Joint Task Offloading and Resource Allocation for LEO Satellite-Based Mobile Edge Computing Systems With Heterogeneous Task Demands

Liang Zhong, Youyuan Li, Ming-Feng Ge, *Senior Member, IEEE*, Mingjie Feng, *Member, IEEE*, and Shiwen Mao, *Fellow, IEEE*

***Abstract*—To satisfy the growing demand for supporting intelligent Internet of Things (IoT) applications in remote areas, satellite mobile edge computing (SMEC) systems are expected to be widely deployed. Meanwhile, as IoT applications are increasingly diversified, the tasks received by SMEC systems present heterogeneous demands for various resources and quality of service (QoS) metrics, necessitating customized design to accommodate such heterogeneity. In this paper, we investigate the problem of joint task offloading and resource allocation in SMEC systems with heterogeneous task demands. We first propose a customized task utility model that captures the diversified demands, which is an extension of traditional unified task utility models. Based on the model, we then formulate a mixed integer non-linear programming (MINLP) problem for joint optimization of task offloading, computing resource allocation, transmission power control, and user association, aiming to maximize the sum utility of all tasks. To solve the MINLP, a multi-layer iterative framework is proposed that decomposes the original problem into two subproblems, which are solved by successive convex approximation (SCA) and deep reinforcement learning (DRL) algorithms, respectively. Simulation results show that, by applying the proposed task utility model, the average utility performance of UEs can be improved by 50% and 100% compared to applying a single metric-based utility model and a typical classification-based utility model, respectively; the proposed task offloading and resource allocation scheme achieves a 30%–80% performance gain compared to benchmark schemes.**

***Index Terms*—Satellite mobile edge computing (SMEC), task offloading, resource allocation, deep reinforcement learning (DRL), successive convex approximation (SCA).**

Received 9 July 2024; revised 22 November 2024 and 26 January 2025; accepted 4 March 2025. Date of publication 7 March 2025; date of current version 18 July 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62371429 and Grant 62101202, and in part by Hubei Provincial Natural Science Foundation and the Innovative Development of Smart Transportation of China under Grant JCZRLH202500424. The review of this article was coordinated by Prof. Xiaojiang Du. *(Corresponding author: Mingjie Feng.)*

Liang Zhong, Youyuan Li, and Ming-Feng Ge are with the School of Mechanical Engineering and Electronic Information, China University of Geosciences, Wuhan 430072, China (e-mail: zhongliang@cug.edu.cn; lyy991202@163.com; gemf@cug.edu.cn).

Mingjie Feng is with the Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: mingjiefeng@hust.edu.cn).

Shiwen Mao is with the Department of Electrical and Computer Engineering, Auburn University, Auburn, AL 36849 USA (e-mail: smao@ieee.org).

Digital Object Identifier 10.1109/TVT.2025.3549119

I. Introduction

With the emergence of computational-intensive Internet of Things (IoT) applications with stringent latency requirements (e.g., autonomous vehicles, intelligent robotics, immersive interactions), it becomes challenging for computation-limited and battery-powered terminal devices to timely process the computational tasks generated by these applications. Mobile edge computing (MEC) is an effective solution to address this issue. By deploying computing servers at the network edge (e.g., base stations), IoT devices can offload their computational tasks to these servers for fast processing, resulting in reduced task completion latency. Meanwhile, as IoT applications are extended to remote areas and oceans (e.g., environment monitoring, renewable energy plant management, unmanned operation), the demand for global computing services is surging. Since traditional MEC systems rely on cellular networks to perform task offloading, they cannot satisfy the computing needs in remote or underserved areas. Deploying MEC servers in these areas seems to be a solution, but the high cost caused by large-scale deployment and long-distance maintenance makes this approach impractical. With the recent rapid deployment of low earth orbit (LEO) satellite communication networks, satellite mobile computing (SMEC) system, which extends MEC functionality to satellite networks, has become a promised paradigm to meet the computing demands of emerging IoT applications. By offloading computational tasks to the MEC servers on satellites (in particular LEO satellites), the computational tasks generated at remote areas can be executed at LEO satellites and/or forwarded to cloud servers for execution, resulting in low task processing latency [1], [2], [3].

Same as traditional MEC systems, task offloading and resource allocation are two major factors that impact the performance of SMEC systems [4], [5], and have been extensively investigated. However, due to the features of tasks processed by SMEC systems, the design of efficient task offloading and resource allocation strategies is challenging. Specifically, as the application scenarios in remote areas are increasingly diverse, the computational tasks to be executed become highly heterogeneous. As a result, the demand for various resources and the preference for different quality of service (QoS) metrics are diversified among tasks. For instance, tasks supporting autonomous operations need a significant amount of computing resources for data processing and intelligent decision-making, and require extremely low latency to guarantee the timeliness of operation; in contrast, monitoring-related tasks need less computing resources since no decision-making is involved, and have much less stringent requirements for latency [6], [7]. Meanwhile, traditional task offloading and resource allocation strategies are designed for homogeneous tasks with unified task utility models (i.e., the utility for completing a task as a function of allocated resources and achieved QoS metrics). In these models, the utility function is a fixed combination of predefined QoS metrics [8], [9], which cannot capture the diverse demand of emerging applications, resulting in low resource utilization and degraded task utility. To this end, it is necessary to establish a customized utility model for each task, and design task offloading and resource allocation strategies according to the utility models of all tasks. However, the relationship between task satisfaction and the achieved resources/various QoS metrics is complex, necessitating novel task utility models that are adaptive to the diverse task demands in SMEC networks.

