE under different computation workloads: (a) task arrival rate; (b) the number of MDs.

to RD and PGOA, respectively. Fig. 6 (b) illustrates the average QoE when we increase the number of MDs. The EN's workload grows when there are a larger number of MDs, leading to a reduction in the average QoE of all methods except LC. However, QECO effectively manages the uncertain load at the ENs. When the number of MDs increases to 90, QECO achieves at least a 29% higher QoE comparing with the other methods. It is worth noting that although improvements in each of the QoE factors can contribute to enhancing system performance, it is essential to consider the user's demands in each time slot. Therefore, the key difference between QECO and other methods is that it prioritizes users' demands, enabling it to strike an appropriate balance among them, ultimately leading to a higher QoE for MDs.

We finally delve into the investigation of the convergence performance of the QECO algorithm, which is

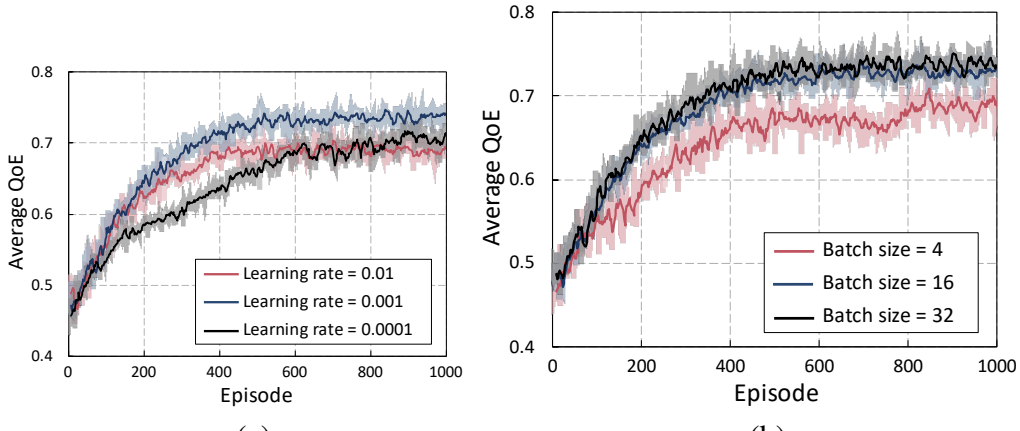


Fig. 7: The convergence of the average QoE across episodes under different hyper-parameters: (a) Learning rate; (b) Batch size.

shown through the average QoE across episodes in Figs. 7 (a) and 7 (b). We explore the impact of two main hyper-parameters on the convergence speed and the converged result of the proposed algorithm. Fig. 7 (a) illustrates the convergence of the proposed algorithm under different learning rates, where the learning rate regulates the step size per iteration towards minimizing the loss function. The QECO algorithm achieves an average QoE of 0.77 after around 400 episodes when the learning rate is 0.001, indicating relatively rapid convergence. However, with smaller learning rates (e.g., 0.0001) or larger values (e.g., 0.01), a slower convergence is observed. Fig. 7 (b) shows the convergence of the proposed algorithm under different batch sizes, which refer to the number of sampled experiences in each training round. An improvement in convergence performance is observed as the batch size increases from 4 to 16. However, further increasing the batch size from 16 to 32 does not notably enhance the converged QoE or convergence speed. Hence, a batch size of 16 may be more appropriate for training processes.

VI. Conclusion

In this paper, we focused on addressing the challenge of offloading in MEC systems, where strict task processing deadlines and energy constraints adversely impact system performance. We formulated an optimization problem that aims to maximize the QoE of each MD individually, while QoE reflects the energy consumption and task completion delay. To address the dynamic and uncertain mobile environment, we proposed a QoE-oriented DRL-based computation offloading algorithm called QECO. Our proposed algorithm empowers MDs to make offloading decisions without relying on knowledge about task models or other MDs' offloading decisions. The QECO algorithm not only adapts to the uncertain dynamics of load levels at ENs, but also effectively manages the ever-changing system environment. Through extensive simulations, we showed that QECO outperforms several established benchmark techniques, while demonstrating a rapid training convergence. Specifically, QECO increases the average user's QoE by 37% compared to an existing work. This advantage can lead to improvements in key performance metrics, including task completion rate, task delay, and energy consumption, under different system conditions and varying user demands.

There are multiple directions for future work. A complementary approach involves extending the task model by considering interdependencies among tasks. This can be achieved by incorporating a task call graph representation. Furthermore, in order to accelerate the learning of optimal offloading policies, it will be beneficial to take advantages of federated learning techniques in the training process. This will allow MDs to collectively contribute to improving the offloading model and enable continuous learning when new MDs join the network.

Abstract

Mobile edge computing (MEC) is envisioned to address the computation demands of Internet of Things (IoT) devices. However, it is crucial for the MEC to operate in coordination with the cloud tier to achieve a highly scalable IoT system. In addition, IoT devices require regular maintenance to either recharge or replace their batteries which may not always be feasible. Wireless energy transfer (WET) can provide IoT devices a stable source of energy. Nonetheless, proper scheduling of energy harvesting and efficient allocation of computing resources are the key for sustainable operation of these devices. In this paper, we introduce a three-tier wireless powered mobile edge computing (WPMEC) consisting of cloud, MEC servers, and IoT devices. We first formulate a combinatorial optimization problem that aims to minimize the wireless energy transmission. To tackle the complexity of the problem, we use bipartite graph matching and propose a harvest-then-offload mechanism for IoT devices. We also exploit parallel processing to increase the performance of the proposed algorithm. Through numerical experiments, we evaluate the performance of our proposed mechanism. Our results show that the proposed mechanism significantly reduces the required energy for the operation of IoT devices comparing to different offloading policies. We further show that WiEnTM results in up to 34% less wireless energy transmission in comparison to an existing work in the literature.

Index Terms

Bipartite graph matching, mobile edge computing (MEC), wireless power transfer (WPT), Internet of Things (IoT)

VII. Introduction

The Internet of Things (IoT) applications (e.g., smart cities [27], homes [28], and augmented/virtual reality [29]) are mostly involved with computation-intensive and latency-sensitive tasks. Mobile edge computing (MEC) is a state-of-the-art computation paradigm to address the challenge of insufficient computing resources in IoT networks [30], [31]. By deploying the edge servers, nearby devices can offload their computation tasks. Dispatching those tasks that require a higher amount of energy will let devices save more energy [32]. However, as the computation tasks become more and more intensive, coordination with cloud servers will be inevitable [33]. The three-tier computing models consisting of cloud, edge, and IoT devices are introduced to overcome the aforementioned challenge [34]–[37].

In addition to their restricted computation resources, IoT devices usually have small battery capacities which is mainly due to their small form factor and the constraints of the production cost [27]. Thus, their battery needs to be replaced or recharged quite often which may be infeasible or costly in some cases such as toxic [38] or hard-to-reach environments [31]. Thanks to the latest developments of radio frequency (RF) enabled wireless energy transfer (WET) technology, a promising solution called wireless powered communication (WPC) has been introduced [39]. In WPC, wireless devices (WDs) are powered by dedicated energy transmitters (ETs) [39]. As wireless powered networks do not require replacement or manual recharging of batteries, they have significantly lower maintenance costs. Therefore, these networks are more scalable compared to the battery powered networks in terms of size [39]. By combining the advantages of MEC and WPC, the newly emerged wireless powered MEC (WPMEC) holds a major promise to overcome the fundamental hurdles in scalability of the IoT networks [40]–[43].

Computation offloading has attracted many research projects in recent years [29], [34], [36], [37], [44]–[48]. It helps WDs save more energy and reduce the tasks' execution time. As energy is crucial for the operation and maintenance of wireless powered systems, many of these works are focusing on analyzing and optimizing energy. Aazam *et al.* in [34] analyzed the energy and performance of a three-tier cloud-edge-IoT offloading model. In [36], authors proposed a scheme that aims to minimize the total cost in terms of energy and delay for a three-tier cloud-edge-IoT system. Gao *et al.* in [37] proposed an offloading algorithm to minimize an objective called the energy time cost for a three-tier network. Although [34], [36], and [37] all benefit from a computationally-scalable three-tier system, IoT devices still rely on battery energy and hence lack the sustainability feature of WPMEC systems. In [29], authors provided a method to maximize energy efficiency in a two-user WPMEC system with user cooperation, non-divisible tasks, and harvest-then-offload TDMA protocol. Wang *et al.* in [44] presented a heuristic solution to reduce the ET's total transmission energy consumption over a finite horizon in a single-user WPMEC system. In [45], authors presented a single-user WPMEC framework with task partitioning to jointly optimize the computation rate and transmit energy. The methods proposed in [29], [44], [45] are not suitable for larger networks. Zhang *et al.* in [46] introduced a decentralized game for a multi-user MEC system to optimize the ET's energy and computation costs. Nonetheless, reaching an equilibrium may require a considerable number of iterations. It also needs all WDs to cooperate with each other which may not be feasible in many cases. Huang *et. al* in [47] proposed an offloading algorithm based on deep reinforcement learning for WPMEC systems with non-divisible tasks to maximize a value function (e.g., computation rate or energy efficiency). However, the proposed system model is oversimplified and thus less applicable to real-world scenarios. He *et al.* in [48] designed a scheduling algorithm to minimize the processing energy consumption for a multi-user WPMEC network with multiple access points (APs) and non-divisible tasks. As a task usually models a program, it consists of a number of interrelated subprograms. It may also include subprograms to send and retrieve data which cannot be offloaded. Therefore, non-divisible task models used in [29], [46], [47], and [48] may not accurately represent real world applications. Additionally, such models cannot properly utilize multiple computing resources.

By considering the challenges of the aforementioned works, the following questions arise:

1) *How can we take advantage of cloud computing in addition to MEC to meet different requirements of IoT applications?*
2) *How can we provide RF energy to enable the battery-less IoT devices operate seamlessly?*
3) *What would be the optimal offloading decision when considering the subtasks' interdependencies?*

To answer these questions, we propose a novel three-tier WPMEC system with multiple WDs to minimize the ET's total transmitted energy. In the proposed system, we partition tasks into a number of fine-grained subtasks to consider their interdependencies. We design an offloading and scheduling algorithm to be executed efficiently on parallel processors such as graphical processing units (GPUs). This significantly reduces the execution time of the algorithm.

Our main contributions are as follows.

1) *Problem formulation*: We formally define the mixed-integer ET's energy minimization (ETEM) problem by deciding on offloading, WET time, and computation scheduling of subtasks for a multi-

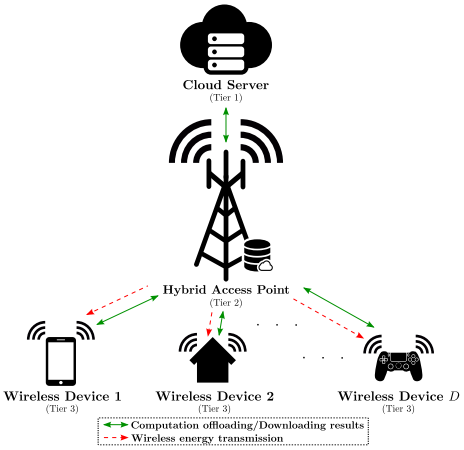


Fig. 8: Proposed three-tier WPMEC system in an IoT network.

user three-tier WPMEC system considering task interdependencies and MEC server computation time.

2) *Algorithm design*: We propose wireless energy transmission minimization (WiEnTM) algorithm based on bipartite graph matching. We first study the properties of the problem to decompose it into individual problems for each subtask to make it suitable for the matching algorithm. By efficiently utilizing parallel processing on decomposed problems, we prove that our proposed algorithm has a logarithmic time complexity with respect to the network size.
3) *Performance evaluation*: Extensive numerical experiments are carried out to evaluate the proposed algorithm. Our experiments show major reduction in the WDs' required energy in comparison with baseline offloading policies. The algorithm also reduces the energy consumption by 34% for large tasks compared to an existing work [44].

The rest of this paper is organized as follows. In Section II, we first present the MEC system model including the edge and cloud servers and IoT devices. We then present the computation task partitioning model. Section III consists of the energy optimization problem and a step-by-step formulation. In Section IV, we propose our WiEnTM algorithm to efficiently solve the problem. Simulation results are presented in Section V. We conclude the paper in Section VI.

VIII. System Model

In this section, we first introduce the IoT network and computation task models. The IoT network consists of IoT devices and computing servers. The task model represents the subtasks as a directed acyclic graph (DAG) and introduces its associated properties. Some of the key notations are listed in Table II.

A. Network Model

As illustrated in Fig. 8, we consider a three-tier WPMEC system with a cloud server and a hybrid access point (HAP)[1] in a network of multiple WDs. The HAP is equipped with an MEC server and an ET. MEC and the cloud servers are both able to receive and execute the offloaded tasks and then send the results back to WDs. ET broadcasts stable RF power to WDs. Thus, with proper scheduling, WDs can sustain their operation by only relying on the harvested energy [39]. In order to allow multiple WDs to share the same RF-channel, we assume a time-division multiple access (TDMA) protocol for WDs, as inspired by [29] and [49]. Moreover, we assume that WDs have half-duplex transmission. As a result, computation offloading cannot be performed simultaneously with energy harvesting [50]. We thus adopt the *harvest-then-offload* [50] model of scheduling, as shown in Fig. 9. In contrast, for the case of full-duplex transmission, scheduling algorithms are categorized as the simultaneous wireless information and power transfer (SWIPT) [39] mode. However, considering the constraints of the production costs and devices' form factor for battery-less WDs, they usually lack a full-duplex transceiver for energy harvesting and data transmission.

We now introduce the detailed network model. Let $\mathcal{D} \triangleq \{1, ..., D\}$ denote the set of WDs where D indicates the total number of WDs. Each device $d \in \mathcal{D}$ is connected to the HAP and hence, the cloud server, using a wireless interface. We regard a complete cycle of executing a single subtask for all WDs having a candidate subtask as an *episode*. Each episode jointly models the harvest-then-offload timing, subtasks' deadlines, and TDMA scheduling of WDs. As shown in Fig. 9, an episode of length Γ is divided into two time frames: WET time (i.e., harvesting time) and offloading/local computation time. In the case of offloading, the latter is consisted of three time slots. The first slot is to relocate the task bits from WD $d \in \mathcal{D}$ to the HAP. Then, the task should be either executed on the MEC server or transmitted to the cloud. We consider that the computation time on the cloud server is negligible due to the high computation capacity compared to WDs or the MEC server. Thus, in the case of offloading to the cloud server, the second time slot indicates the time required to send the task data of WD $d \in \mathcal{D}$ from the HAP to the cloud server and then retrieve the processed data. Finally, the results should be transmitted from the HAP back to the corresponding WD in the third time slot.

Now that we have introduced the network model, we need to provide an approach to calculate the length of episode Γ. As stated earlier, each episode reflects the requirements of the harvest-then-offload timing and subtask deadlines. Therefore, Γ is directly related to the individual subtask deadlines being executed in that episode. In other words, Γ can be calculated as the summation of the subtasks' deadlines. In the next section, we will calculate the length of each episode.

B. Task Model

Generally, there are two main methods for computation offloading: 1) *Binary offloading*, and 2) *partial offloading* which is also known as *program partitioning* [31]. In the former method, each task, as an

[1]Our analysis can be extended to the case of multiple HAPs by modeling the MEC server as a pool of edge computing servers and considering the fact that each IoT device can only be connected to an HAP.

TABLE II: List of key notations

Symbol	Description
$x^{\mathrm{L}}_{v^u_d}, x^{\mathrm{M}}_{v^u_d}, x^{\mathrm{C}}_{v^u_d}$	Binary offloading decisions
τ_{1,v^u_d}	Length of scheduled time for data transmission of subtask v^u_d
τ_{2,v^u_d}	Length of scheduled time for execution of subtask v^u_d
τ_{3,v^u_d}	Length of scheduled time for retrieving the result of subtask v^u_d
τ^u_0	WET time in episode u
p^u	ET's power in episode u
Γ^u	Length of episode u
$\mu_{v^u_d}$	Response time of cloud server for subtask v^u_d
$C_{v^u_d}$	Number of CPU cycles required for subtask v^u_d
$N_{i,j}$	Number of bits required to transmit the result of subtask i and j
g_d	Channel gain between WD d and ET
h_d	Channel gain between WD d and MEC server
σ^2	Noise power at the HAP
η_d	Energy harvesting efficiency of WD d
ζ_d	CPU's effective switched capacitance of WD d
r_{t,v^u_d}	Virtual resource for tier t and subtask v^u_d
$F^{\mathrm{M,max}}$	Maximum CPU frequency of MEC server
$F^{\mathrm{L,max}}_d$	Maximum CPU frequency of WD d
$P^{\mathrm{ET,max}}$	ET's maximum transmission power
$P^{\mathrm{AP,max}}$	AP's maximum transmission power
$P^{\mathrm{WD,max}}_d$	Maximum transmission power of WD d
D	Number of WDs
V_d	Number of subtasks of WD d
B_d	Bandwidth of WD d
$\mathcal{D}$, $\mathcal{D}^u$	Set of WDs
$\mathcal{U}$	Set of episodes
$\mathcal{V}_d$	Set of subtasks of WD d
$\mathcal{R}^{\mathrm{virt}}$	Set of virtual resources
$\mathcal{I}_{v^u_d}$	Set of episodes associated to executing prerequisite subtasks of the subtask v^u_d
$\mathcal{J}_{v^u_d}$	Set of episodes associated to executing subtasks with the subtask v^u_d as their prerequisite

independent unit, is either fully computed locally or offloaded to an MEC server, based on the system constraints such as task arrival time, deadline, and energy harvesting-related constraints. However, the latter provides a more fine-grained model of tasks by dividing each of them into a number of interrelated subtasks based on the dependencies inside each task [31]. These subtasks together form a directed, weighted, and acyclic graph known as *task call graph* [31]. The partial offloading provides a more realistic model of tasks as it leverages the task call graph to represent each task. This accurately models the call stack of

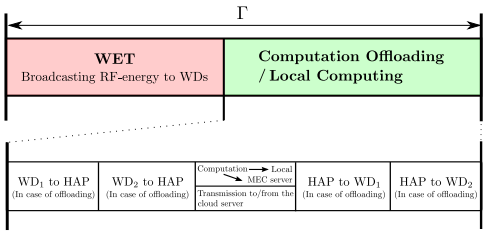


Fig. 9: Harvest-then-offload scheduling with TDMA protocol for a system of two WDs. An episode of length Γ is divided into two times frames of WET and offloading/local computing. In the case of offloading, the offloading/local computing time frame consists of three slots. The first and last slots are for transmitting the data and retrieving the results, respectively. The second slot is dedicated to the computation in the MEC server or transmission to/from the cloud server.