Besides task heterogeneity, task offloading and resource allocation in SMEC systems are also challenged by the dynamics of satellite networks. As satellites move along their orbits, the connectivity and the channel of device-satellite links are constantly changing, resulting in time-varying network topology, link quality, and resource availability [10], [11]. As a result, traditional "snapshot-based" solutions, which solely use the system state in the current time slot to determine the strategies in the next time slot, cannot adapt to the dynamics of SMEC systems, resulting in low resource utilization. To address this issue, deep reinforcement learning (DRL) has been applied to generate *proactive* strategies in MEC systems [12], [13]. DRL-based solutions enable environment-aware strategy adjustments by capturing the pattern of system dynamics, thereby enhancing task utility. However, due to the fast-growing number of IoT devices in SMEC systems, the dimensions of the problems handled by DRL algorithms also increase exponentially [14]. This results in slow training convergence, which may not adapt to the time-varying network topology of SMEC systems, necessitating novel solutions with reduced dimensions and accelerated convergence.

In this paper, we investigate the problem of task offloading and resource allocation in LEO satellite-based SMEC systems with heterogeneous task demands. Aiming to improve the sum task utility of all devices served by an SMEC system, we develop novel models and solutions to capture the diverse task demands as well as the dynamics of SMEC systems. The main contributions of this paper are summarized as follows:

1) We establish a multi-dimensional heterogeneous task utility model based on Gaussian mixture model (GMM), in which a customized task utility function is derived for every individual task. The task utility function has two parts. The first part is a weighted sum of Gaussian functions, where the weights indicate the preferences of demands for various resources and QoS metrics. The second part is the energy consumption of the device. The expectation maximization (EM) algorithm is applied to yield the optimal weight of each Gaussian component.
2) Based on the task utility model, we formulate joint task offloading and resource allocation as a mixed-integer nonlinear programming (MINLP) problem. Specifically, we formulate joint optimization of task offloading, computing resource allocation, transmission power control, and user association for full offloading (tasks are either locally executed or offloaded) and partial offloading (tasks can be partitioned and partially offloaded) scenarios, aiming to maximize the sum utility of all tasks.
3) Due to the presence of coupling constraints and binary decision variables, the formulated MINLP cannot be solved by standard approaches. To this end, we propose a multi-layer iterative framework that decomposes the original problem into two subproblems, which are solved by DRL and successive convex approximation (SCA), respectively.
4) The effectiveness of the proposed solutions in both full offloading and partial offloading scenarios is evaluated by simulations. The results show that the proposed schemes achieve higher utility than benchmark schemes, indicating the superior capability in capturing diverse task demands.

The remainder of this paper is organized as follows. The related works are summarized in Section II. The system model and problem formulation are given in Section III and Section IV, respectively. In Section V, we present the proposed solution. The simulation results are shown and analyzed in Section VI. Finally, Section VII concludes this paper.

II. Related Work

As two key design issues of MEC systems, task offloading and resource allocation have been widely investigated. Based on different models, several optimization techniques have been applied to obtain solutions for task offloading and resource allocation, such as Lyapunov optimization [15], [16], [17] and convex optimization [18], [19]. In particular, in the presence of multiple edge nodes, game theory was applied to enable collaborative task offloading and resource allocation [20], and matching theory was employed to derive a low-complexity solution for user association [7], [21]. Furthermore, heuristic algorithms like genetic algorithms and simulated annealing algorithms were also used to obtain suboptimal solutions that can be easily implemented [22], [23]. However, these methods did not fully consider the system dynamics, which may not achieve satisfactory performance in fast-varying environments (e.g., satellite networks) [14]. In contrast, our solution employs DRL to capture the pattern of environmental change caused by satellite movement and obtain proactive strategies that can adapt to the dynamics of SMEC systems.