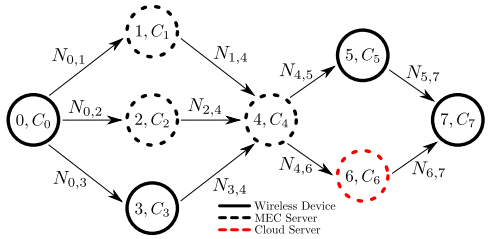


Fig. 10: An example of task call graph with eight subtasks for device d. Each parameter C_i indicates the computation intensity of subtask i in terms of the number of CPU cycles. Weights $N_{i,j}$ represent the number of bits transferred between subtasks i and j.

a computer program[2] It can also lead to building networks with a higher performance as the optimal offloading decisions are made for each subtask instead of the whole task. Apart from that, by leveraging program partitioning, one can extract and process them in parallel. That being said, it can be a preferred model for many applications including distributively training deep neural networks [27], advanced image processing based on parallel processing, and also most of the augmented reality applications [52].

In this paper, we assume that each device $d \in \mathcal{D}$ has a task which can be partitioned into a number of V_d subtasks using the program partitioning method. We denote $\mathcal{G}_d(\mathcal{V}_d, \mathcal{E}_d)$ as the task call graph of device $d \in \mathcal{D}$ where $\mathcal{V}_d$ indicates the set of vertices which are the subtasks and $\mathcal{E}_d$ denotes the set of edges representing the connection between each two subtasks. Each directed edge $(i, j) \in \mathcal{E}_d$ shows the transmission of the i-th subtask output to the j-th subtask. An example of this graph with specified computation modes is shown in Fig. 10. For illustration purposes, solid-circled vertices show the subtasks that are computed locally, while vertices with dashed circles correspond to the offloaded subtasks. The black ones are computed on the nearby MEC server and the red one is computed on the remote cloud server. In addition, each subtask i is labeled by a parameter C_i which indicates the required CPU cycles to perform the task and represents the task computation intensity. The weight $N_{i,j}$ of the edge from i to j denotes the number of bits required to transmit the result of subtask i to subtask j. As the task requires

[2]For instance, *Pyan* [51] is a Python module that generates the task call graph of a Python code by analysis of its modules and functions.

the subtasks to accumulate or send/receive data at the start and end of its procedure, the ones at these ends are executed locally and no offloading decision is made for them [31].

IX. Problem Formulation

This section presents the steps toward formulating the ETEM problem. After providing motivations behind the selection of the objective, we formulate each part of the problem from the objective function to the constraints.

A. Problem Objective

Due to the high attenuation of RF energy over distances [39], WET may require the HAP to consume a considerable amount of energy to provide sustainable operation for WDs. Due to its high cost, it is desired for HAP to consume less energy at the ET. In this paper, we formulate an optimization problem to minimize the total energy consumption of the HAP at the ET by jointly considering computation offloading, scheduling, and power allocation while guaranteeing the task completion and energy constraints. We also adopt the network and task models described in Section II.

Since the proposed WPMEC system consists of multiple WDs and hence multiple task call graphs, we consider the time horizon of the problem to be the number of required episodes to execute the subtasks of all WDs. We denote $\mathcal{U} \triangleq \{0, ..., U-1\}$ as the set of episodes, where $U \triangleq \max_{d\in\mathcal{D}} V_d$ indicates the number of episodes. Note that V_d is obtained from the call graph of the corresponding task. Therefore, the ET's cumulative energy consumption will be the sum of energy broadcasting over the time horizon. We define Γ^u, τ_0^u, and p^u as the length of episode $u \in \mathcal{U}$, WET time, and ET's power in episode $u \in \mathcal{U}$, respectively. We will then calculate the HAP's energy consumption at the ET as follows.

$$E^{\mathrm{ET}} = \sum_{u\in\mathcal{U}} \tau_0^u p^u = \boldsymbol{\tau}_0 \boldsymbol{p}^{\intercal}, \tag{31}$$

where $\boldsymbol{\tau}_0 = (\tau_0^u)_{u\in\mathcal{U}}$ and $\boldsymbol{p} = (p^u)_{u\in\mathcal{U}}$ denote the vectors of WET time and ET's transmission power, respectively.

B. Energy harvesting

In this subsection, we present the energy harvesting-related constraints. As described earlier, we assume sustainable operation of WDs by only relying on the harvested energy (i.e., battery-less network). Thus, the cumulative energy consumption of the devices on each episode must be no more than their harvested energy. The required energy of WDs can be divided into two parts: 1) Local computation energy, and 2) energy to send the task's data to the MEC server. We adopt the CPU's dynamic power equation [50], [31], [44] to model the local computation energy. We denote $\mathcal{D}^u$ as the set of WDs having a candidate subtask in episode $u \in \mathcal{U}$. We also denote $v_d^u \in \mathcal{V}_d$ as the candidate subtask of WD $d \in \mathcal{D}^u$ to be executed in episode

$u \in \mathcal{U}$ and τ_{2,v_d^u} as the length of time slot scheduled for local computation of this subtask. Therefore, the energy consumption for locally executing subtask $v_d^u \in \mathcal{V}_d$ is

$$E^{\mathrm{LO}}(\tau_{2,v_d^u}) = \zeta_d C_{v_d^u} \left(\frac{C_{v_d^u}}{\tau_{2,v_d^u}}\right)^2 = \frac{\zeta_d C_{v_d^u}^3}{\left(\tau_{2,v_d^u}\right)^2}, \quad \forall d \in \mathcal{D}^u, \forall u \in \mathcal{U}, \tag{32}$$

where $C_{v_d^u}$ is the number of CPU cycles for executing subtask $v_d^u \in \mathcal{V}_d$ of the corresponding task call graph. Moreover, ζ_d is the CPU's effective switched capacitance [44] of device $d \in \mathcal{D}^u$.

In order to derive the WD energy consumption in case of offloading, we first use the Shannon formula to calculate the transmission rate between the device and MEC server. Assume that $B_d, g_d, q_{v_d^u}$, and σ^2 denote the spectrum bandwidth, wireless channel gain of device $d \in \mathcal{D}^u$ for data transmission, transmission power for offloading subtask $v_d^u \in \mathcal{V}_d$, and noise power at the HAP, respectively. The transmission rate is then given as

$$r_{v_d^u} = B_d \log_2\left(1 + \frac{g_d q_{v_d^u}}{\sigma^2}\right), \quad \forall d \in \mathcal{D}^u, \forall u \in \mathcal{U}. \tag{33}$$

By utilizing (33), we will have the following equation for the offloading transmission power.

$$q_{v_d^u} = \frac{\sigma^2}{g_d}\left(2^{\frac{r_{v_d^u}}{B_d}} - 1\right), \quad \forall d \in \mathcal{D}^u, \forall u \in \mathcal{U}. \tag{34}$$

According to the task call graph in Fig. 10, the prerequisite of remotely executing a subtask $v_d^u \in \mathcal{V}_d$ for each device $d \in \mathcal{D}^u$ is to offload the required input data (i.e., the summation of weights of the subtask's input edges) to the MEC server. A time slot with length denoted as τ_{1,v_d^u} is scheduled for data transmission of subtask $v_d^u \in \mathcal{V}_d$ from device $d \in \mathcal{D}^u$ to the MEC server. We consider $\mathcal{I}_{v_d^u} = \{i \in \mathcal{U} | (v_d^i, v_d^u) \in \mathcal{E}_d\}$ as the set of episodes associated to those subtasks which have an output edge to vertex $v_d^u \in \mathcal{V}_d$. Therefore, by utilizing (34), we can calculate the offloading energy consumption for each subtask $v_d^u \in \mathcal{V}_d$ as follows.

$$E^{\mathrm{OF}}(\tau_{1,v_d^u}) = \tau_{1,v_d^u} q_{v_d^u} = \frac{\tau_{1,v_d^u}\sigma^2}{g_d}\left(2^{\frac{\sum_{i\in\mathcal{I}_{v_d^u}} N_{v_d^i, v_d^u}}{\tau_{1,v_d^u} B_d}} - 1\right), \quad \forall d \in \mathcal{D}^u, \forall u \in \mathcal{U}. \tag{35}$$

We now determine the amount of harvested energy. Let η_d and h_d denote the energy harvesting efficiency of WD $d \in \mathcal{D}$ and channel gain between the WD and ET, respectively. The harvested energy during each episode $u \in \mathcal{U}$, denoted as $E^{\mathrm{EH}}(\tau_0^u, p^u)$, is

$$E^{\mathrm{EH}}(\tau_0^u, p^u) = \eta_d h_d \tau_0^u p^u, \quad \forall u \in \mathcal{U}. \tag{36}$$

By denoting $P^{\mathrm{ET,max}}$ as the maximum power ET can transmit, we will have the following constraint on p^u.

$$0 \le p^u \le P^{\mathrm{ET,max}}, \quad \forall u \in \mathcal{U}. \tag{37}$$

We denote $x_{v_d^u}^{\mathrm{L}}$, $x_{v_d^u}^{\mathrm{M}}$, and $x_{v_d^u}^{\mathrm{C}}$ as binary variables that indicate the offloading decision of subtask v_d^u of

device $d \in \mathcal{D}^u$ in episode $u \in \mathcal{U}$. If $x^{\mathrm{L}}_{v^u_d} = 1$, the subtask is performed locally while $x^{\mathrm{M}}_{v^u_d} = 1$ indicates that the subtask is offloaded to the MEC server and $x^{\mathrm{C}}_{v^u_d} = 1$ means that it is offloaded to the cloud server. As stated earlier, we consider that the proposed WPMEC system is fully wireless powered. As a result, the consumed energy of each WD should be at most equal to its harvested energy. Therefore, by utilizing (32), (35), and (36), we have the following constraint.

$$E^{\mathrm{LO}}(\tau_{2,v^u_d})x^{\mathrm{L}}_{v^u_d} + E^{\mathrm{OF}}(\tau_{1,v^u_d})\left(x^{\mathrm{M}}_{v^u_d} + x^{\mathrm{C}}_{v^u_d}\right) \leq E^{\mathrm{EH}}(\tau^u_0, p^u), \quad \forall d \in \mathcal{D}^u, \forall u \in \mathcal{U}. \tag{38}$$

Note that the excessive harvested energy cannot be stored for future use.

C. Harvest-then-offload TDMA scheduling

We adopt the harvest-then-offload scheduling mechanism to enable WDs with half-duplex transmission to execute or offload their computation tasks. We also leverage a TDMA protocol for scheduling multiple WDs according to Fig. 9. We consider that an episode of WET and task execution for all WDs should not exceed the length of episode Γ^u. Therefore, by assuming $u \in \mathcal{U}$ as the episode, the following constraints should be satisfied.

$$\tau^u_0 + \sum_{d \in \mathcal{D}^u} \left(\tau_{1,v^u_d} + \tau_{2,v^u_d} + \tau_{3,v^u_d}\right) \leq \Gamma^u, \quad \forall u \in \mathcal{U}, \tag{39}$$

$$0 \leq \tau^u_0 \leq \Gamma^u, \quad \forall u \in \mathcal{U}, \tag{40}$$

where τ_{3,v^u_d} denotes the length of scheduled time for retrieving the results of subtask $v^u_d \in \mathcal{V}_d$ from the MEC server. It is worth noting that inequality (39) also enforces a hard deadline for each cycle of subtasks' execution in each episode. Thus, the total execution time of each task generated by WDs is constrained. This way, we can address the requirements of latency-sensitive tasks as well. Also, it can be concluded that Γ^u is a key factor in the optimal value of the ETEM problem. A longer episode (i.e., larger Γ^u) provides more time for execution of the tasks. In fact, this will enlarge the feasible region of the ETEM optimization problem and enhances the energy-efficiency of the algorithm.

As illustrated in Fig. 9, in the case of offloading to the MEC server (i.e., $x^{\mathrm{M}}_{v^u_d} = 1$) or cloud server (i.e., $x^{\mathrm{C}}_{v^u_d} = 1$), the time τ_{1,v^u_d} must be at most Γ^u. The figure also shows that if the subtask is executed locally (i.e., $x^{\mathrm{L}}_{v^u_d} = 1$ and $x^{\mathrm{M}}_{v^u_d} = x^{\mathrm{C}}_{v^u_d} = 0$), τ_{1,v^u_d} should be equal to zero. Therefore, the following constraint holds.

$$\tau^{\min}_{1,v^u_d} \leq \tau_{1,v^u_d} \leq \tau^{\max}_{1,v^u_d}, \quad \forall d \in \mathcal{D}^u, \forall u \in \mathcal{U}, \tag{41}$$

where $\tau^{\min}_{1,v^u_d} \triangleq \dfrac{\sum_{i \in \mathcal{I}_{v^u_d}} N_{v^i_d, v^u_d}}{B_d \log_2\left(1 + \frac{g_d P^{\mathrm{WD,max}}_d}{\sigma^2}\right)}\left(x^{\mathrm{M}}_{v^u_d} + x^{\mathrm{C}}_{v^u_d}\right)$ and $\tau^{\max}_{1,v^u_d} \triangleq \Gamma^u\left(x^{\mathrm{M}}_{v^u_d} + x^{\mathrm{C}}_{v^u_d}\right)$ are obtained using (33). In addition, $P^{\mathrm{WD,max}}_d$ denotes the maximum transmission power for WD $d \in \mathcal{D}$.

Constraints (38) and (41) address a tradeoff between the energy consumption and the transmission duration in the WDs. Above statements are also held for time τ_{3,v^u_d}. Thus, we introduce the following constraint:

$$\tau^{\min}_{3,v^u_d} \leq \tau_{3,v^u_d} \leq \tau^{\max}_{3,v^u_d}, \quad \forall d \in \mathcal{D}^u, \forall u \in \mathcal{U}, \tag{42}$$

where $\tau^{\min}_{3,v^u_d} \triangleq \frac{\sum_{j\in\mathcal{J}_{v^u_d}} N_{v^u_d,v^j_d}}{B_d \log_2\left(1+\frac{g_d P^{\mathrm{AP,max}}}{\sigma^2}\right)} \left(x^{\mathrm{M}}_{v^u_d} + x^{\mathrm{C}}_{v^u_d}\right)$, $\tau^{\max}_{3,v^u_d} \triangleq \Gamma^u \left(x^{\mathrm{M}}_{v^u_d} + x^{\mathrm{C}}_{v^u_d}\right)$, $\mathcal{J}_{v^u_d} = \{j \in \mathcal{U} | (v^u_d, v^j_d) \in \mathcal{E}_d\}$, and $P^{\mathrm{AP,max}}$ denotes the maximum AP's transmission power.

It is also worth noting that according to Fig. 9, in the case of local computation, there will not be any time required for data transmission between the MEC server and the corresponding WD. Thus, we have the following constraints for the first and last subtasks of each task call graph.

$$\tau_{1,v^0_d} = \tau_{3,v^0_d} = \tau_{1,v^{V_d-1}_d} = \tau_{3,v^{V_d-1}_d} = 0, \quad \forall d \in \mathcal{D}, \tag{43}$$

$$x^{\mathrm{M}}_{v^0_d} = x^{\mathrm{C}}_{v^0_d} = x^{\mathrm{M}}_{v^{V_d-1}_d} = x^{\mathrm{C}}_{v^{V_d-1}_d} = 0, \quad \forall d \in \mathcal{D}, \tag{44}$$

where $v^0_d, v^{V_d-1}_d \in \mathcal{V}_d$ indicate the first and last subtasks of device $d \in \mathcal{D}$, respectively.