Due to the capability of intelligent decision-making in dynamic environments, DRL algorithms were applied to task offloading and resource allocation recently. In [24], a DRL-based task offloading and resource allocation strategy was designed to maximize the long-term utility of vehicle edge computing networks. In [25], a software-defined framework was introduced to manage and orchestrate networking, caching, and computing resources, and a deep Q-learning approach was proposed to obtain an efficient solution. Since DRL models require extensive training, the training process should be optimized to accelerate convergence. In [14], an adaptive genetic algorithm was incorporated into the DRL framework to enhance the accuracy of exploration by avoiding useless exploration, which speeds up convergence without sacrificing the performance of task processing. Aiming to support real-time decisions for task offloading and resource allocation, a two-timescale DRL approach was proposed to reduce overhead, which consists of a fast-timescale and a slow-timescale learning process [26]. Considering that the optimization of task offloading and resource allocation may involve continuous decision variables (e.g., offloading ratio and power control), an actor-critic-based DRL framework was designed to deal with continuous action space [27]. Recently, DRL algorithms were applied in task offloading and resource allocation in SMEC systems. Aiming to minimize the latency of satellite-assisted Internet of Things services, joint task offloading and multidimensional resource allocation were formulated as a mixed integer dynamic programming problem and solved by a DRL-based approach [28]. In [29], the authors proposed a DRL-based task offloading and resource allocation strategy using deep recurrent Q-learning networks, in which recurrent layers are added into DQN to learn the temporal features induced by satellite movement and thereby minimize the task completion latency of IoT devices. In [30], coordinated task offloading and resource allocation in SMEC systems was formulated and analyzed under a multi-agent DRL model, in which a centralized training with decentralized execution framework was implemented for applying DRL algorithms at various network nodes. Although DRL has been successfully applied in these works, the solutions were designed for homogeneous tasks with unified demands. In contrast, our scheme can accommodate tasks with heterogeneous requirements by establishing customized task utility models.

Recently, studies have investigated optimizing task offloading and resource allocation for diversified tasks. Typically, the tasks are classified into several categories and processed accordingly. In [31], tasks are classified into two types, computing-intensive and delay-sensitive, each type has a different constraint of QoS metrics. In [32], the tasks are assigned with seven priorities according to IEEE standards. For example, higher-priority tasks have lower delay tolerance and stricter QoS constraints. The reward for receiving tasks is proportional to the amount of data and priority, hence the task priority is a reference for resource allocation. In [33], a fog queue system with limited infrastructure resources was designed to handle real-time heterogeneous tasks, in which the tasks are classified into three types: standard, storage-intensive, and CPU-intensive. Each type of task has a different preference for various resources, and the resource allocation is performed according to the preferences. In [34], an energy-aware collaborative computing offload paradigm was proposed for processing heterogeneous tasks, in which tasks are classified into two types: delay-sensitive tasks, such as face recognition and fingerprint recognition, and energy-sensitive tasks, such as interactive games. In [35], tasks are classified according to the required computational load: light load tasks, such as video transcoding, medium load tasks, such as chess, and heavy load tasks, such as face recognition. In [36],

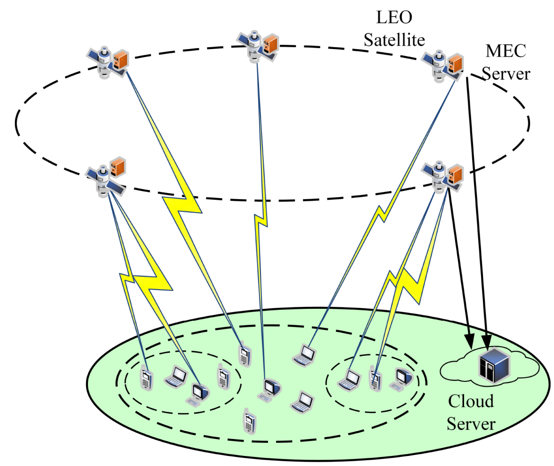


Fig. 1. Illustration of LEO Satellite-based MEC System.

priority-aware task data offloading for edge computing-assisted medical services was investigated, in which the medical services are classified into two categories: emergency services (e.g., remote surgery), which require real-time response and are referred to as hard deadline tasks, and other tasks such as medical report analysis, which are usually tolerant to delays and referred to as soft deadline tasks. In [37], task offloading and resource allocation strategies are designed according to the types of mobile devices: high-priority (subscription) mobile devices and low-priority (best effort) mobile devices. Each mobile user generates tasks supporting multimedia applications based on Poisson processes with varying densities, such as video uploads, video conferencing, augmented reality, or online games. To guarantee the QoS of high-priority devices, the tasks generated by these devices are allocated more resources from the server than the ones generated by low-priority devices. Although these works considered the diversity of tasks, the tasks are classified into given types, and prior knowledge of task type is required for designing task offloading and resource allocation strategies. Besides, the above classifications are relatively static, which may not accurately capture the customized demands of individual tasks. For example, a task can be a weighted combination of delay-sensitive and computational-intensive, hence the demand for various resources and QoS metrics varies across tasks. In contrast, we design a customized task utility function for each task without requiring prior knowledge of the task type.

III. System Model

As shown in Fig. 1, we consider an SMEC system with a remote cloud computing server and LEO satellites. The LEO satellites are connected to the remote cloud server through wireless backhaul channels. The network comprises M heterogeneous IoT devices/users indexed by $\mathcal{M} = \{1, \ldots, m, \cdots M\}$,[1] N LEO satellites indexed by $\mathcal{N} = \{1, \ldots, n, \cdots N\}$, and a cloud server. Every LEO satellite is outfitted with an MEC server, which executes the IoT tasks offloaded to the LEO

[1] For convenience, we use "device" and "user" interchangeably in this paper.