We denote $F^{\mathrm{L,max}}_d$ as the maximum CPU frequency of device $d \in \mathcal{D}$. We also denote $F^{\mathrm{M,max}}$ as the maximum CPU frequency of the MEC server. We consider $C_{v^u_d}$ as the number of CPU cycles for executing a subtask $v^u_d \in \mathcal{V}_d$ on device $d \in \mathcal{D}^u$. Thus, τ_{2,v^u_d} will be at least $C_{v^u_d}/F^{\mathrm{L,max}}_d$ if the subtask is locally computed or $C_{v^u_d}/F^{\mathrm{M,max}}$ in the case of offloading to the MEC server. However, as the cloud servers mostly benefit from powerful processors, we omit the processing delay if the subtask is offloaded to the cloud server. Therefore, the scheduling time τ_{2,v^u_d} is affected by the server's load and also the round-trip and transmission delay from the MEC server to the cloud. We refer this as the response time which is denoted as $\mu_{v^u_d} \geq 0$ for each device $d \in \mathcal{D}^u$ and episode $u \in \mathcal{U}$. As a result, we have

$$\tau^{\min}_{2,v^u_d} \leq \tau_{2,v^u_d} \leq \Gamma^u, \quad \forall d \in \mathcal{D}^u, \forall u \in \mathcal{U}, \tag{45}$$

where $\tau^{\min}_{2,v^u_d} \triangleq \frac{C_{v^u_d}}{F^{\mathrm{L,max}}_d} x^{\mathrm{L}}_{v^u_d} + \frac{C_{v^u_d}}{F^{\mathrm{M,max}}} x^{\mathrm{M}}_{v^u_d} + \mu_{v^u_d} x^{\mathrm{C}}_{v^u_d}$. The response time $\mu_{v^u_d}$ is estimated by the MEC server at the beginning of each episode $u \in \mathcal{U}$ for each WD $d \in \mathcal{D}^u$.

D. ET's Energy Minimization (ETEM) Problem

To formulate the problem, we first define three offloading decision vectors for cloud, edge, and local computation as $\boldsymbol{x}^{\mathrm{C}} = \left(x^{\mathrm{C}}_{v^u_d}\right)_{\sum_{d\in\mathcal{D}} V_d}$, $\boldsymbol{x}^{\mathrm{M}} = \left(x^{\mathrm{M}}_{v^u_d}\right)_{\sum_{d\in\mathcal{D}} V_d}$, and $\boldsymbol{x}^{\mathrm{L}} = \left(x^{\mathrm{L}}_{v^u_d}\right)_{\sum_{d\in\mathcal{D}} V_d}$, respectively. We also define decision matrix $\boldsymbol{X} = \left(x_{t,v^u_d}\right)_{3\times\sum_{d\in\mathcal{D}} V_d} = \left(\boldsymbol{x}^{\mathrm{C}}; \boldsymbol{x}^{\mathrm{M}}; \boldsymbol{x}^{\mathrm{L}}\right)$, where each row corresponds to a tier number $t \in \{1,2,3\}$. We also define $\boldsymbol{T} = (\tau_{m,v^u_d})_{3\times\sum_{d\in\mathcal{D}} V_d}$ as the computation scheduling matrix with $m \in \{1,2,3\}$ representing the time frame number. We can now formulate the ETEM problem as follows.

$$\text{ETEM:} \underset{\boldsymbol{X},\boldsymbol{p},\boldsymbol{\tau}_0,\boldsymbol{T}}{\text{minimize}} \quad \boldsymbol{\tau}_0 \boldsymbol{p}^{\intercal} \tag{46a}$$

$$\text{subject to} \quad \text{constraints } (37)\text{–}(45), \tag{46b}$$

$$x^{\mathrm{L}}_{v^u_d} + x^{\mathrm{M}}_{v^u_d} + x^{\mathrm{C}}_{v^u_d} = 1, \quad \forall d \in \mathcal{D}^u, \forall u \in \mathcal{U}, \tag{46c}$$

$$x^{\mathrm{L}}_{v^u_d}, x^{\mathrm{M}}_{v^u_d}, x^{\mathrm{C}}_{v^u_d} \in \{0,1\}, \quad \forall d \in \mathcal{D}^u, \forall u \in \mathcal{U}. \tag{46d}$$

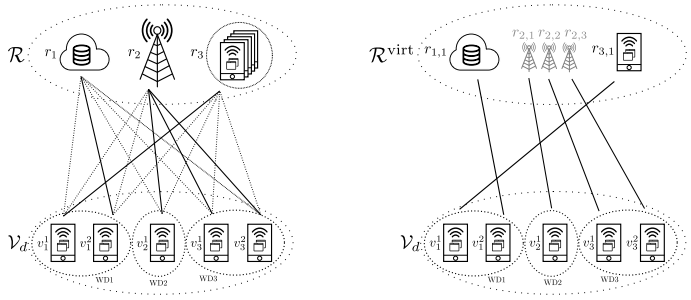
ETEM is a mixed-integer optimization problem which is intractable. In the next section, we will propose our algorithm called WiEnTM to efficiently solve the problem by leveraging bipartite graph matching and parallel processing [53].

X. Wireless Energy Transmission Minimization (WiEnTM) Algorithm

In this section, we propose our parallel WiEnTM algorithm to solve the ETEM problem using bipartite graph matching. We also analyze the time complexity of each parallel block of the algorithm.

A. Bipartite Graph Matching

Matching in graph theory can provide a tractable approach to combinatorial optimization problems [54]. Matching theory has been widely used for resource allocation in wireless networks where the users and resources are modeled as a bipartite graph [53], [54]. In this graph, users are connected to resources via edges. Each edge is associated with a weight which represents the cost of using the corresponding resource. A subset of edges associated with a feasible resource allocation is called a *matching*. The goal is to find the minimum weighted graph matching.



(a) (b)

Fig. 11: An example for matching bipartite graph of subtasks $\mathcal{V}_d$ and computation resources $\mathcal{R}$. Set $\mathcal{R}$ contains resources r_1 (i.e., cloud servers), r_2 (i.e., MEC servers), and r_3 (i.e., local devices). a) Many-to-one matching. b) Equivalent one-to-one matching by utilizing virtual resources $\mathcal{R}^{\text{virt}}$ where each r_{t,v_d^u} is related to tier number $t \in \{1, 2, 3\}$ and subtask v_d^u.

The bipartite graph in Fig. 11 provides an example of matching problem in the proposed WPMEC system. In Fig. 11a, each subtask $v_d^u \in \mathcal{V}_d$ of device $d \in \mathcal{D}^u$ in episode $u \in \mathcal{U}$ is matched with computation resources of set $\mathcal{R}$ which consists of cloud, MEC server, and local devices. It is important to note that device d can only be matched with itself or cloud/edge resources not other devices.

To leverage the matching algorithm, we need to calculate the cost of each matching between subtasks and computing resources. To accomplish this, we first transform the ETEM problem into an equivalent simplified problem. We then show that the transformed problem is decomposable, and the resultant decomposed problems are still equivalent to the ETEM problem.

For each matching, the binary decision variables $x^{\text{L}}_{v_d^u}$, $x^{\text{M}}_{v_d^u}$, and $x^{\text{C}}_{v_d^u}$ will have known values (e.g., matching subtask $v_d^u \in \mathcal{V}_d$ with the cloud resource will result in $x^{\text{C}}_{v_d^u} = 1$ and $x^{\text{L}}_{v_d^u} = x^{\text{M}}_{v_d^u} = 0$). However, the ETEM

problem is still non-convex even with eliminating binary variables. This is due to the term $\tau_0^u p^u$ which is neither a convex nor concave function of τ_0^u and p^u. For the sake of resolving this issue, we present the following theorem.

Theorem 1. *Given a matching (i.e., particular values of binary variables $x^L_{v_d^u}$, $x^M_{v_d^u}$, and $x^C_{v_d^u}$), ETEM can be transformed into a convex optimization problem.*

Proof: Refer to Appendix A. ■

We now transform the ETEM problem with the change of variable $\varepsilon^u = \tau_0^u p^u$ into a more tractable form. This helps us calculate the edge weights in the matching graph. In order to accomplish this, it is required to decompose the aforementioned problem into problems for each subtask $v_d^u \in \mathcal{V}_d$ of device $d \in \mathcal{D}$ in episode $u \in \mathcal{U}$. We first provide two important remarks as follows.

Remark. *The optimal value of variable τ_{3,v_d^u} will be*

$$\tau^*_{3,v_d^u} \triangleq \tau^{\min}_{3,v_d^u}, \quad \forall d \in \mathcal{D}^u, \forall u \in \mathcal{U}. \tag{47}$$

The above equation eliminates constraint (42).

Variable τ_{3,v_d^u} is only present in constraints (39) and (42). Since (39) forces a hard deadline on the scheduling times, it is desirable for τ_{3,v_d^u} to have the smallest possible value which is dictated by constraint (42). Therefore, at the optimal point of ETEM problem, 47 will be achieved.

Remark. *We can calculate the transmitted wireless energy during each episode $u \in \mathcal{U}$ as*

$$\varepsilon^u = \max_{d \in \mathcal{D}^u} \left\{ \frac{E^{LO}(\tau_{2,v_d^u}) x^L_{v_d^u}}{\eta_d h_d} + \frac{E^{OF}(\tau_{1,v_d^u}) \left(x^M_{v_d^u} + x^C_{v_d^u} \right)}{\eta_d h_d} \right\}, \tag{48}$$

where $v_d^u \in \mathcal{V}_d$ is the candidate subtask of device $d \in \mathcal{D}^u$ that is being executed in episode $u \in \mathcal{U}$.

We know that the objective of the ETEM problem is to minimize the wireless energy transmission. This minimum value must meet the energy requirements of all WDs. Therefore, it is the maximum energy consumption among WDs in each episode $u \in \mathcal{U}$. Using the above remark, we can rewrite the objective function and eliminate constraint (38).

Using (48), we can transform ETEM into a minimax optimization problem but it may still be difficult to solve when there are large number of WDs. We know that constraint (39) is creating a dependency between subtasks. By rewriting this constraint for each subtask, we will be able to reduce this minimax optimization problem into a number of independent problems. As a result, the ETEM problem will convert into a minisum optimization problem [55] which is easier to solve.

In order to eliminate the aforementioned dependency in constraint (39), we first consider a given value for the WET time τ_0^u which satisfies constraint (40). Thus, the ET's transmitted energy will only depend on p^u. There are iterative methods to calculate the appropriate value of τ_0^u such as the one proposed in [50]. By decomposing the summation on the left-hand side of (39), this constraint can be rewritten for each subtask $v_d^u \in \mathcal{V}_d$. We first introduce a new parameter $0 \leq \beta_{v_d^u} \leq |\mathcal{D}^u|$ which reflects the priority,

computation complexity, or input/output size of each subtask $v_d^u \in \mathcal{V}_d$. We then introduce the following constraint.

$$\tau_{1,v_d^u} + \tau_{2,v_d^u} \leq \frac{\beta_{v_d^u}}{|\mathcal{D}^u|}\left(\Gamma^u - \tau_0^u\right) - \tau^*_{3,v_d^u}. \tag{49}$$

We now reconstruct (39) using (49). To do so, we accumulate each side of (49) over all WDs in $\mathcal{D}$ and choose $\beta_{v_d^u}$ such that

$$\sum_{d\in\mathcal{D}^u} \beta_{v_d^u} \leq |\mathcal{D}^u|, \quad \forall u \in \mathcal{U}. \tag{50}$$

It can be easily shown that when the above condition holds, constraint (49) ensures that constraint (39) is met. Moreover, in the case of equality, constraints (39) and (49) will be equivalent. We will analyze the effect of $\beta_{v_d^u}$ on the optimal solution in Section V.

For ease of exposition, we now define the following two functions for each $v_d^u \in \mathcal{V}_d$ assuming $x^{\mathrm{L}}_{v_d^u}$, $x^{\mathrm{M}}_{v_d^u}$, and $x^{\mathrm{C}}_{v_d^u}$ are given.

$$\begin{aligned} e_1(\tau_{1,v_d^u}) &\triangleq \frac{E^{\mathrm{OF}}_{v_d^u}\left(x^{\mathrm{M}}_{v_d^u} + x^{\mathrm{C}}_{v_d^u}\right)}{\eta_d h_d} \\ &= \frac{\tau_{1,v_d^u}\left(x^{\mathrm{M}}_{v_d^u} + x^{\mathrm{C}}_{v_d^u}\right)}{\sigma^{-2} g_d \eta_d h_d}\left(2^{\frac{\sum_{i\in\mathcal{I}_{v_d^u}} N_{v_d^i,v_d^u}}{\tau_{1,v_d^u} B_d}} - 1\right), \end{aligned} \tag{51}$$

$$e_2(\tau_{2,v_d^u}) \triangleq \frac{E^{\mathrm{LO}}_{v_d} x^{v_d}_{\mathrm{L}}}{\eta_d h_d} = \frac{\zeta_d C^3_{v_d^u} x^{\mathrm{L}}_{v_d^u}}{\eta_d h_d \left(\tau_{2,v_d^u}\right)^2}. \tag{52}$$

As WDs may require different amounts of power, the problem for each subtask may find different values for p^u. Thus, we define $p_{v_d^u}$ as the required ET's power for subtask $v_d^u \in \mathcal{V}_d$. Let $p^*_{v_d^u}$ and p^{u^*} denote the optimal values of $p_{v_d^u}$ and p^u, respectively. We know that p^u must be enough to power all WDs. As a result, we can calculate p^{u^*} by selecting the maximum $p^*_{v_d^u}$ as follows.

$$p^{u^*} = \max_{d\in\mathcal{D}^u} p^*_{v_d^u}, \quad \forall u \in \mathcal{U}. \tag{53}$$

In the following remark, we will present an equality constraint for calculating $p^*_{v_d^u}$.

Remark. *The following constraint is equivalent to (38) when τ_0^u is given.*

$$e_1\left(\tau_{1,v_d^u}\right) + e_2\left(\tau_{2,v_d^u}\right) - p_{v_d^u}\tau_0^u = 0. \tag{54}$$

Based on the definition of $p_{v_d^u}$, $p_{v_d^u}\tau_0^u$ will be the energy required by each subtask $v_d^u \in \mathcal{V}_d$. We know that $e_1\left(\tau_{1,v_d^u}\right) + e_2\left(\tau_{2,v_d^u}\right)$ is the total energy consumption for local computation or offloading of subtask $v_d^u \in \mathcal{V}_d$. In the optimal point of ETEM problem, there should not be any excessive WET. As a result, equality (54) holds.

We now utilize (49) and (51)–(54) to reformulate the ETEM problem into a simplified ETEM problem

(S-ETEM) with given offloading decisions $x^{\mathrm{L}}_{v^u_d}$, $x^{\mathrm{M}}_{v^u_d}$, and $x^{\mathrm{C}}_{v^u_d}$.

$$\text{S-ETEM:} \underset{\boldsymbol{p}^{\mathrm{v}},\boldsymbol{\tau}_1,\boldsymbol{\tau}_2}{\text{minimize}} \quad \sum_{u\in\mathcal{U}}\sum_{d\in\mathcal{D}^u} e_1\left(\tau_{1,v^u_d}\right)+e_2\left(\tau_{2,v^u_d}\right) \tag{55a}$$

$$\text{subject to} \quad 0\le p_{v^u_d}\le P^{\mathrm{ET,max}}, \forall d\in\mathcal{D}^u, \forall u\in\mathcal{U}, \tag{55b}$$

$$\text{constraints } (41),(45), \tag{55c}$$

$$\text{constraints } (49),(54), \quad \forall d\in\mathcal{D}^u, \forall u\in\mathcal{U}. \tag{55d}$$

where $\boldsymbol{p}^{\mathrm{v}}\triangleq(p_{v^u_d})_{\sum_{d\in\mathcal{D}}V_d}$, $\boldsymbol{\tau}_1\triangleq(\tau_{1,v^u_d})_{\sum_{d\in\mathcal{D}}V_d}$, and $\boldsymbol{\tau}_2\triangleq(\tau_{2,v^u_d})_{\sum_{d\in\mathcal{D}}V_d}$.

The above problem is a convex optimization problem. The proof is similar to the proof of Theorem 1 and is omitted due to space limit. It is also worth noting that for a given $\beta_{v^u_d}$, by solving S-ETEM, we can achieve the optimal solution of the ETEM problem. In other words, by determining the optimal value of $\beta_{v^u_d}$ for each $d\in\mathcal{D}^u$ and $u\in\mathcal{U}$, the optimal ET's transmitted energy can be obtained and there would not be any approximation.

The S-ETEM problem can be further decomposed as its objective function is separable. Thus, we can have problems specific to each subtask $v^u_d\in\mathcal{V}_d$. We regard these problems as decomposed S-ETEM (DS-ETEM). We can then calculate the weight of each matching as the solution of DS-ETEM which will be individually solved for each subtask $v^u_d\in\mathcal{V}_d$ of WD $d\in\mathcal{D}^u$ and episode $u\in\mathcal{U}$.

$$\text{DS-ETEM:} \underset{p_{v^u_d},\tau_{1,v^u_d},\tau_{2,v^u_d}}{\text{minimize}} \quad e_1\left(\tau_{1,v^u_d}\right)+e_2\left(\tau_{2,v^u_d}\right) \tag{56a}$$

$$\text{subject to} \quad \tau^{\min}_{1,v^u_d}\le\tau_{1,v^u_d}\le\tau^{\max}_{1,v^u_d}, \tag{56b}$$

$$\tau^{\min}_{2,v^u_d}\le\tau_{2,v^u_d}\le\Gamma^u, \tag{56c}$$

$$0\le p_{v^u_d}\le P^{\mathrm{ET,max}}, \tag{56d}$$

$$\text{constraints } (49),(54). \tag{56e}$$

We continue by presenting *matching rules* [53] which are constraints on selecting optimal matching to ensure it is a feasible solution of ETEM problem. The first matching rule is that each WD can only be matched to one of the resources corresponding to the equality constraint (46c) in the ETEM problem. Note that each resource can be assigned to multiple WDs. Therefore, we call this matching as a *many-to-one* matching. For a specific matching, if the constraints of DS-ETEM problem are satisfied for all subtasks, the matching (i.e., a subset of edges in the matching graph) is considered as a feasible matching [53]. We exclude the edges corresponding to infeasible matchings from the problem by setting their weights to infinity. Thus, these edges will never be selected in the minimum weight matching as they have infinite edge weights.

Since solving many-to-one matching problems are difficult [53], we further convert this matching into a *one-to-one* matching similar to Fig. 11b. In order to perform this transformation, we first introduce the concept of *virtual resource* as inspired by [53]. Each virtual resource corresponds to an identical copy of

a resource associated to each subtask. The matching between a subtask and a virtual resource is called a *virtual matching*. From Fig. 11a, it can be observed that subtasks v_2^1, v_3^1, and v_3^2 of WD2 and WD3 are connected to resource r_2. We replace this resource with three virtual resources $r_{2,1}$, $r_{2,2}$, and $r_{2,3}$. This makes the many-to-one matching of Fig. 11a, an equivalent one-to-one matching.

In the next subsection, we propose an algorithm to solve this one-to-one matching.

B. Proposed Algorithm

In this subsection, we describe our parallel algorithm named WiEnTM to solve the non-convex ETEM problem using the aforementioned minimum weighted graph matching. The algorithm consists of three stages. In the first stage, the weight of all possible matchings are calculated in parallel. We regard this stage as *matching graph weight calculation* (MGW) algorithm. Note that in order to calculate the weights of subtasks, there is no need to actually execute them. The only required information is the size of input and output of each subtask which is already known from the task call graph. Thus, Lines 4 to 17 of the MGW algorithm can be run simultaneously. The next stage is intended to find the optimal matchings based on these weights. In the final stage, the algorithm utilizes the optimal virtual matching to determine the desired optimal offloading decisions and scheduling.

Let $\mathcal{R}^{\text{virt}}$ denote the set of virtual resources. We consider $r_{t,v_d^u}^{\text{virt}} \in \mathcal{R}^{\text{virt}}$ as the virtual resource for tier $t \in \{1,2,3\}$ and subtask $v_d^u \in \mathcal{V}_d$. We then define the matrix $\boldsymbol{W}^{\text{virt}} = \left(w_{v_d^u, r_{t,v_d^u}}^{\text{virt}}\right)_{\left(\sum_{d\in\mathcal{D}}(V_d-2)\right)\times|\mathcal{R}^{\text{virt}}|}$ whose entries are the edge weights of the one-to-one matching graph initialized with infinity for all elements. We similarly define matrices $\boldsymbol{T}_i^{\text{virt}} = \left(\tau_{i,v_d^u}^{\text{virt}}\right)_{\left(\sum_{d\in\mathcal{D}}(V_d-2)\right)\times|\mathcal{R}^{\text{virt}}|}, i \in \{1,2,3\}$ and $\boldsymbol{P}^{\text{virt}} = \left(p_{v_d^u}^{\text{virt}}\right)_{\left(\sum_{d\in\mathcal{D}}(V_d-2)\right)\times|\mathcal{R}^{\text{virt}}|}$ as auxiliary matrices to store the optimal scheduling times and required powers for different matchings, respectively. As the first and last subtasks are always performed locally, we exclude them from the matching algorithm. Thus, these matrices as well as the power vector will all have $\sum_{d\in\mathcal{D}}(V_d-2)$ rows which corresponds to all subtasks except the first and last ones.

We now explain the MGW algorithm. The first stage of this algorithm (Lines 5-19) requires $|\mathcal{R}^{\text{virt}}| \times \left(\sum_{d\in\mathcal{D}}(V_d-2)\right)$ parallel executions as shown by a block inside *cobegin* and *coend* keywords. We regard this block as a *kernel*. Each kernel has two input parameters λ_{id} and λ_{count} in the form of $\lambda_{\text{id}}(0:\lambda_{\text{count}})$ where λ_{id} is the identifier of each kernel and is given during the execution. In addition, λ_{count} is the total number of required executions of the parallel block. In each kernel, the algorithm solves DS-ETEM for virtual resource $r_{t,v_d^u}^{\text{virt}} \in \mathcal{R}^{\text{virt}}$ of subtask $v_d^u \in \mathcal{V}_d$ (Lines 11 and 16) using the techniques such as sequential quadratic programming [56]. Note that this is a convex problem with only three variables. For the subtasks at both ends, the algorithm enforces the local computation by setting $x_{v_d^u}^{\text{L}^*} = 1$ and $x_{v_d^u}^{\text{M}^*} = x_{v_d^u}^{\text{C}^*} = 0$ (Lines 14-15). It then similarly solves DS-ETEM to obtain optimal values of τ_{0,v_d^u} and τ_{2,v_d^u} denoted by $\tau_{0,v_d^u}^*$ and $\tau_{2,v_d^u}^*$ (Line 16). Moreover, if $\tau_{j,v_d^u}^{\min} > \tau_{j,v_d^u}^{\max}$ for $j \in \{1,2\}$, the algorithm skips solving DS-ETEM. The next kernel (Lines 20-25) checks for matching feasibility. In the case of having infinite weights for all virtual resources of subtask $v_d^u \in \mathcal{V}_d$, the algorithm stops the executions and indicates the infeasibility. Therefore, checking this condition for each subtask requires $\sum_{d\in\mathcal{D}}(V_d-2)$ parallel execution.

Next, we analyze the complexity of the MGW algorithm. Problem DS-ETEM is formulated for each subtask $v_d^u \in \mathcal{V}_d$. Thus, it has a time complexity of $O(1)$. The if-statement in the last kernel can also be

Algorithm 1: MGW algorithm

Input: $\mathcal{D}, \mathcal{D}^u, \mathcal{R}^{\text{virt}}, \mathcal{V}_d, \mathcal{U}, \Gamma^u, V_d, \tau^{\min}_{1,v^u_d}, \tau^{\min}_{2,v^u_d}, \tau^{\min}_{3,v^u_d}, B_d, \sigma,$
$\mu_{v^u_d}, g_d, \eta_d, h_d, \beta_{v^u_d}, \tau^u_0, F^{\text{L,max}}_d, F^{\text{M,max}}, \forall r^{\text{virt}}_{t,v^u_d} \in \mathcal{R}^{\text{virt}}, t \in \{1,2,3\}, v^u_d \in \mathcal{V}_d, d \in \mathcal{D}, u \in \mathcal{U}$

Output: Virtual matching graph weights $\boldsymbol{W}^{\text{virt}}$, virtual scheduling times $\boldsymbol{T}^{\text{virt}}_1$, $\boldsymbol{T}^{\text{virt}}_2$, $\boldsymbol{T}^{\text{virt}}_3$, and virtual power values $\boldsymbol{P}^{\text{virt}}$

1 $\boldsymbol{W}^{\text{virt}} \leftarrow (+\infty)_{\left(\sum_{d\in\mathcal{D}}(V_d-2)\right)\times|\mathcal{R}^{\text{virt}}|}$;
2 $\boldsymbol{T}^{\text{virt}}_1$, $\boldsymbol{T}^{\text{virt}}_2$, $\boldsymbol{T}^{\text{virt}}_3 \leftarrow (+\infty)_{\left(\sum_{d\in\mathcal{D}}(V_d-2)\right)\times|\mathcal{R}^{\text{virt}}|}$;
3 $\boldsymbol{P}^{\text{virt}} \leftarrow (+\infty)_{\left(\sum_{d\in\mathcal{D}}(V_d-2)\right)\times|\mathcal{R}^{\text{virt}}|}$;
4 **cobegin** $r_{t,v^u_d}(0 : |\mathcal{R}^{virt}|)$
5 Assign $x^{\text{L}}_{v^u_d}$, $x^{\text{M}}_{v^u_d}$, and $x^{\text{C}}_{v^u_d}$ based on $r^{\text{virt}}_{t,v^u_d}$;
6 /* Eqs. (47), (51), and (52) */
7 **if** $\tau^{\min}_{j,v^u_d} \le \tau^{\max}_{j,v^u_d}, j \in \{1,2\}$ **then**
8 **if** $u \notin \{0, V_d - 1\}$ **then**
9 $\tau^{\text{virt}^*}_{3,v^u_d} \leftarrow \tau^*_{3,v^u_d}$; // Eq. (47)
10 Solve DS-ETEM for $p^*_{v^u_d}, \tau^*_{1,v^u_d}, \tau^*_{2,v^u_d}$ and assign to $p^{\text{virt}^*}_{v^u_d}, \tau^{\text{virt}^*}_{1,v^u_d}, \tau^{\text{virt}^*}_{2,v^u_d}$;
11 $w^{\text{virt}}_{v^u_d, r_{t,v^u_d}} \leftarrow e_1\left(\tau^*_{1,v^u_d}\right) + e_2\left(\tau^*_{2,v^u_d}\right)$;
12 **else**
13 $x^{\text{L}^*}_{v^u_d} \leftarrow 1$; $x^{\text{M}^*}_{v^u_d}, x^{\text{C}^*}_{v^u_d} \leftarrow 0$;
14 $\tau^*_{1,v^u_d}, \tau^*_{3,v^u_d} \leftarrow 0$;
15 Solve DS-ETEM for $p^*_{v^u_d}, \tau^*_{2,v^u_d}$ given $x^{\text{L}^*}_{v^u_d}, x^{\text{M}^*}_{v^u_d}, x^{\text{C}^*}_{v^u_d}$;
16 **end**
17 **end**
18 **coend**
19 **cobegin** $v^u_d\left(0 : \sum_{d\in\mathcal{D}}(V_d - 2)\right)$
20 **if** $w^{virt}_{v^u_d, r_{t,v^u_d}} = +\infty, \forall r^{virt}_{t,v^u_d} \in \mathcal{R}^{virt}$ **then**
21 **return** infeasible;
22 **end**
23 **coend**

executed with the time complexity of $O(1)$. This is due to having only three resource options for a given subtask. In order to analyze the time complexity of the parallel algorithm, a parameter known as *depth* [57] is defined as the time complexity of the longest series of computations that are required to be executed sequentially. Another parameter known as *work* [57] denotes the total number of primitive works executed by the parallel processor. Therefore, the first kernel of the MGW algorithm has the depth of $O(1)$ and work of $O(UD)$ where U and D are the number of episodes and WDs, respectively, and UD is the total number of virtual resources. In addition, the second kernel of MGW has the depth of $O(1)$ and work of $O(UD)$.

Algorithm 2 presents the WiEnTM algorithm. The algorithm first exploits the MGW algorithm in Line 1 to find the matching weights and related matrices. Lines 3 to 6 are intended to find the index in the weight matrix which offers the minimum weight for subtask $v^u_d \in \mathcal{V}_d$ to obtain the corresponding optimal

Algorithm 2: WiEnTM algorithm

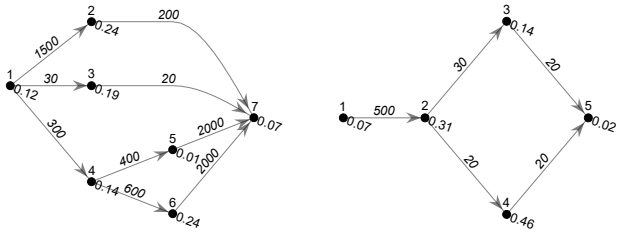
Input: $\mathcal{D}, \mathcal{D}^u, \mathcal{R}^{\text{virt}}, \mathcal{V}_d, \mathcal{U}, \Gamma^u, V_d, \tau^{\min}_{1,v^u_d}, \tau^{\min}_{2,v^u_d}, \tau^{\min}_{3,v^u_d}, B_d, \sigma,$
$\mu_{v^u_d}, g_d, \eta_d, h_d, \beta_{v^u_d}, \tau^u_0, F^{\text{L,max}}_d, F^{\text{M,max}}, \forall r^{\text{virt}}_{t,v^u_d} \in \mathcal{R}^{\text{virt}}, t \in \{1,2,3\}, v^u_d \in \mathcal{V}_d, d \in \mathcal{D}, u \in \mathcal{U}$

Output: Offloading decisions $\boldsymbol{X}^*$, WET powers $\boldsymbol{p}^*$, and scheduling times $\boldsymbol{T}^*$

1 $\boldsymbol{X}^{\text{virt}^*} \leftarrow (0)_{\left(\sum_{d\in\mathcal{D}}(V_d-2)\right)\times|\mathcal{R}^{\text{virt}}|}$;
2 Run MGW algorithm;
3 **cobegin** $v^u_d \left(0 : \sum_{d\in\mathcal{D}}(V_d-2)\right)$
4 $\quad r_{\min} \leftarrow \text{argmin}_{r_{t,v^u_d}\in\mathcal{R}^{\text{virt}}}\left(w_{v^u_d, r_{t,v^u_d}}\right)$;
5 $\quad x^{\text{virt}^*}_{v^u_d, r_{\min}} \leftarrow 1$;
6 **coend**
7 **for** $u \in \mathcal{U}$ **do**
8 $\quad$ **cobegin** $d\,(0 : |\mathcal{D}^u|)$
9 $\quad\quad$ **if** $u \notin \{0, V_d - 1\}$ **then**
10 $\quad\quad\quad$ Find the non-zero element in $\boldsymbol{x}^{\text{virt}^*}_{v^u_d}$ row;
11 $\quad\quad\quad$ Set $x^{\text{L}^*}_{v^u_d}, x^{\text{M}^*}_{v^u_d}, x^{\text{C}^*}_{v^u_d}$, and $p^*_{v^u_d}, \tau^*_{1,v^u_d}, \tau^*_{2,v^u_d}, \tau^*_{3,v^u_d}$ based on the value of tier number t determined in Line 10 and the outputs of MGW in Line 2;
12 $\quad\quad$ **end**
13 $\quad$ **coend**
14 $\quad p^{u^*} \leftarrow \max_{d\in\mathcal{D}^u} p^*_{v^u_d}$;
15 **end**

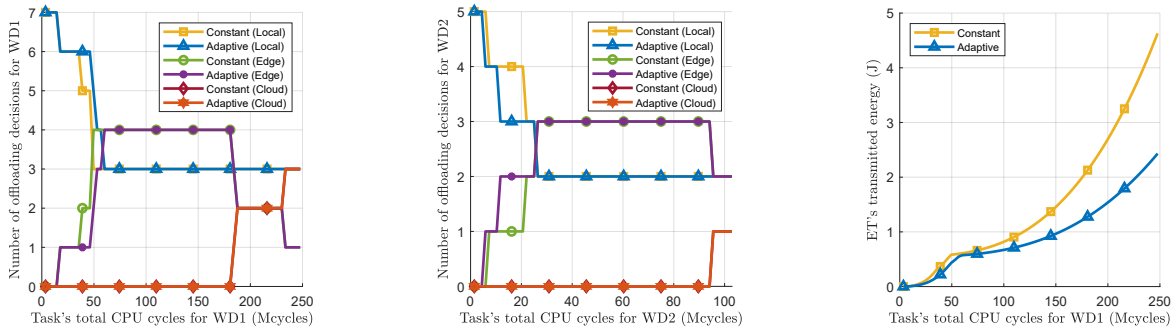
virtual matching. In this kernel, we consider $r_{\min}$ as a temporary variable to store the minimum-weighted virtual resource. Let $\boldsymbol{X}^{\text{virt}^*} = \left(x^{\text{virt}^*}_{v^u_d, r_{t,v^u_d}}\right)_{\left(\sum_{d\in\mathcal{D}}(V_d-2)\right)\times|\mathcal{R}^{\text{virt}}|}$ denote the optimal virtual matching. In Line 5, we set the corresponding position in $\boldsymbol{X}^{\text{virt}^*}$ matrix. The last loop of Algorithm 2 from Line 7 to Line 15 aims to map $\boldsymbol{X}^{\text{virt}^*}$ to the corresponding optimal decision matrix which we denote as $\boldsymbol{X}^*$. In addition, we also determine the optimal values of $\boldsymbol{p}$ and $\boldsymbol{T}$ denoted as $\boldsymbol{p}^*$ and $\boldsymbol{T}^*$, respectively. This is done according to the matching graph and the output of the previous stage. We know that the non-zero element in each row of $\boldsymbol{X}^{\text{virt}^*}$ corresponds to the optimal matching of a subtask. We can obtain the optimal values of $\boldsymbol{X}^*$ and $\boldsymbol{T}^*$ by executing Line 11. In order to calculate the optimal vector of WET powers $\boldsymbol{p}^*$, we need to find the maximum value of $p^*_{v^u_d}$ over all WDs according to (53) (Line 14).