TABLE I
KEY NOTATIONS

Notation	Description
$\beta_{m,j}$	Association indicator between user m and satellite j
ρ_m	Offloading ratio of user m
$a_{m,n}^{\text{edge}}$	Offloading decision indicator for user m and satellite n
a_m^{cloud}	Offloading decision indicator for user m and cloud server
D_m	Data size of task m
C_m	Number of CPU cycles needed to execute task m
T_m^{total}	Total delay for completing task m
$T_m^{\max}$	Maximum tolerable delay for completing task m
$F_m^{\min}$	Minimum required computing resource for task m
F_m^{avg}	Average computing resource available to task m
$r_{m,n}$	Uplink data transmission rate of user m and satellite n
E_m^{total}	Total energy consumption for completing task m
p_m	Transmission power of user device m
f_m^{local}	Computational capability of the device of user m
$f_{m,n}^{\text{edge}}$	Computing resource allocated to task m by satellite n
f_m^{cloud}	Computing resource allocated to task m by the cloud server
λ_1, λ_2	Weights in task utility functions

satellite. Inter-satellite links exist between satellites and data are transmitted via laser communications.

In each scheduling time slot, each device generates one task, hence the tasks are also indexed by $\mathcal{M}$. The uplink bandwidth of satellite n is denoted by B_n, and the computing resource of its corresponding MEC server is denoted as F_n. All tasks can be executed locally and/or remotely on an edge server or a cloud server. The offloading destination of a task is determined by the association indicator and the offloading indicator. The association indicator $\beta_{m,j}$ represents the connection relationship between the user m and the LEO satellite j. If a task is offloaded, it can be processed by the edge server deployed at the associated satellite or transmitted to another edge or cloud server. The offloading indicator $a_{m,n}^{\text{edge}}$ signifies whether user m's task is offloaded to LEO satellite n, while a_m^{cloud} indicates whether the task is offloaded to the cloud server.

Subsequently, we define the offloading decision as ρ_m, which is used to indicate whether task m will be offloaded (fully offloaded) or the proportion of offloading (partially offloaded). Evidently, it is influenced by both the association indicator and the offloading indicator. For a fully offloaded scenario, if $\rho_m = 0$, the task is processed locally; conversely, if $\rho_m = 1$, the task is offloaded. For a partially offloaded scenario, $\rho_m \in [0, 1]$.

For task m, the parameter is denoted as $\boldsymbol{\vartheta}_m = \langle D_m, C_m, T_m^{\max}, F_m^{\min} \rangle, \forall m \in \mathcal{M}$. Here, D_m represents the data amount of the task (in bits), C_m refers to the number of CPU cycles required to complete the task (in CPU cycles), $T_m^{\max}$ indicates the maximum tolerable delay, and $F_m^{\min}$ signifies the minimum computing resource necessary for successful task processing (in GHz).

Since tasks can only be offloaded to a single MEC or cloud server, the following constraint must be satisfied:

$$1 - \rho_m + \rho_m \left(\sum_n a_{m,n}^{\text{edge}} + a_m^{\text{cloud}} \right) = 1, \forall m \in \mathcal{M}, \quad (1)$$

The key notations used in the paper are listed in Table I.

A. Communication Model

We consider Non-orthogonal multiple access (NOMA) based space-ground communication for task offloading from users to LEO satellites. At each LEO satellite, Successive Interference Cancellation (SIC) technique is applied to decode signals from different mobile devices connected to the same LEO satellite. To perform SIC, the LEO satellite first decodes the signal of the user with the highest channel gain, while the signals of other users are treated as interference. After decoding the signal of the first user, the LEO satellite removes the decoded signal from the received signals and decodes the strongest signal among the remaining ones. Such a process is repeated until the signals of all users are decoded [38]. Compared to the delay for task execution and data transmission, the delay for SIC is negligible [39], [40], [41]. Thus, we neglect the delay of SIC and assume that task processing begins after all user signals are decoded. Let $h_{m,n}$ be the channel gain between user m and LEO satellite n. Without loss of generality, we assume $h_{1,n} \leq h_{2,n} \leq \cdots \leq h_{M,n}$. For satellite-user wireless links, the Line-of-Sight (LoS) path plays the dominant role, while the weak scattering components that traverse various non-line-of-sight (NLoS) paths are less significant. Thus, the channel for task offloading is modeled as Rician channels [42]. The channel gain between user m and LEO satellite n is as follows:

$$h_{m,n} = \sqrt{\frac{\eta}{1+\eta}} h_{m,n}^{\text{LoS}} + \sqrt{\frac{1}{1+\eta} (d_{m,n})^{-\alpha_m^{'}}} h_{m,n}^{\text{NLoS}}, \quad (2)$$

where η is the Rician factor, $\alpha_m^{'}$ is the path loss exponent under the NLoS transmission, and $h_{m,n}$ is the channel gain in the LoS case, which can be represented by:

$$h_{m,n}^{\text{LoS}} = \sqrt{(d_{m,n})^{-\alpha_m}} e^{-j \frac{2\pi}{\varphi} d_{m,n}}, \quad (3)$$