We further analyze the complexity of WiEnTM algorithm. The argmin operation (Line 4) is only executed over three virtual resources of the subtask. Therefore, the depth and work of the block in Lines 3-6 are $O(1)$ and $O(UD)$, respectively. In the next kernel, finding the non-zero element in each row only requires three comparisons. As a result, the primitive in this block has the time complexity of $O(1)$. Therefore, the depth of this kernel over WDs has the complexity of $O(1)$. It also has the work with complexity $O(|\mathcal{D}^u|) = O(D)$. In Line 14, we execute a $\max$ operation. In general, maximization is a traditional reduction operation which can be parallelized in a tree-like structure with the depth of $O(\log_2 D)$ and the work of $O(2D)$. Therefore, the last stage of the algorithm has the depth of $O(U \log_2 D)$ and the work of $O(UD)$.



(a) (b)

Fig. 12: Task call graph of a) WD1 and b) WD2. The numbers on each node represent the sequence in which the corresponding subtask is executed as well as the normalized computation cycle. Edge weights are input/output data sizes of subtasks in Kbits.



(a) (b) (c)

Fig. 13: Impact of increasing the task's total number of CPU cycles (i.e., $\sum_{v_d \in \mathcal{V}_d} C_{v_d}$) on a) The optimal number of offloading decisions of WD1, b) the optimal number of offloading decisions of WD2, and c) ET's transmitted energy, for constant and adaptive $\beta_{v_d^u}$ approaches.

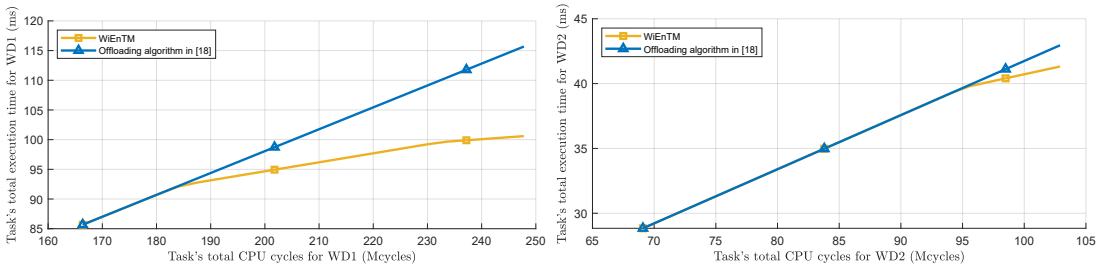
As the value of U represents the maximum number of subtasks among devices, it does not scale with the size of network D. Therefore, we can consider it as a constant in analyzing time complexity. To summarize, by considering this fact, the depth and work of our parallel WiEnTM algorithm are $O(\log_2 D)$ and $O(D)$, respectively.

XI. Performance Evaluation

In this section, we investigate the performance of WiEnTM algorithm by evaluating the computation time of tasks and the ET's energy transmission. We further compare our proposed algorithm with an existing work in [44].

A. Simulation Setup

We consider two devices WD1 and WD2 with task call graphs shown in Fig. 12a and Fig. 12b. These figures show the dependencies in the task call graph of each WD. An arrow leaving node i toward j indicates that task j is dependent on task i. The size of subtasks to be transferred between the devices and cloud/MEC server is shown in Kbits as the weight of edges. The normalized computation cycles of each subtask is



(a) (b)

Fig. 14: The total computation time of our three-tier WPMEC and the scheme proposed in [44] for different task's total CPU cycles (i.e., $\sum_{v_d \in \mathcal{V}_d} C_{v_d}$) for a) WD1, and b) WD2.

also shown next to the vertices. This value reflects the required computation cycles for each subtask (i.e., $C_{v_d^u}, \forall d \in \mathcal{D}^u, \forall u \in \mathcal{U}$) divided by the task's total computation cycle (i.e., $\sum_{u \in \mathcal{U}} C_{v_d^u}, \forall d \in \mathcal{D}$). Constant parameters are set as $\zeta_1 = \zeta_2 = 10^{-28}$F (i.e., CPU's effective switched capacitance) [50], $\eta_1 = \eta_2 = 0.51$ (i.e., energy harvesting efficiency) [50], $\sigma^2 = 10^{-9}$W (i.e., noise power) [50], $\Gamma_1 = 0.8$s, $\Gamma_2 = \Gamma_3 = \Gamma_4 = \Gamma_5 = 0.6$s, $\Gamma_6 = 0.3$s, $\Gamma_7 = 0.4$s (i.e., length of each episode), $P^{\text{ET,max}} = 50$W, $P^{\text{AP,max}} = 25$W, and $P_1^{\text{WD,max}} = P_2^{\text{WD,max}} = 25$mW. We set the cloud response times $\mu_{v_d^u} = 20\text{ms}, \forall d \in \mathcal{D}^u, \forall u \in \mathcal{U}$ as inspired by [58]. We also set $B_1 = B_2 = 50$MHz, $F_1^{\text{L,max}} = F_2^{\text{L,max}} = 160$MHz, and $F^{\text{M,max}} = 2.2$GHz. We adopt the simplified path loss model [59] for wireless channel gains. Without loss of generality, we set $g_d = h_d = -5\text{dB}, \forall d \in \mathcal{D}$.

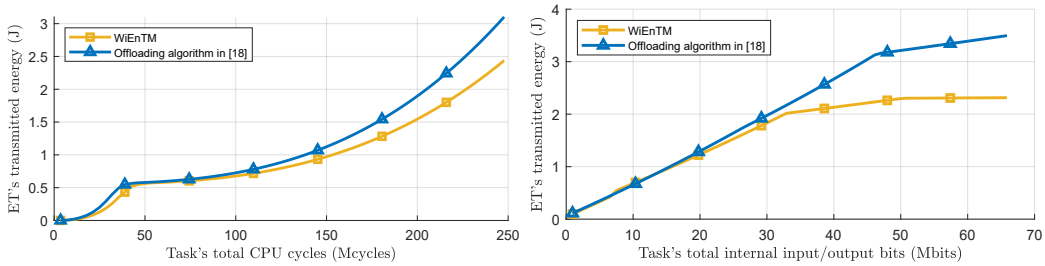
In order to analyze the effect of $\beta_{v_d^u}$ on the optimal solution, we consider two cases for $\beta_{v_d^u}$: 1) Constant with $\beta_{v_d^u} = 1, \forall v_d^u \in \mathcal{V}_d$, 2) adaptive $\beta_{v_d^u}$ with $0 \leq k \leq 1$ given as follows.

$$\begin{aligned}\beta_{v_d^u} =& \frac{k C_{v_d^u}}{\sum_{d' \in \mathcal{D}^u} C_{v_{d'}^u}} |\mathcal{D}^u| + \\ & \frac{(1-k) \sum_{v_d^i \in \mathcal{I}_{v_{d'}^u}} N_{v_{d'}^i, v_d^u}}{\sum_{d' \in \mathcal{D}^u} \sum_{v_{d'}^i \in \mathcal{I}_{v_{d'}^u}} N_{v_{d'}^i, v_{d'}^u}} |\mathcal{D}^u|, \quad \forall d \in \mathcal{D}^u, \forall u \in \mathcal{U},\end{aligned} \tag{57}$$

where we consider $k = 0.8$. Note that the above value is an example of adaptive $\beta_{v_d^u}$ and determining it's optimal value is beyond the scope of this work. [3]

We now describe the intuition behind the adaptive $\beta_{v_d^u}$ value given in (57). According to (31) and (32), it can be observed that increasing the number of CPU cycles and input/output data bits of each subtask will increase the energy required by the subtask. Constraint (49) depicts that increasing $\beta_{v_d^u}$ will expand the feasible space for τ_{1,v_d^u} and τ_{2,v_d^u} which will decrease the energy consumption. Therefore, subtasks with high number of CPU cycles and high amount of input/output data bits are the bottleneck of energy consumption. As a result, modeling $\beta_{v_d^u}$ based on a weighted sum of normalized number of CPU cycles and input/output bits should decrease the energy consumption compared to assigning equal $\beta_{v_d^u}$ to all subtasks.

[3] For the case of non-optimal $\beta_{v_d^u}$, one needs to analyze the approximation ratio of the proposed algorithm. An approach similar to [50] can be followed. However, we leave this extension as future work.



(a) (b)

Fig. 15: Comparison of ET's transmitted energy between WiEnTM and the algorithm proposed in [44] for different a) task's CPU cycles (i.e., $\sum_{v_d \in \mathcal{V}_d} C_{v_d}$) and, b) task's internal input/output bits (i.e., $\sum_{v_d^i, v_d^j \in \mathcal{V}_d} N_{v_d^i, v_d^j}$).

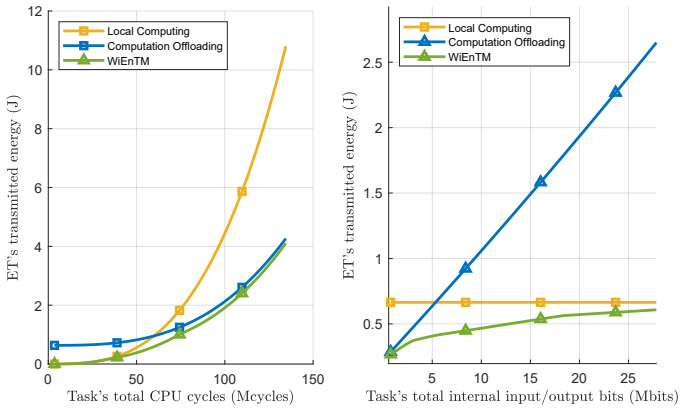
B. Simulation Results

Figs. 13a and 13b show the transition between different optimal offloading decisions as we increase the task's CPU cycles (i.e., $\sum_{u \in \mathcal{U}} C_{v_d^u}, \forall d \in \mathcal{D}$) for WD1 and WD2, respectively. In both of these figures, we illustrate the number of offloading decisions for the local, edge, and cloud computing. Note that for each WD, the sum of offloading decisions is the number of subtasks. It can be observed that as we increase the task CPU cycles, the offloading decisions gradually transit from the local to the edge and finally to the cloud. Additionally, it is shown that for WD1 which has a larger number of task CPU cycles, the system relies more on the cloud server than WD2. Fig. 13c shows the optimal ET's energy transmission. From Fig. 13c, it can be perceived that adaptive technique offers up to 47% less energy consumption in comparison with the constant $\beta_{v_d^u}$ approach. It means that setting $\beta_{v_d^u}$ according to the number of CPU cycles and input/output sizes greatly enhances the performance comparing to the constant approach.

By considering adaptive $\beta_{v_d^u}$ given in (57), we also compare our proposed three-tier WPMEC system with the scheme proposed in [44]. Figs. 14a and 14b illustrate the task's execution time for WD1 and WD2, respectively. The results show that by using WiEnTM, up to 13% overall less total computation time can be obtained for WD1 in comparison to [44]. This is because WD1 has a computationally intensive task. However, a lower improvement is achieved for WD2 as it has a lighter task which is mostly executed locally by the device or on the MEC server.

We further investigate the performance of our proposed WiEnTM algorithm by evaluating the ET's transmitted energy in comparison to that of [44]. Figs. 15a and 15b illustrate the effect of the task's CPU cycles and the task size (i.e., the number of input/output bits), respectively. The results show that by using WiEnTM, the ET consumes up to 34% less energy compared to the algorithm in [44].

We now consider 200 WDs and investigate the ET transmission energy required to enable WDs execute their tasks. We study three different policies: 1) Local computing, 3) computation offloading, and 3) WiEnTM algorithm. We assume that among these devices, 100 WDs have a task with the call graph shown in Fig. 12a and the rest with the task call graph of Fig. 12b. In the local computing case, all WDs harvest the energy to execute their subtasks locally while in the computation offloading case, the energy is used to offload all the subtasks. Fig. 16a shows the ET's energy transmission when we vary the task's



(a) (b)

Fig. 16: Comparison of ET's transmitted energy for three offloading policies when there are 200 WDs. 1) Local computation-only, 2) computation offloading-only, and 3) WiEnTM algorithm.

total CPU cycles. Additionally, Fig. 16b illustrates the ET's energy for different numbers of the task's total input/output bits, which reflect the size of the tasks. As can be observed, WiEnTM substantially reduces the required energy as it achieves the optimal offloading decisions. In particular, according to Fig. 16a, comparing to the local computing case, a higher improvement is obtained for the computationally intensive tasks. This is because the WDs require more energy to execute all their subtasks locally. However, for light tasks with low computation cycles, the WiEnTM algorithm significantly outperforms the offloading case as it is more preferred to execute such tasks locally. The results of Fig. 16b imply that WiEnTM mainly follows the policy of offloading to edge/cloud server for tasks with a small number of input/output bits. As we increase the number of bits, the algorithm tends to rely more on local computing. As can be observed, WiEnTM greatly outperforms both schemes.

XII. Conclusion

In this paper, we studied a three-tier WPMEC system that is consisting of edge and cloud servers as well as WDs. We first proposed a harvest-then-offload mechanism that efficiently schedules the computing resources and allocates the wireless energy. This is achieved by formulating an optimization problem that aims to minimize the ET's energy transmission while satisfying the tasks computation deadline. We then proposed a matching algorithm called WiEnTM that optimally solves the aforementioned problem. We further proposed a parallel implementation of this algorithm with nearly logarithmic complexity. We also investigated the performance of WiEnTM through numerical experiments. Our results show a significant reduction in ET's transmitted energy compared to different offloading policies. Our algorithm also requires up to 34% less overall ET's energy transmission compared to an existing work in the literature.

For future work, we will consider the dynamic behavior of three-tier WPMEC system and propose an online learning approach to achieve the optimal decisions under uncertainty. We will also study more powerful WDs with full-duplex capability in future.

REFERENCES

[1] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surv. Tutor.*, vol. 19, no. 4, pp. 2322–2358, Aug 2017.

[2] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proc IEEE*, vol. 107, no. 8, pp. 1738–1762, Aug 2019.

[3] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *J. Syst. Archit.*, vol. 98, pp. 289–330, Sep 2019.

[4] A. Kaur and A. Godara, "Machine learning empowered green task offloading for mobile edge computing in 5G networks," *IEEE Trans. Netw. Sci. Eng.*, vol. 21, no. 1, pp. 810–820, Feb 2024.

[5] H. Shah-Mansouri and V. W. Wong, "Hierarchical fog-cloud computing for IoT systems: A computation offloading game," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 3246–3257, May 2018.

[6] C. Jiang, X. Cheng, H. Gao, X. Zhou, and J. Wan, "Toward computation offloading in edge computing: A survey," *IEEE Access*, vol. 7, pp. 131 543–131 558, Aug 2019.

[7] L. Wu, P. Sun, H. Chen, Y. Zuo, Y. Zhou, and Y. Yang, "NOMA-enabled multiuser offloading in multicell edge computing networks: A coalition game based approach," *IEEE Trans. Netw. Sci. Eng.*, vol. 11, no. 2, pp. 2170–2181, Mar 2024.

[8] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Process Mag*, vol. 34, no. 6, pp. 26–38, Nov 2017.

[9] L. Huang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Trans. Mob. Comput.*, vol. 19, no. 11, pp. 2581–2593, Jul 2019.

[10] M. Bolourian and H. Shah-Mansouri, "Deep Q-learning for minimum task drop in SWIPT-enabled mobile-edge computing," *IEEE Wireless Commun. Letters*, vol. 13, no. 3, pp. 894–898, Mar 2024.

[11] N. Zhao, Y.-C. Liang, D. Niyato, Y. Pei, M. Wu, and Y. Jiang, "Deep reinforcement learning for user association and resource allocation in heterogeneous cellular networks," *IEEE Trans. Wireless Commun.*, vol. 18, no. 11, pp. 5141–5152, Aug 2019.

[12] M. Tang and V. W. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Trans. Mob. Comput.*, vol. 21, no. 6, pp. 1985–1997, Nov 2020.

[13] C. Sun, X. Li, C. Wang, Q. He, X. Wang, and V. C. Leung, "Hierarchical deep reinforcement learning for joint service caching and computation offloading in mobile edge-cloud computing," *accepted for publication in IEEE Trans. Services Computing*, 2024.

[14] Y. Dai, K. Zhang, S. Maharjan, and Y. Zhang, "Edge intelligence for energy-efficient computation offloading and resource allocation in 5G beyond," *IEEE Trans. Veh. Technol.*, vol. 69, no. 10, pp. 12 175–12 186, Aug 2020.

[15] H. Huang, Q. Ye, and Y. Zhou, "Deadline-aware task offloading with partially-observable deep reinforcement learning for multi-access edge computing," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 6, pp. 3870–3885, Sep 2021.

[16] Z. Liu, Y. Zhao, J. Song, C. Qiu, X. Chen, and X. Wang, "Learn to coordinate for computation offloading and resource allocation in edge computing: A rational-based distributed approach," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 5, pp. 3136–3151, Dec 2021.

[17] H. Zhou, K. Jiang, X. Liu, X. Li, and V. C. Leung, "Deep reinforcement learning for energy-efficient computation offloading in mobile-edge computing," *IEEE Internet Things J.*, vol. 9, no. 2, pp. 1517–1530, Jun 2021.

[18] Z. Gao, L. Yang, and Y. Dai, "Large-scale computation offloading using a multi-agent reinforcement learning in heterogeneous multi-access edge computing," *IEEE Trans. Mob. Comput.*, vol. 22, no. 6, pp. 3425–3443, Jan 2023.

[19] Y. Gong, H. Yao, J. Wang, M. Li, and S. Guo, "Edge intelligence-driven joint offloading and resource allocation for future 6G Industrial Internet of Things," *accepted for publication in IEEE Trans. Netw. Sci. Eng.*, 2024.

[20] L. Liao, Y. Lai, F. Yang, and W. Zeng, "Online computation offloading with double reinforcement learning algorithm in mobile edge computing," *J Parallel Distrib Comput*, vol. 171, pp. 28–39, Jan 2023.

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb 2015.

[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput*, vol. 9, no. 8, pp. 1735–1780, Sep 1997.

[23] L. Yang, H. Zhang, X. Li, H. Ji, and V. C. Leung, "A distributed computation offloading strategy in small-cell networks integrated with mobile edge computing," *IEEE ACM Trans. Netw.*, vol. 26, no. 6, pp. 2762–2773, Dec 2018.

[24] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590–3605, Sep 2016.

[25] A. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single-node case," *IEEE/ACM Trans. Netw*, vol. 1, no. 3, pp. 344–357, Jun 1993.

[26] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. of International Conference on Machine Learning*. New York, NY, Jun 2016.

[27] X. Wang, Y. Han, V. C. M. Leung, X. Y. D. Niyato, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Commun. Surveys & Tuts.*, vol. 22, no. 2, pp. 869–904, Second Quarter 2020.

[28] D. Ma, G. Lan, M. Hassan, W. Hu, and S. K. Das, "Sensing, computing, and communications for energy harvesting IoTs: A survey," *IEEE Commun. Surveys & Tuts.*, vol. 22, no. 2, pp. 1222–1250, Second Quarter 2020.

[29] L. Ji and S. Guo, "Energy-efficient cooperative resource allocation in wireless powered mobile edge computing," *IEEE Internet of Things J.*, vol. 6, no. 3, pp. 4744–4754, Jun. 2019.

[30] Y. Yu, Y. Yan, S. Li, and D. Wu, "Task delay minimization in wireless powered mobile edge computing networks: A deep reinforcement learning approach," in *Proc. of Int. Conf. Wireless Commun. Signal Process. (WCSP)*, Changsha, China, Mar. 2021.

[31] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys & Tuts.*, vol. 19, no. 4, pp. 2322–2358, Fourth Quarter 2017.

[32] Y. Li, Y. Wu, M. Dai, B. Lin, W. Jia, and X. Shen, "Hybrid NOMA-FDMA assisted dual computation offloading: A latency minimization approach," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 5, pp. 3345–3360, Sep. 2022.

[33] K. Guo, M. Yang, Y. Zhang, and J. Cao, "Joint computation offloading and bandwidth assignment in cloud-assisted edge computing," *IEEE Trans. Cloud Comput.*, vol. 10, no. 1, pp. 451–460, Mar. 2022.

[34] M. Aazam, S. Islam, S. T. Lone, and A. Abbas, "Cloud of Things (CoT): Cloud-Fog-IoT task offloading for sustainable Internet of Things," *IEEE Trans. Sustain. Comput.*, vol. 7, no. 1, pp. 87–98, Mar. 2022.

[35] B. Kar, W. Yahya, Y.-D. Lin, and A. Ali, "A survey on offloading in federated cloud-edge-fog systems with traditional optimization and machine learning," [Online]. Available: https://arxiv.org/abs/2202.10628.

[36] R. Zhang and C. Zhou, "A computation task offloading scheme based on mobile-cloud and edge computing for WBANs," in *Proc. of IEEE Int. Conf. Commun. (ICC)*, Seoul, Korea, Repulic of, Mar. 2022.

[37] Z. Gao, W. Hao, and S. Yang, "Joint offloading and resource allocation for multi-user multi-edge collaborative computing system," *IEEE Trans. Veh. Technol.*, vol. 71, no. 3, pp. 3383–3388, Mar. 2022.

[38] P. X. Nguyen, D.-H. Tran, O. Onireti, P. H. Tin, S. Q. Nguyen, S. Chatzinotas, and H. V. Poor, "Backscatter-assisted data offloading in OFDMA-based wireless-powered mobile edge computing for IoT networks," *IEEE Internet of Things J.*, vol. 8, no. 11, pp. 9233–9243, Jun. 2021.

[39] S. Bi and Y. J. A. Zhang, "Wireless powered communication: Opportunities and challenges," *IEEE Commun. Mag.*, vol. 53, no. 4, pp. 117–125, Apr. 2015.

[40] F. Wang, J. Xu, X. Wang, and S. Cui, "Joint offloading and computing optimization in wireless powered mobile-edge computing systems," *IEEE Trans. Wireless Commun.*, vol. 17, no. 3, pp. 1784–1797, Mar. 2018.

[41] Y. J. A. Z. S. Bi, "Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading," *IEEE Trans. Wireless Commun.*, vol. 17, no. 6, pp. 4177–4190, Jun. 2018.

[42] T. Bai, C. Pan, H. Ren, Y. Deng, M. Elkashlan, and A. Nallanathan, "Resource allocation for intelligent reflecting surface aided wireless powered mobile edge computing in OFDM systems," *IEEE Trans. Wireless Commun.*, vol. 20, no. 8, pp. 5389–5407, Aug. 2021.

[43] X. Wang, Z. ning, L. Guo, S. Guo, and G. Wang, "Online learning for distributed computation offloading in wireless powered mobile edge computing networks," *IEEE Trans. Parallel and Distrib. Syst.*, vol. 33, no. 8, pp. 1841–1855, Aug. 2022.

[44] F. Wang, J. Xu, and S. Cui, "Optimal energy allocation and task offloading policy for wireless powered mobile edge computing systems," *IEEE Trans. Wireless Commun.*, vol. 19, no. 4, pp. 2443–2459, Apr. 2020.

[45] D. Wu, F. Wang, X. Cao, and J. Xu, "Wireless powered user cooperative computation in mobile edge computing systems," in *Proc. of IEEE Globecom Workshops (GC Wkshps)*, Abu Dhabi, UAE, Dec. 2018.

[46] Y. Zhang, X. Dong, and Y. Zhao, "Decentralized computation offloading over wireless-powered mobile-edge computing networks," in *Proc. of IEEE Int. Conf. on Artif. Intell. and Inf. Syst. (ICAIIS)*, Dalian, China, Mar. 2020.

[47] L. Huang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile edge computing networks," *IEEE Trans. Mobile Comput.*, vol. 19, no. 11, pp. 2581–2593, Nov. 2020.

[48] X. He, Y. Shen, X. Wang, S. Wang, S. Xu, and J. Ren, "Online scheduling for energy minimization in wireless powered mobile edge computing," in *Proc. of IEEE Wireless Commun. and Netw. Conf. (WCNC)*, Austin, TX, Apr. 2022.

[49] S. Mao, J. Wu, L. Liu, D. Lan, and A. Taherkordi, "Energy-efficient cooperative communication and computation for wireless powered mobile-edge computing," *IEEE Syst. J.*, vol. 16, no. 1, pp. 287–298, Mar. 2022.

[50] T. Zhu, J. Li, Z. Cai, Y. Li, and H. Gao, "Computation scheduling for wireless powered mobile edge computing networks," in *Proc. of IEEE INFOCOM*, Toronto, Canada, Jul. 2020.

[51] D. Fraser, "Pyan," https://github.com/davidfraser/pyan, 2020.

[52] A. Baek, K. Lee, and H. Choi, "CPU and GPU parallel processing for mobile augmented reality," in *Proc. of Int. Congr. on Image and Signal Process. (CISP)*, Hangzhou, China, Dec. 2013.

[53] B. Ma, H. Shah-Mansouri, and V. W. S. Wong, "Full-duplex relaying for D2D communication in mmWave based 5G networks," *IEEE Trans. Wireless Commun.*, vol. 17, no. 7, pp. 4417–4431, Apr. 2018.

[54] Y. Gu, W. Saad, M. Bennis, M. Debbah, and Z. Han, "Matching theory for future wireless networks: Fundamentals and applications," *IEEE Commun. Mag.*, vol. 53, no. 5, pp. 52–59, May 2015.

[55] J. Krarup and P. M. Pruzan, "Reducibility of minimax to minisum 0-1 programming problems," *Eur. J. of Oper. Res.*, vol. 6, no. 2, pp. 125–132, Jan. 2011.

[56] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer, 1999.

[57] Y. Shiloach and U. Vishkin, "An $O(n^2 \log n)$ parallel max-flow algorithm," *J. of Algorithms*, vol. 3, no. 2, pp. 128–146, Feb. 1982.

[58] M. Keller and H. Karl, "Response time-optimized distributed cloud resource allocation," in *Proc. of ACM SIGCOMM Workshop on Distrib. Cloud Comput.*, New York, NY, Aug. 2014.

[59] H. Chen, J. Gong, L. Yuan, and Y. Huang, "Various channel models in wireless communications," in *Proc. of Int. Conf. on Comput. Syst., Electronics and Control (ICCSEC)*, Dalian, China, Dec. 2017.

Responses to the Reviewers' Comments

- **Title of Manuscript**: A QoE-Oriented Computation Offloading Algorithm based on Deep Reinforcement Learning for Mobile Edge Computing
- **Authors**: Iman Rahmati, Hamed Shah-Mansouri, and Ali Movaghar
- **Affiliation**: Departments of Electrical Engineering and Computer Engineering, Sharif University of Technology, Tehran, Iran
- **Paper Reference Number**: TNSE-2024-03-0468

We would like to express our gratitude to the reviewers for their valuable recommendations toward improving the quality of our paper. We also want to gratefully thank the associate editor for handling the paper. Our responses to the reviewers' comments are given below. For convenience, we have used blue font for the revised or newly added paragraphs in the manuscript.

I. Response to Reviewer 1

1) Reviewer's Comment: *"Lots of the studies focus on the research on resource optimisation of MEC by DRL, it is expected to survey the state-of-the-art within 3 years, and summarise it as an individual section, departing from the Section of Introduction. Besides, the existing related work only has one paragraph without any discussion about the pros/cons of the previous work."*

   Authors' Reply:

   Ref

2) Reviewer's Comment: *"Please highlight what and how the proposed approach provides "high performance" to further specify the motivations and contributions. Especially, please highlight the reason why the authors design the framework by integrating LSTM. Otherwise, it will make the innovations seem weak."*

   Authors' Reply:

3) Reviewer's Comment: *"In section III, why is the size of the task selected from a discrete set? Is it selected randomly or in another way? Besides, the authors define the offloading indicate function as $y_{i,j}(t$, it should be involved when modelling the Edge Execution phrase."*

   Authors' Reply:

4) Reviewer's Comment: *"Some sota methods (e.g., DDQN, DDPG) are expected to be compared in the section on performance evaluation, and authors should have more discussions about the evaluation results rather than merely introducing them. Besides, I think it is better to design the evaluations with a real dataset. Besides, 9% and 6% are not significant improvements compared to other methods."*

   Authors' Reply:

5) Reviewer's Comment: *"Other concerns are also expected to be addressed: (a) be careful to use words like "battery level" to express the energy consumption, because some of the energy is provided by the wired line power. (b) Most of the references are out of 3 years, please investigate more studies published in recent years. (c) The authors should thoroughly check the writing style, grammar, and spelling to enhance the paper's clarity, conciseness, and overall impact."*

   Authors' Reply:

   Ref

II. Response to Reviewer 2

1) Reviewer's Comment: *"The authors should provide a comparative table to highlight their contributions. Specially, the authors should compare their method with other references on computation task offloading cited in the paper, including aspects such as scenarios, methods, advances, drawbacks, and so on."*

Authors' Reply:
Ref

TABLE III: Comparison of Related Works

Paper	Scenario	Problem	Objective	Model	Algorithm/Method	Drawbacks
[9]	Wireless-powered MEC	Computation offloading and resource allocations	Maximize computation rate	MIP	DQN	Not consider delay-sensitive tasks
[10]	MEC with EH devices	Energy harvesting computation offloading	Minimize drop rate and energy consumption.	MDP	Deep Q-Learning	
[11]	MEC	Wireless channels competition	Optimize downlink utility	MDP	Multi-agent D3QN	
[12]	MEC	Computation offloading	Minimize drop rate and computation delay.	MDP	D3QN + LSTM	Not consider system energy consumtion
[13]	MEC	Computation offloading and service caching	Minimize average service delay	MDP	hierarchical DRL	Not consider system energy consumtion
[14]	MEC	Resource allocation and computation offloading	Minimize energy consumption	MDP	DDPG	
[15]	MEC	Computation offloading	Minimizing total energy consumption	POMDP	DDPG	
[16]	MEC	Resource allocation and computation offloading	Minimize execution cost	POMDP	MADRL	Not consider delay-sensitive tasks
[17]	MEC	Resource allocation and computation offloading	Minimize energy consumption of the entire system	MDP	DDQN	Only for a singel MEC system model
[18]	heterogeneous MEC	Decentralized computation offloading	Optimize system cost and completion rates	MDP	Multi-agent DDPG	Not take MD's demans into consideration
[19]	MEC-based IIoT	Resource allocation and computation offloading	Minimize task computation delay and energy consumption	MDP	DRL	Not take MD's demans into consideration
[20]	MEC	Online computation offloading	Minimize computation delay and energy consumption	MDP	DDQN	Only for a singel MEC system model
[?]	MEC-based IIoT	Computation offloading	Optimize delay and energy consumption	POMDP	Multi-agent DQN	
[?]	SDN-based MEC	Stochastic game-based resource allocation	Minimize energy consumption and processing delay	MDP	MARL	
[?]	MEC-based IIoT	Joint power allocation and computation offloading	Optimization of privacy protection and quality of service.	MDP	Multi-agent DDPG	
[?]	MEC-based IIoT	Privacy aware computation offloading	Optimization of computation rate and energy consumption	MDP	DDPG	
QECO	MEC	Computation offloading	Maximize the QoE of each MD individually	MDP	D3QN + LSTM	

2) Reviewer's Comment: *"I think more explanations should be added for the QoE function. How can the defined function reflect QoE?"*

Authors' Reply:

3) Reviewer's Comment: *"(a) What inputs and outputs of Algorithm 2? (b) They should be added to make readers clearly understand the results."*

Authors' Reply: (a) To address the reviewer concern about algorithm 2, we have revised secound paragraph of subsection (Training Process Algorithm at EN $j \in \mathcal{J}$) in Section IV to clarify the inputs and outputs of algorithms 2, as follows.

> In particular, regarding experience n, the target-Q value $\hat{Q}_{i,n}^{\mathrm{T}}$ represents the long-term QoE for action $\boldsymbol{a}_i(n)$ under state $\boldsymbol{s}_i(n)$. This value corresponds to the QoE observed in experience n, as well as the approximate expected upcoming QoE. Based on the previous sample experiences $\mathcal{N}$, the EN computes the vector $\hat{\mathbf{Q}}_i^{\mathrm{T}} = (\hat{Q}_{i,n}^{\mathrm{T}})_{n\in\mathcal{N}}$ and trains the MD's neural network (in steps 11-21 of Algorithm 4) to keep parameter vector θ_i^{E} in Net_i^{E} update for the MD's *UpdateRequest*. The key idea of updating Net_i^{E} is to minimize the disparity in Q-values between Net_i^{E} and Net_i^{T}, as indicated by the following loss function: ...

We have also revised algorithm 1 and 2, as below.

Algorithm 3: QECO Algorithm (Training Process)

Input: experience $(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t), \boldsymbol{q}_i(t), \boldsymbol{s}_i(t+1))$ from MD $i \in \mathcal{I}$

Output: parameter vector θ_i^{E}

Algorithm 4: QECO Algorithm (Offloading Decision)

Input: state space $\mathcal{S}$, action space $\mathcal{A}$

Output: MD $i \in \mathcal{I}$ experience $(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t), \boldsymbol{q}_i(t), \boldsymbol{s}_i(t+1))$

4) Reviewer's Comment: *"The authors should open their experiments in github, so that other researchers can compare their work with other ones."*

Authors' Reply: To address the reviewer comment, we make our experiments more accessible by sharing our GitHub rerepository. We have added it into the end of first paragraph of Section V, as follows.

> Detailed settings regarding the neural networks and simulation can be found at our github repository.

We have also provided repository URL link in the footnote of Page 8 as below.