$d_{m,n}$ is the distance between user m and satellite n, α_m is the path loss exponent in the LoS case, and φ is the wavelength of Ka-band. $h_{m,n}^{\text{NLoS}}$ is the corresponding small-scale channel gain, which follows distribution $\mathcal{CN}(0, 1)$. Then, the achievable uplink transmission rates of user m when associated with LEO satellite n can be expressed as:

$$r_{m,n} = B_n \log_2 \left(1 + \frac{p_m h_{m,n}}{\sum_{i=1, i\neq m}^{M-1} \beta_{i,n} p_i h_{i,n} + \sigma^2} \right), \quad (4)$$

where B_n is the bandwidth of LEO satellite n, p_m is the uplink transmission power of user m, σ^2 is the additive white Gaussian noise power. $\sum_{i=1, i\neq m}^{M-1} \beta_{i,n} p_i h_i$ represents the interference originating from other mobile devices associated with the same LEO satellite.

The channel between different LEO satellites follows the free space path loss model, given by:

$$|h_{j,n}| = G \cdot \left(\frac{c}{4\pi f d_{j,n}} \right)^2, \quad (5)$$

where G is the antenna gain of each LEO satellite, $d_{j,n}$ is the distance between satellite j and n, c is the speed of light, and f is the carrier frequency.

Considering that inter-satellite communication links are usually implemented by highly directional laser beams, the interference between different links can be neglected. Then, the data rate between satellites n and j is given by

$$R_{j,n} = B^{\text{sat}} \log_2 \left(1 + \frac{|h_{j,n}^2| \cdot p^{\text{sat}}}{\sigma'^2}\right), \tag{6}$$

where σ'^2 is the Gaussian white noise power; and p^{sat} is the transmitted power of each satellite.

B. Delay and Energy Models

1) Local Computing: Let f_m^{local} (in CPU cycles/s) be the computational capacity of mobile device m. When mobile device m locally processes its task, the computing delay and energy consumption can be expressed as:

$$T_m^{\text{local}} = \frac{(1-\rho_m)\, C_m}{f_m^{\text{local}}}, \forall m \in \mathcal{M}, \tag{7}$$

$$E_m^{\text{local}} = \kappa\,(1-\rho_m)\, C_m \left(f_m^{\text{local}}\right)^2, \forall m \in \mathcal{M}, \tag{8}$$

where κ is the effective capacitance coefficient, which depends on the chip architecture of the mobile device.

2) Edge Computing: Without loss of generality, suppose task m is executed by (full or partial) satellite n. We first consider the case where task m is offloaded to and executed by its associated satellite j, i.e., $j = n$. Since the data size of the task outcome is small, we neglect the delay and energy consumption for transmitting the task outcome. Then, the communication delay is the uplink transmission delay during offloading, calculated by:

$$T_{m,n}^{\text{edge,comm}} = \frac{\rho_m D_m}{r_{m,n}}, \forall m \in \mathcal{M}, \forall n \in \mathcal{N}. \tag{9}$$

We then consider the case where user m is associated with satellite j while its task is executed by satellite n. In this case, the task will first be sent to satellite j and then forwarded to satellite n for processing, i.e., $(j \neq n)$. After that, satellite n sends the outcome to satellite j, which then forwards the outcome to user m. The total communication delay from user m offloading its task to satellite n is calculated by:

$$T_{m,n}^{\text{edge,comm}} = \frac{\rho_m D_m}{r_{m,j}} + \frac{2d_{j,n}}{c} + T_{j,n}^{\text{trans}}, \forall m \in \mathcal{M}, \forall n, j \in \mathcal{N}, \tag{10}$$

where $d_{j,n}$ is the distance between satellite j and satellite n, and $T_{j,n}^{\text{trans}}$ is the transmission delay for the satellite to transmit the received computational task.

The total energy consumption for data transmission during task offloading is calculated as follows:

$$E_{m,n}^{\text{edge,comm}} = \begin{cases} p_m \frac{\rho_m D_m}{r_{m,j}} + p_{\text{sat}} T_{j,n}^{\text{trans}}, & j \neq n; \\ p_m \frac{\rho_m D_m}{r_{m,n}}, & j = n, \end{cases} \tag{11}$$

where p_m is the transmission power of user device m, $\frac{\rho_m D_m}{r_{m,j}}$ is the time for transmitting the task data from user m satellite j. Thus, the energy consumption of data transmission during task offloading is $p_m \frac{\rho_m D_m}{r_{m,j}}$. Similarly, $p_{\text{sat}} T_{j,n}^{\text{trans}}$ denotes the energy consumption of data transmission from satellite j to satellite n.

Denote the computing resources allocated to user m as $f_{m,n}^{\text{edge}}$, which is the number of CPU cycles allocated to task m by the MEC server on satellite n [43], [44]. Given the task offloading ratio ρ_m, the number of CPU cycles required to complete the part of task m at satellite n is $\rho_m C_m$. Then, the computing delay for executing task m by satellite n is given by:

$$T_{m,n}^{\text{edge,comp}} = \frac{\rho_m C_m}{f_{m,n}^{\text{edge}}}, \forall m \in \mathcal{M}, \forall n \in \mathcal{N}. \tag{12}$$

The energy consumption for executing task m at satellite n is:

$$E_{m,n}^{\text{edge,comp}} = \kappa \rho_m C_m \left(f_{m,n}^{\text{edge}}\right)^2, \forall m \in \mathcal{M}, \forall n \in \mathcal{N}. \tag{13}$$

The total delay and energy consumption for completing task m when it is offloaded to satellite n are respectively given by:

$$T_{m,n}^{\text{edge}} = T_{m,n}^{\text{edge,comm}} + T_{m,n}^{\text{edge,comp}}, \forall m \in \mathcal{M}, \forall n \in \mathcal{N}, \tag{14}$$

$$E_{m,n}^{\text{edge}} = E_{m,n}^{\text{edge,comm}} + E_{m,n}^{\text{edge,comp}}, \forall m \in \mathcal{M}, \forall n \in \mathcal{N}. \tag{15}$$

Compared to other components in an SMEC network, the energy supplies of user devices and satellites are unstable since they are typically powered by batteries with limited life and capacity. Thus, we take the energy consumption of users and satellites into account.

3) Cloud Computing: When user m offloads (full or partial) its task to the cloud server for processing, it first sends the task to its associated satellite (i.e., satellite j). Then, satellite j forwards the task to the cloud server. After completing the task, the task outcome is sent back from the cloud server to user m via satellite j. Thus, the total communication delay when the task is offloaded to the cloud server is given by:

$$T_m^{\text{cloud,comm}} = \frac{\rho_m D_m}{r_{m,j}} + \frac{2d_{j,\text{cloud}}}{c} + \frac{\rho_m D_m}{r_{j,\text{cloud}}}, \quad \forall m \in \mathcal{M}, \forall j \in \mathcal{N}, \tag{16}$$

where $d_{j,\text{cloud}}$ is the distance between satellite j and the cloud server, $r_{j,\text{cloud}}$ is the data rate for the transmission from satellite j to the cloud server, which can be calculated in the same way as $r_{m,j}$.

Let f_m^{cloud} be the computing resource allocated to task m by the cloud server, which is predetermined by the cloud service agreement. Then, the computing delay of task m is given by:

$$T_m^{\text{cloud,comp}} = \frac{\rho_m C_m}{f_m^{\text{cloud}}}, \forall m \in \mathcal{M}. \tag{17}$$

Then, the total delay for offloading and processing the task in the cloud is:

$$T_m^{\text{cloud}} = T_m^{\text{cloud,comm}} + T_m^{\text{cloud,comp}}, \forall m \in \mathcal{M}. \tag{18}$$

Based on the above calculations, we can calculate the overall delay and energy consumption for each task m as a function of decision variables $< \beta_{m,j}, a_{m,n}^{edge}, a_m^{cloud}, \rho_m >$. The total delay for completing task m can be expressed as:

$$T_m^{\text{total}} = T_m^{\text{local}} + \sum_{n \in \mathcal{N}} a_{m,n}^{\text{edge}} T_{m,n}^{\text{edge}} + a_m^{\text{cloud}} T_m^{\text{cloud}}. \tag{19}$$

The total energy consumption of devices and satellites for completing task m is calculated by:

$$E_m^{\mathrm{total}} = E_m^{\mathrm{local}} + \sum_{n \in \mathcal{N}} a_{m,n}^{\mathrm{edge}} E_{m,n}^{\mathrm{edge}}. \tag{20}$$

The average computing resource available to task m is:

$$F_m^{\mathrm{avg}} = (1-\rho_m)\, f_m^{\mathrm{local}} + \rho_m f_{m,n}^{\mathrm{edge}} \sum_{n \in \mathcal{N}} a_{m,n}^{edge} + \rho_m a_m^{cloud} f_m^{cloud}, \forall m \in \mathcal{M}. \tag{21}$$

IV. Problem Formulation

In this section, we first present the GMM framework that establishes the task utility model. Then, we present the formulation of task offloading and resource allocation, where the joint optimization of task offloading, transmission power control, computing resource allocation, and user association is formulated as an MINLP problem.

A. GMM-Based Task Utility Model

As mentioned, identifying the preference of resources for each task is critical for efficient resource utilization and improving task utility. Given the limited resources and the heterogeneous tasks with different preferences, task offloading and resource allocation strategies should be designed according to the features of a group of tasks that share the same computing utilities and resources. Therefore, we apply the GMM model to establish task utility models that are based on the group of tasks to be executed at the same time, aiming to obtain task utility functions that can accurately reflect both the preferences of demands of a task and the relative importance of this demand compared to all tasks' demands.

In our task utility model, the demand of each task is modeled as a weighted combination of several baseline demands. Each baseline demand corresponds to a demand for a certain QoS metric or a type of resource, and the weight of a baseline demand is the task's preference for that demand. Correspondingly, the utility of a task is a weighted sum of the utilities obtained by satisfying various baseline demands. The utility obtained from satisfying a baseline demand is a function of the achieved QoS metric or allocated resource for that demand. The weights indicate the preferences for different demands, and we use GMM to obtain the weights.