> The code for reproducing the simulation results of this letter is available at https://github.com/ImanRHT/QECO.

5) Reviewer's Comment: *"The following references closely related to the current work should be discussed or compared with the current work for completeness."*

> [30] G. Wu, Z. Xu, H. Zhang, S. Shen, and S. Yu, "Multi-agent DRL for joint completion delay and energy consumption with queuing theory in MEC-based IIoT," in *Journal of Parallel and Distributed Computing*, vol. 176, pp. 80–94, Jun. 2023.
>
> [32] G. Wu, H. Wang, H. Zhang, Y. Zhao, S. Yu, and S. Shen, "Computation offloading method using stochastic games for software-defined-network-based multiagent mobile edge computing," *IEEE Internet of Things Journal*, vol. 10, no. 20, pp. 17620–17634, Oct. 2023.
>
> [33] G. Wu, X. Chen, Z. Gao, H. Zhang, S. Yu, and S. Shen, "Privacy-preserving offloading scheme in multi-access mobile edge computing based on MADRL," *Journal of Parallel and Distributed Computing*, vol. 10, no. 1, pp. 451–460, Mar. 2022.
>
> [36] G. Wu, X. Chen, Y. Shen, Z. Xu, H. Zhang, S. Shen, and S. Yu,"Combining Lyapunov optimization with Actor-Critic networks for privacy-aware IIoT computation offloading," in *IEEE Internet of Things Journal*, 2024, Early Access

Authors' Reply:

Ref

III. Response to Reviewer 1

1) Reviewer's Comment: *"(a) The Introduction is verbose and confusing. (b) The authors should consider clarifying the relationship between task offloading and energy optimization in this paper."*

Authors' Reply: (a) To address the reviewer comment, we have first summarized the explanation of the Internet of Things (IoT) applications, mobile edge computing (MEC), and the motivations of the proposed three-tier computing model. We have revised the first paragraph of Section I as follows.

> The Internet of Things (IoT) applications (e.g., smart cities [27], homes [28], and augmented/virtual reality [29]) are mostly involved with computation-intensive and latency-sensitive tasks. Mobile edge computing (MEC) is a state-of-the-art computation paradigm to address the challenge of insufficient computing resources in IoT networks [30], [31]. By deploying the edge servers, nearby devices can offload their computation tasks. Dispatching those tasks that require a higher amount of energy will let devices save more energy [32]. However, as the computation tasks become more and more intensive, coordination with cloud servers will be inevitable [33]. The three-tier computing models consisting of cloud, edge, and IoT devices are introduced to overcome the aforementioned challenge [34]–[37].

We have also revised the second paragraph of Section I to better emphasize the role of the proposed system in overcoming the challenges of IoT systems.

> In addition to their restricted computation resources, IoT devices usually have small battery capacities which is mainly due to their small form factor and the constraints of the production cost [27].
> ...
> By combining the advantages of MEC and WPC, the newly emerged wireless powered MEC (WPMEC) holds a major promise to overcome the fundamental hurdles in scalability of the IoT networks [40]–[43].

(b) Furthermore, we clarify the relationship between computation offloading and energy optimization. Note that in WPMEC systems, the energy for the operation of WDs is provided by the HAP. Thus, we formulate an optimization problem to minimize the total energy consumption of the HAP at the ET by jointly considering computation offloading, scheduling, and power allocation while guaranteeing the task completion and energy constraints. The following explanations have been added to Section I of the revised manuscript.

> By deploying the edge servers, nearby devices can offload their computation tasks.

> Dispatching those tasks that require a higher amount of energy will let devices save more energy [32].
> . . .
> However, due to the limitation of WET and the cost of ETs, reducing the energy consumption is crucial.
> . . .
> It helps WDs save more energy and reduce the tasks' execution time, which consequently reduces the energy consumption of WPMEC system as WDs harvest energy from ETs.

We have also revised the first paragraph of Section III.A to address the reviewer comment.

> In WPMEC systems, the energy for the operation of WDs is provided by the HAP. Due to the high attenuation of WET over distances [39], a considerable amount of energy is required to be provided for sustainable operation of WDs, which imposes a high cost to HAP. In this paper, we formulate an optimization problem to minimize the total energy consumption of the HAP at the ET by jointly considering computation offloading, scheduling, and power allocation while guaranteeing the task completion and energy constraints.

We hope the above changes make the introduction more concise and less confusing.

2) Reviewer's Comment: *"The authors introduce many works on computation offloading and energy consumption, however, the contributions of the paper are not clear."*

Authors' Reply: We first clarify our contributions in comparison to the existing works. In the ETEM problem formulation, unlike other works in the WPMEC literature (e.g., [29], [44], [47]), we first consider a three-tier computing paradigm that includes wireless devices, edge servers, and the remote cloud server. This architecture helps in further scaling WPMEC systems in terms of computation capability. We also take the MEC server computation time into account to more accurately model the MEC servers. Another main contribution is considering task interdependencies to capture real-world applications. In the WiEnTM algorithm, we first exploit the bipartite graph matching to tackle the complexity of the combinatorial ETEM problem. With efficient parallelization, we then propose a low complexity algorithm that can solve ETEM in real-time.
In summary, the main contributions of our work are:

a) Formulating ETEM problem for a three-tier wireless powered mobile edge computing system considering task interdependencies and MEC server computation time to minimize the energy consumption.
b) Proposing WiEnTM offloading and scheduling algorithm that greatly enhances energy efficiency while imposing a logarithmic time complexity by leveraging parallel processing and bipartite

graph matching. Therefore, the algorithm can easily accommodate the real-time requirements of an offloading scheme. In order to apply the matching algorithm to ETEM problem, we study the characteristics of the problem. We first reduce it into a simplified form (i.e., S-ETEM). Next, we introduce a design parameter $\beta_{v_d^u}$ for each subtask $v_d^u \in \mathcal{V}_d$ of WD $\mathcal{D}^u$ and episode $u \in \mathcal{U}$ to decompose it into a number of independent problems (i.e., DS-ETEM).

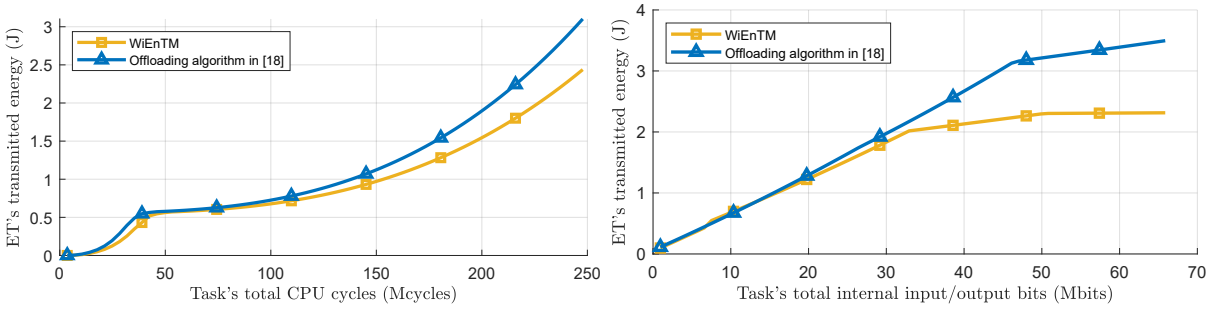
c) Evaluating the performance of WiEnTM in different scenarios and comparing to other state-of-the-art algorithms.

In Section I of the revised manuscript, we have further explained the related works and clarified our main contributions as follows.

> In [36], authors proposed a scheme that aims to minimize the total cost in terms of energy and delay for a three-tier cloud-edge-IoT system. Gao *et al.* in [37] proposed an offloading algorithm to minimize an objective called the energy time cost for a three-tier network. Although [34], [36], and [37] all benefit from a computationally-scalable three-tier system, IoT devices still rely on battery energy and hence lack the sustainability feature of WPMEC systems.
> ...
> However, the proposed system model is oversimplified and thus less applicable to real-world scenarios.

To make the contributions more clear, we have made the following changes to Section I of the revised manuscript.

> By considering the challenges of the aforementioned works, the following questions arise:
> a) *How can we take advantage of cloud computing in addition to MEC to meet different requirements of IoT applications?*
> b) *How can we provide RF energy to enable the battery-less IoT devices operate seamlessly?*
> c) *What would be the optimal offloading decision when considering the subtasks' interdependencies?*
> ...
> a) *Problem formulation*: We formally define the mixed-integer ET's energy minimization (ETEM) problem by deciding on offloading, WET time, and computation scheduling of subtasks for a multi-user three-tier WPMEC system considering task interdependencies and MEC server computation time.
> b) *Algorithm design*: We propose wireless energy transmission minimization (WiEnTM) algorithm based on bipartite graph matching. We first study the properties of the problem to decompose it into individual problems for each subtask to make it suitable for the matching algorithm. By efficiently utilizing parallel processing on decomposed



(a) (b)

Fig. 8: Comparison of ET's transmitted energy between WiEnTM and the algorithm proposed in [44] for different a) task's CPU cycles (i.e., $\sum_{v_d \in \mathcal{V}_d} C_{v_d}$) and, b) task's internal input/output bits (i.e., $\sum_{v_d^i, v_d^j \in \mathcal{V}_d} N_{v_d^i, v_d^j}$).

problems, we prove that our proposed algorithm has a logarithmic time complexity with respect to the network size.

c) *Performance evaluation*: Extensive numerical experiments are carried out to evaluate the proposed algorithm. Our experiments show major reduction in the WDs' required energy in comparison with baseline offloading policies. The algorithm also reduces the energy consumption by 34% for large tasks compared to an existing work [44].

3) Reviewer's Comment: *"Experiments need to be improved. WiEnTm proposed by this paper only compares time with WPMEC- [16]. However, WiEnTm is an energy optimization algorithm, and the main purpose of this paper is energy efficiency. I would like to know the energy performance of WiEnTm compared to other works, but it is not reflected in the paper. In addition, if authors want to show the performance of the proposed mechanism in various aspects, then it is necessary to compare it with state-of-the-art works."*

Authors' Reply: To address the reviewer's comment, we have included Fig. 8 in Section V of the revised manuscript. This figure shows the energy consumption of our proposed algorithm in comparison to a state-of-the-art work in the literature. We have also updated the following paragraph in Section V of the revised manuscript to explain the results.

We further investigate the performance of our proposed WiEnTM algorithm by evaluating the ET's transmitted energy in comparison to that of [44]. Figs. 8a and 8b illustrate the effect of the task's CPU cycles and the task size (i.e., the number of input/output bits), respectively. The results show that by using WiEnTM, the ET consumes up to 34% less energy compared to the algorithm in [44].

We highly appreciate the reviewer for the comments and suggestions. We hope that we have properly

addressed these issues.

IV. Response to Reviewer 2

1) Reviewer's Comment: *"It would be great if the authors provided one or multiple tables of key symbols and their corresponding description in the appendix or paper body instead of describing them in the text. This way is much easier to refer to the symbols."*

   Authors' Reply: In the revised manuscript, we have added the following table which includes the key notations alongside their descriptions.

We would like to gratefully thank the reviewer for the valuable suggestion and comment. We hope that we have addressed this comment satisfactorily.

TABLE I: List of key notations

Symbol	Description
$x^{\mathrm{L}}_{v^u_d}, x^{\mathrm{M}}_{v^u_d}, x^{\mathrm{C}}_{v^u_d}$	Binary offloading decisions
τ_{1,v^u_d}	Length of scheduled time for data transmission of subtask v^u_d
τ_{2,v^u_d}	Length of scheduled time for execution of subtask v^u_d
τ_{3,v^u_d}	Length of scheduled time for retrieving the result of subtask v^u_d
τ^u_0	WET time in episode u
p^u	ET's power in episode u
Γ^u	Length of episode u
$\mu_{v^u_d}$	Response time of cloud server for subtask v^u_d
$C_{v^u_d}$	Number of CPU cycles required for subtask v^u_d
$N_{i,j}$	Number of bits required to transmit the result of subtask i and j
g_d	Channel gain between WD d and ET
h_d	Channel gain between WD d and MEC server
σ^2	Noise power at the HAP
η_d	Energy harvesting efficiency of WD d
ζ_d	CPU's effective switched capacitance of WD d
r_{t,v^u_d}	Virtual resource for tier t and subtask v^u_d
$F^{\mathrm{M,max}}$	Maximum CPU frequency of MEC server
$F^{\mathrm{L,max}}_d$	Maximum CPU frequency of WD d
$P^{\mathrm{ET,max}}$	ET's maximum transmission power
$P^{\mathrm{AP,max}}$	AP's maximum transmission power
$P^{\mathrm{WD,max}}_d$	Maximum transmission power of WD d
D	Number of WDs
V_d	Number of subtasks of WD d
B_d	Bandwidth of WD d
$\mathcal{D}$, $\mathcal{D}^u$	Set of WDs
$\mathcal{U}$	Set of episodes
$\mathcal{V}_d$	Set of subtasks of WD d
$\mathcal{R}^{\mathrm{virt}}$	Set of virtual resources
$\mathcal{I}_{v^u_d}$	Set of episodes associated to executing prerequisite subtasks of the subtask v^u_d
$\mathcal{J}_{v^u_d}$	Set of episodes associated to executing subtasks with the subtask v^u_d as their prerequisite

V. Response to Reviewer 3

1) Reviewer's Comment: *"Although the optimization objective of the investigated problem is to minimize energy consumption of the HAP, the time consumption for all subtasks generated by WDs still should be constrained. Otherwise, the QoS of latency-sensitive tasks cannot be guaranteed by the proposed algorithms."*

Authors' Reply: The time consumption for all subtasks generated by WDs are indeed constrained in our work. With the introduction of episodes, we have simultaneously addressed the requirements of

harvest-then-offload mechanism and the subtasks' deadlines. Constraint (39) applies a hard deadline Γ^u on each episode $u \in \mathcal{U}$ (i.e., a cycle of subtasks' executions) which is obtained based on the deadline of subtasks. In order to clarify this, we have included the following sentences in Section III of the revised manuscript.

> It is worth noting that inequality (39) also enforces a hard deadline for each cycle of subtasks' execution in each episode. Thus, the total execution time of each task generated by WDs is constrained. This way, we can address the requirements of latency-sensitive tasks as well.

Furthermore, we have included the following sentences to better illustrate the intention behind defining an *episode* in Section II of the revised manuscript.

> We regard a complete cycle of executing a single subtask for all WDs having a candidate subtask as an *episode*. Each episode jointly models the harvest-then-offload timing, subtasks' deadlines, and TDMA scheduling of WDs.

2) Reviewer's Comment: *"(a) The definition of an "episode" is not stated clearly. (b) The first question is how to decide the length of an episode? (c) Since the length of an episode will affect the performance of scheduling, it should be analyzed theoretically. (d) Besides, how to decide the number of episodes?"*

Authors' Reply: We address each of the above comments one by one.

(a) In Section II of the revised manuscript, we have first clarified the definition of an episode as follows.

> We regard a complete cycle of executing a single subtask for all WDs having a candidate subtask as an *episode*. Each episode jointly models the harvest-then-offload timing, subtasks' deadlines, and TDMA scheduling of WDs.

(b) An episode is a complete cycle of subtasks' execution, including harvest-then-offload timing and subtasks' deadlines. Therefore, similar to other works in the literature of WPMEC (e.g., [44], [47], [50]), we consider that the length of an episode is directly related to subtasks' deadlines. In the revised manuscript, we have included the following paragraph in Section II to discuss an approach to calculate the length of an episode.

> Now that we have introduced the network model, we need to provide an approach to

calculate the length of episode Γ. As stated earlier, each episode reflects the requirements of the harvest-then-offload timing and subtask deadlines. Therefore, Γ is directly related to the individual subtask deadlines being executed in that episode. In other words, Γ can be calculated as the summation of the subtasks' deadlines. In the next section, we will calculate the length of each episode.

(c) The length of an episode is present in inequality constraint (39). Thus, a longer episode (i.e., the subtasks' deadlines) will enlarge the feasible region of the ETEM optimization problem and enhances the optimal solution and results in less energy consumption. In Section V, we have used this intuition to propose the adaptive approach for calculating the $\beta_{v_d^u}$ parameters. To further clarify, we have included the following sentences in Section III of the revised manuscript.

Also, it can be concluded that Γ^u is a key factor in the optimal value of the ETEM problem. A longer episode (i.e., larger Γ^u) provides more time for execution of the tasks. In fact, this will enlarge the feasible region of the ETEM optimization problem and enhances the energy-efficiency of the algorithm.

(d) The number of episodes corresponds to the time horizon of the problem. We defined the horizon to be the time required to accomplish a task on all WDs. Therefore, as long as there exists a subtask to be executed on WDs, a new episode will be required. This will result in having $\max_{d\in\mathcal{D}} V_d$ episodes. In the revised manuscript, we have updated the following sentences to further clarify how the number of required episodes (i.e., U) is calculated.

We denote $\mathcal{U} \triangleq \{0, ..., U-1\}$ as the set of episodes, where $U \triangleq \max_{d\in\mathcal{D}} V_d$ indicates the number of episodes. Note that V_d is obtained from the call graph of the corresponding task.