GMM is a soft clustering method for determining the probability that each data point belongs to a given cluster. It is based on the assumption that all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. Mapping into our model, the task parameters are the data points, each baseline demand is a cluster, and the weights in the utility function (i.e., the preferences for different demands) are the weights for various Gaussian distributions. By obtaining the weights and parameters of various Gaussian distributions via GMM, the task utility function can be obtained. Specifically, given a set of tasks parameterized by the tuple $\boldsymbol{\vartheta}_m, m \in \mathcal{M}$, GMM is used to approximate the distribution of task parameters by obtaining the mean, covariance, and weight of each Gaussian distribution, which indicates the importance of each distribution in the overall distribution, reflecting the relative importance of each QoS metric/resource to the task.

The weight for each Gaussian distribution in GMM is obtained by maximizing a likelihood function with an expectation maximization (EM) algorithm [45]. Suppose a GMM is comprised of K Gaussian distributions, which correspond to K dimensions of task demands. Then, the probability density function of the parameters of task m is given by:

$$\mathrm{prob}\left(x|\boldsymbol{\pi}_m, \boldsymbol{\theta}_m\right) = \sum_{k=1}^{K} \pi_{m,k} \phi\left(x|\theta_{m,k}\right), \tag{22}$$

where $\boldsymbol{\pi}_m = \{\pi_{m,1}, \ldots, \pi_{m,K}\}$ and $\boldsymbol{\theta}_m = \{\theta_{m,1}, \ldots, \theta_{m,K}\}$. $\pi_{m,k}$ is the weight of the k-th Gaussian distribution (demand dimension) of task m, which satisfies $\sum_{k=1}^{K} \pi_{m,k} = 1$. $\phi(x|\theta_{m,k})$ is the probability density function of task demand dimension k.

In a Gaussian Mixture Model, the covariance matrix Cov for each component is calculated using maximum likelihood estimation. The covariance matrix of the k-th Gaussian component is calculated by $\frac{1}{N_k}\sum_{i=1}^{N} \gamma_{i,k}(x_i - \mu_k)(x_i - \mu_k)^\top$, where N_k represents the number of data points assigned to the k-th component, while $\gamma_{i,k}$ indicates the responsibility of the i-th data point for that component, typically derived from posterior probabilities. x_i denotes the i-th data point and μ_k is the mean vector of the k-th component.

Then, the EM algorithm is used to find a set of optimal parameters through iterations. Recall that the set of task parameters is $\{\boldsymbol{\vartheta}_1, \boldsymbol{\vartheta}_2, \ldots, \boldsymbol{\vartheta}_M\}$, the weight optimization problem is formulated as:

$$\max_{\boldsymbol{\pi}_m, \boldsymbol{\theta}_m} \prod_{m=1}^{M} \mathrm{prob}\left(\boldsymbol{\vartheta}_m|\boldsymbol{\pi}_m, \boldsymbol{\theta}_m\right). \tag{23}$$

Let $\boldsymbol{\pi}_m^* = \{\pi_{m,1}^*, \ldots, \pi_{m,K}^*\}$ be the optimal solution for (23). Then, the coordinate point of task m in a K-dimensional task demand coordinate system can be obtained, as shown in Fig. 2.

Based on the multidimensional coordinate system, the mapping between task demand and task utility can be established by the GMM model, and the optimal weights and parameters in the task utility function are obtained by the EM algorithm.

Let $U_k(\cdot)$ be the utility function for the k-th task demand dimension, which is a concave function (e.g., logarithmic function) to reflect the effect of diminishing marginal utility Specifically, the gain for allocating more resource/improving a QoS metric is reduced as more resource is allocated/better QoS is achieved. Then, the multi-dimension utility model for task m is given by:

$$U(\mathbf{Q}_m) = \sum_{k=1}^{K} (-1)^{\varpi} U_k(\boldsymbol{\pi}_m^*(k) \cdot Q_m(k)), \tag{24}$$

where $Q_m(k)$ denotes the achieved QoS metric/allocated resource of task m in dimension k, and ϖ indicates if the QoS metric/allocated resource is positively correlated ($\varpi = 0$) or negatively correlated ($\varpi = 1$) to demand dimension k.

In this paper, we use two primary dimensions of task demand as an example: computing resources and delay. Our model and solution can be extended to the case with more dimensions.