3) Reviewer's Comment: *"There are too many symbols in this paper. It is hard for readers to quickly find the meaning of each symbol. A symbol table should be added to improve the readability of this paper."*

Authors' Reply: In the revised manuscript, we have added Table I which includes the key notations alongside their descriptions.

4) Reviewer's Comment: *"In the proposed MGW algorithm, how could all executions in line 4 to line 17 proceed parallelly? As the authors mentioned that all subtasks have dependencies, one subtask should be computed based on the results of other subtasks. In this case, how could the weights be assigned parallelly? A more delicate methods for dealing the dependencies among subtasks should*

TABLE I: List of key notations

Symbol	Description
$x^{\mathrm{L}}_{v^u_d}, x^{\mathrm{M}}_{v^u_d}, x^{\mathrm{C}}_{v^u_d}$	Binary offloading decisions
τ_{1,v^u_d}	Length of scheduled time for data transmission of subtask v^u_d
τ_{2,v^u_d}	Length of scheduled time for execution of subtask v^u_d
τ_{3,v^u_d}	Length of scheduled time for retrieving the result of subtask v^u_d
τ^u_0	WET time in episode u
p^u	ET's power in episode u
Γ^u	Length of episode u
$\mu_{v^u_d}$	Response time of cloud server for subtask v^u_d
$C_{v^u_d}$	Number of CPU cycles required for subtask v^u_d
$N_{i,j}$	Number of bits required to transmit the result of subtask i and j
g_d	Channel gain between WD d and ET
h_d	Channel gain between WD d and MEC server
σ^2	Noise power at the HAP
η_d	Energy harvesting efficiency of WD d
ζ_d	CPU's effective switched capacitance of WD d
r_{t,v^u_d}	Virtual resource for tier t and subtask v^u_d
$F^{\mathrm{M,max}}$	Maximum CPU frequency of MEC server
$F^{\mathrm{L,max}}_d$	Maximum CPU frequency of WD d
$P^{\mathrm{ET,max}}$	ET's maximum transmission power
$P^{\mathrm{AP,max}}$	AP's maximum transmission power
$P^{\mathrm{WD,max}}_d$	Maximum transmission power of WD d
D	Number of WDs
V_d	Number of subtasks of WD d
B_d	Bandwidth of WD d
$\mathcal{D}$, $\mathcal{D}^u$	Set of WDs
$\mathcal{U}$	Set of episodes
$\mathcal{V}_d$	Set of subtasks of WD d
$\mathcal{R}^{\mathrm{virt}}$	Set of virtual resources
$\mathcal{I}_{v^u_d}$	Set of episodes associated to executing prerequisite subtasks of the subtask v^u_d
$\mathcal{J}_{v^u_d}$	Set of episodes associated to executing subtasks with the subtask v^u_d as their prerequisite

be provided."

Authors' Reply: Lines 4 to 17 of the MGW algorithm can be indeed run simultaneously. The algorithm does not require any information about the output of the subtasks' executions in order to execute these lines. In fact, it only requires the sizes of inputs and outputs of each subtask which are given by the task call graph. We have added the following sentences in Section IV of the revised manuscript to better clarify this fact.

Note that in order to calculate the weights of subtasks, there is no need to actually execute them. The only required information is the size of input and output of each subtask which is already known from the task call graph. Thus, Lines 4 to 17 of the MGW algorithm can be run simultaneously.

5) Reviewer's Comment: *"No theoretical guarantee for the energy consumption of the proposed algorithm was provided in this paper. What is the approximation ratio of the proposed algorithm? Theoretical analysis should be provided."*

Authors' Reply: It should be noted that S-ETEM problem achieves the optimal solution of the original ET's energy minimization (ETEM) problem. Furthermore, DS-ETEM problem can achieve the optimal solution with no approximation. In particular, the ETEM problem has been transformed into S-ETEM problem. These two problems are equivalent and can achieve the same optimal solution. This has been achieved by converting the many-to-one matching into a one-to-one matching by introducing the concept of virtual resources. In order to further decompose the ETEM problem and eliminate the dependency of constraint (39), we have introduced the parameter $\beta_{v_d^u}$. By determining the optimal value of $\beta_{v_d^u}$, the optimal energy can be obtained and there would not be any approximation when solving the problem. Please note that determining the optimal value of $\beta_{v_d^u}$ is beyond the scope of our work and is considered for future works. However, there are approaches to calculate it iteratively such as the one given in [50]. In order to address the reviewer's comment, we have revised the manuscript and further clarified the derivation of ETEM and S-ETEM problems and the role of $\beta_{v_d^u}$.

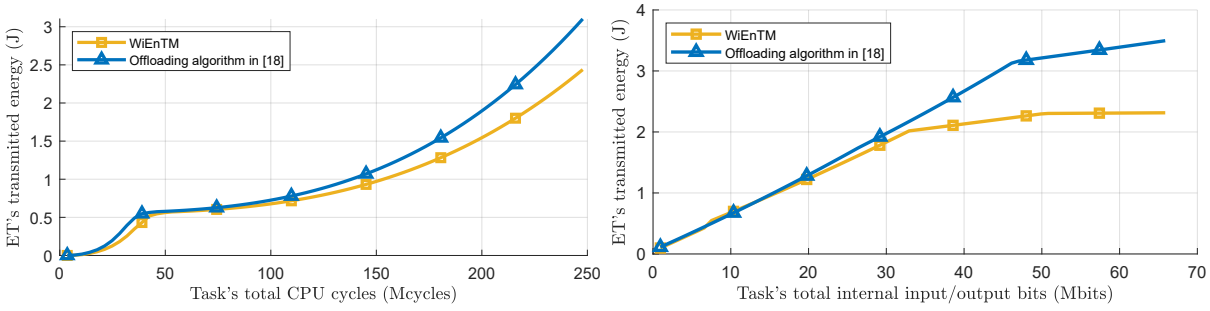
To leverage the matching algorithm, we need to calculate the cost of each matching between subtasks and computing resources. To accomplish this, we first transform the ETEM problem into an equivalent simplified problem. We then show that the transformed problem is decomposable, and the resultant decomposed problems are still equivalent to the ETEM problem.

...

It is also worth noting that for a given $\beta_{v_d^u}$, by solving S-ETEM, we can achieve the optimal solution of the ETEM problem. In other words, by determining the optimal value of $\beta_{v_d^u}$ for each $d \in \mathcal{D}^u$ and $u \in \mathcal{U}$, the optimal ET's transmitted energy can be obtained and there would not be any approximation.

We have also added the following explanation in the footnote of Page 10.

For the case of non-optimal $\beta_{v_d^u}$, one needs to analyze the approximation ratio of the proposed algorithm. An approach similar to [50] can be followed. However, we leave this extension as future work.



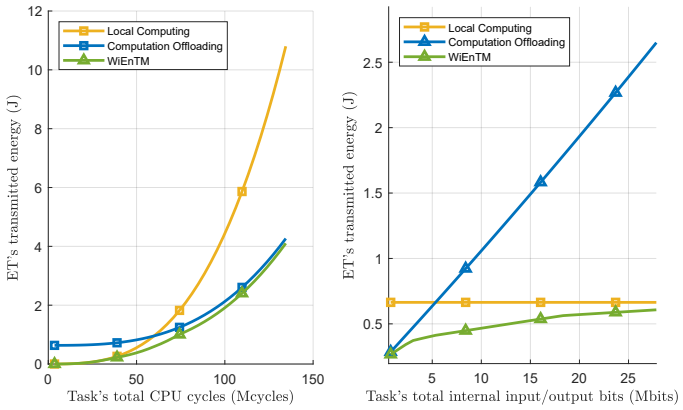
(a) (b)

Fig. 8: Comparison of ET's transmitted energy between WiEnTM and the algorithm proposed in [44] for different a) task's CPU cycles (i.e., $\sum_{v_d \in \mathcal{V}_d} C_{v_d}$) and, b) task's internal input/output bits (i.e., $\sum_{v_d^i, v_d^j \in \mathcal{V}_d} N_{v_d^i, v_d^j}$).

6) Reviewer's Comment: *"In the experiments, only two WDs were deployed in the network. And the dependencies of subtasks generated by each WD were not provided in the experiments. Besides, the authors only evaluated the impact of CPU cycles for WDs on algorithm performance, which is not enough."*

Authors' Reply: To address the reviewer's comment, we have included more simulation results in the revised manuscript. First, we have considered a scenario with 200 WDs and included Figs. 9a and 9b that show ET's transmitted energy for WiEnTM and two baseline schemes. As suggested by the reviewer, in Fig. 9a, we vary the task's total internal input and output bits, while in Fig. 9b, we evaluate the effect of total CPU cycles on energy consumption. In addition, we have included Figs. 8a and 8b that evaluate the effect of both the total number of CPU cycles and the total internal input/output bits of the task on ET's transmitted energy for WiEnTM in comparison to the offloading algorithm in [44]. We have included the following explanations in Section V of the revised manuscript.

> We further investigate the performance of our proposed WiEnTM algorithm by evaluating the ET's transmitted energy in comparison to that of [44]. Figs. 8a and 8b illustrate the effect of the task's CPU cycles and the task size (i.e., the number of input/output bits), respectively. The results show that by using WiEnTM, the ET consumes up to 34% less energy compared to the algorithm in [44].
>
> . . .
>
> We now consider 200 WDs and investigate the ET transmission energy required to enable WDs execute their tasks. We study three different policies: 1) Local computing, 3) computation offloading, and 3) WiEnTM algorithm. We assume that among these devices, 100 WDs have a task with the call graph shown in Fig. 12a and the rest with the task call graph of Fig. 12b.



(a) (b)

Fig. 9: Comparison of ET's transmitted energy for three offloading policies when there are 200 WDs. 1) Local computation only, 2) computation offloading only, and 3) WiEnTM algorithm.

> ...
> Fig. 9a shows the ET's energy transmission when we vary the task's total CPU cycles. Additionally, Fig. 9b illustrates the ET's energy for different numbers of the task's total input/output bits, which reflect the size of the tasks.
> ...
> In particular, according to Fig. 9a, comparing to the local computing case, a higher improvement is obtained for the computationally intensive tasks.
> ...
> The results of Fig. 9b imply that WiEnTM mainly follows the policy of offloading to edge/cloud server for tasks with a small number of input/output bits. As we increase the number of bits, the algorithm tends to rely more on local computing. As can be observed, WiEnTM greatly outperforms both schemes.

Furthermore, the dependencies between subtasks of each WD are shown in Fig. 12. We have included the following sentences in Section V of the revised manuscript to further clarify this issue.

> These figures show the dependencies in the task call graph of each WD. An arrow leaving node i toward j indicates that subtask j is dependent on subtask i.

We would like to gratefully thank the reviewer for the valuable comments and suggestions. We hope that we have addressed these comments satisfactorily.

VI. Response to Reviewer 4

1) Reviewer's Comment: *""an" MEC rather than "a" MEC;"*

   Authors' Reply: We have applied the reviewer's comment throughout the paper and updated the manuscript.

2) Reviewer's Comment: *"This paper considers half-duplex transmission for WDs, where computation offloading cannot be performed simultaneously with energy harvesting. How about the full-duplex?"*

   Authors' Reply: Inspired by [29], [41], [49], [50], in our work, we consider half-duplex transmission for WDs. Due to the production costs, battery-less devices usually lack a full-duplex transceiver. In the case that devices are equipped with full-duplex transceivers, we do not need a harvest-then-offload mechanism as both offloading and energy harvesting can be done simultaneously. There are several works studying simultaneous wireless information and power transfer (SWIPT) [39] for devices with full-duplex transmission. However, this is beyond the scope of our work and is considered for future works. We have added the following explanations to Section II of the revised manuscript.

   > In contrast, for the case of full-duplex transmission, scheduling algorithms are categorized as the simultaneous wireless information and power transfer (SWIPT) [39] mode. However, considering the constraints of the production costs and devices' form factor for battery-less WDs, they usually lack a full-duplex transceiver to perform energy harvesting and data transmission simultaneously.

   We have also added the following sentence to Section VI of the revised manuscript.

   > We will also study more powerful WDs with full-duplex capability in future.

3) Reviewer's Comment: *"The font size in Fig. 2 should be adjusted. For example, the font is too large for "WET"."*

   Authors' Reply: We have adjusted the font size of Fig. 2 as suggested.

4) Reviewer's Comment: *"The authors state that "the partial offloading provides a more realistic model of tasks". It is suggested to provide more explanation."*

   Authors' Reply: In the partial offloading model, each task is represented by its call stack which exactly maps to a real computer program. Note that binary offloading is a special case of partial offloading when we only consider a single partition. To further clarify this issue, we have included

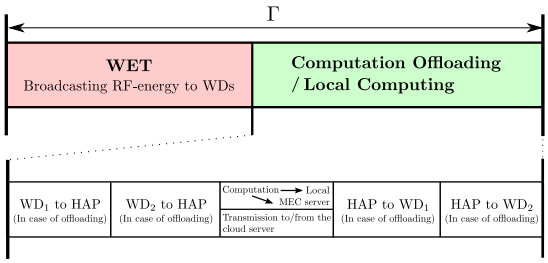


Fig. 2: Harvest-then-offload scheduling with TDMA protocol for a system of two WDs. An episode of length Γ is divided into two times frames of WET and offloading/local computing. In the case of offloading, the offloading/local computing time frame consists of three slots. The first and last slots are for transmitting the data and retrieving the results, respectively. The second slot is dedicated to the computation in the MEC server or transmission to/from the cloud server.

the following explanation in Section II of the revised manuscript.

> The partial offloading provides a more realistic model of tasks as it leverages the task call graph to represent each task. This accurately models the call stack of a computer program.

We have also provided an application example in the footnote of Page 3 as below.

> For instance, *Pyan* [51] is a Python module that generates the task call graph of a Python code by analysis of its modules and functions.

5) Reviewer's Comment: *"Why "the cumulative energy consumption of the devices is no more than their harvested energy"? could you please give a realistic example to illustrate the assumption?"*

Authors' Reply: Indeed, the cumulative energy consumption of devices must be no more than their harvested energy. It is because we consider wireless-powered devices which cannot store energy for later use. Thus, the devices cannot use more energy than the harvested amount. It also ensures the sustainable operation of the devices. The same assumption has been made in similar works [36], [41], [42], [44], [46], [50]. We have included the following sentences in Section II of the revised manuscript to make this fact more clear.

> As described earlier, we assume sustainable operation of WDs by only relying on the harvested energy (i.e., battery-less network). Thus, the cumulative energy consumption of the devices on each episode must be no more than their harvested energy.

6) Reviewer's Comment: *"More recent and important references are required."*

Authors' Reply: To address the reviewer's comment, we have removed a few of outdated references and cited five recently published papers as follows.

[30] Y. Yu, Y. Yan, S. Li, and D. Wu, "Task delay minimization in wireless powered mobile edge computing networks: A deep reinforcement learning approach," in *Proc. of Int. Conf. Wireless Commun. Signal Process. (WCSP)*, Changsha, China, Mar. 2021.

[32] Y. Li, Y. Wu, M. Dai, B. Lin, W. Jia, and X. Shen, "Hybrid NOMA-FDMA assisted dual computation offloading: A latency minimization approach," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 5, pp. 3345–3360, Sep. 2022.

[33] K. Guo, M. Yang, Y. Zhang, and J. Cao, "Joint computation offloading and bandwidth assignment in cloud-assisted edge computing," *IEEE Trans. Cloud Comput.*, vol. 10, no. 1, pp. 451–460, Mar. 2022.

[36] R. Zhang and C. Zhou, "A computation task offloading scheme based on mobile-cloud and edge computing for WBANs," in *Proc. of IEEE Int. Conf. Commun. (ICC)*, Seoul, Korea, Repulic of, Mar. 2022.

[37] Z. Gao, W. Hao, and S. Yang, "Joint offloading and resource allocation for multi-user multi-edge collaborative computing system," *IEEE Trans. Veh. Technol.*, vol. 71, no. 3, pp. 3383–3388, Mar. 2022.

We have also cited these works in the following sentences in Section I of the revised manuscript.

Mobile edge computing (MEC) is a state-of-the-art computation paradigm to address the challenge of insufficient computing resources in Internet of Things (IoT) networks [30], [31]. By deploying the edge servers, nearby devices can offload their computation tasks. Dispatching those tasks that require a higher amount of energy will let devices save more energy [32]. However, as the computation tasks become more and more intensive, a coordination with cloud servers might be inevitable [33]. As a result, three-tier models consisting of cloud, edge, and IoT devices are introduced to overcome the aforementioned challenges [34]–[37].

. . .

In the existing work [36], authors proposed a scheme that aims to minimize the total cost in terms of energy and delay for a three-tier cloud-edge-IoT system. Gao *et al.* in [37] proposed an offloading algorithm to minimize an objective called the energy time cost (ETC) for a three-tier network. Although [34], [36], and [37] all benefit from a computationally-scalable three-tier system, IoT devices still rely on battery energy and they thus lack the sustainability feature of WPMEC systems.

We would like to gratefully thank the reviewer for the valuable comments and suggestions. We hope that we have satisfactorily addressed these comments.