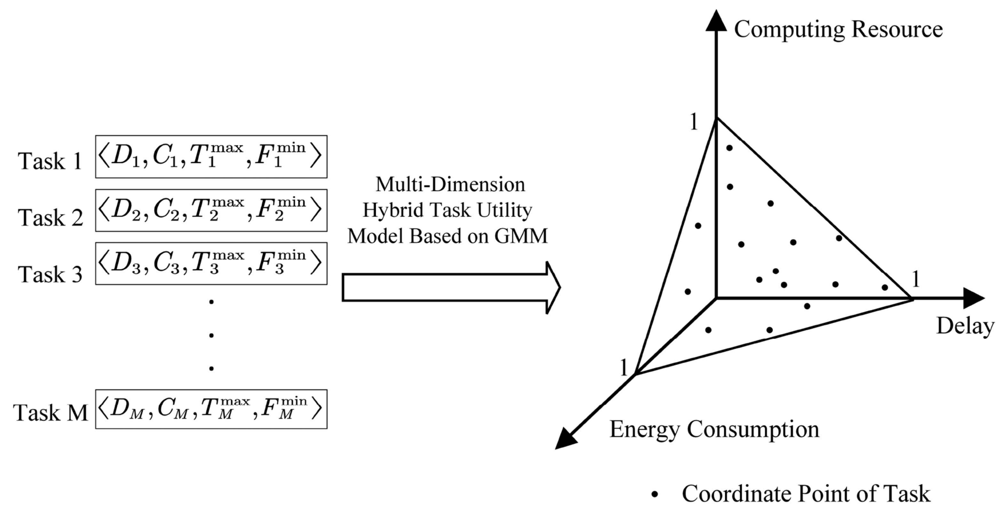


Fig. 2. Schematic diagram of multi-dimension task model.

The utility of task m obtained from the computing resource dimension is positively correlated to the average available computing resource F_m^{avg}. Using the minimum required computing resource $F_m^{\min}$ as the reference, the utility for the computing resource dimension is defined as:

$$U_1\left(\lambda_1 \frac{F_m^{\mathrm{avg}}}{F_m^{\min}}\right), \forall m \in \mathcal{M}, \tag{25}$$

where λ_1 is a normalization factor.

Likewise, the utility of task m obtained from the delay dimension is negatively correlated to the task completion delay T_m^{total}. Using the maximum tolerable delay $T_m^{\max}$ as the reference, the utility for the delay dimension is defined as:

$$U_2\left(\lambda_2 \frac{T_m^{\max}}{T_m^{\mathrm{total}}}\right), \forall m \in \mathcal{M}, \tag{26}$$

where λ_2 is also a normalization factor. Since $U_2(\cdot)$ is a concave function, it can also be expressed as $-U_2(\lambda_2 \frac{T_m^{\mathrm{total}}}{T_m^{\max}})$.

Let $(\boldsymbol{\pi}_m^*(1)$ and $\boldsymbol{\pi}_m^*(2))$ be the weights of computing resources and delay obtained by solving Eq.(23). The utility of task m is expressed as:

$$U_1\left(\boldsymbol{\pi}_m^*(1)\lambda_1 \frac{F_m^{avg}}{F_m^{\min}}\right) - U_2\left(\boldsymbol{\pi}_m^*(2)\lambda_2 \frac{T_m^{\mathrm{total}}}{T_m^{\max}}\right). \tag{27}$$

The task utility function is a linear combination of utilities across multiple dimensions, which is concave with respect to the achieved QoS metrics.

B. Formulation for Joint Optimization of Task Offloading and Resource Allocation

With the multi-dimensional task utility functions, we formulate joint task offloading and resource allocation as an MINLP problem, aiming to improve task utilization as well as reduce energy consumption. The decision variables include the ones for transmission power control $\mathbf{p} = \{p_m, \forall m \in \mathcal{M}\}$, computing resource allocation $\mathbf{f} = \{f_m, \forall m \in \mathcal{M}\}$, task offloading factor $\boldsymbol{\rho} = \{\rho_m, \forall m \in \mathcal{M}\}$, task offloading indicator $\boldsymbol{\Pi} = \{\boldsymbol{\Pi}_m, \forall m \in \mathcal{M}\}$, where $\boldsymbol{\Pi}_m = \{a_{m,1}^{\mathrm{edge}}, \ldots, a_{m,N}^{\mathrm{edge}}, a_m^{\mathrm{cloud}}\}$ and user association indicator $\boldsymbol{\beta} = \{\beta_{m,j}, \forall m \in \mathcal{M}, \forall j \in \mathcal{N}\}$. Then the optimization problem is defined as follows:

$$\begin{aligned}
\mathbf{P1}: \max_{\{\boldsymbol{\rho},\boldsymbol{\beta},\boldsymbol{\Pi},\mathbf{f},\mathbf{p}\}} \; & w_1 \sum_{m\in\mathcal{M}} \left[U_1\left(\lambda_1 \boldsymbol{\pi}_m^*(1) \frac{F_m^{\mathrm{avg}}}{F_m^{\min}}\right) \right. \\
& \left. - U_2\left(\lambda_2 \boldsymbol{\pi}_m^*(2) \frac{T_m^{\mathrm{total}}}{T_m^{\max}}\right)\right] - w_2 \sum_{m\in\mathcal{M}} E_m^{\mathrm{total}} \\
\text{s.t.} \quad & C1: \; T_m^{\mathrm{total}} \le T_m^{\max}, \; \forall m \in \mathcal{M} \\
& C2: \; F_m^{\mathrm{avg}} \ge F_m^{\min}, \forall m \in \mathcal{M} \\
& C3: \; \sum_{m\in\mathcal{M}} a_{m,n}^{\mathrm{edge}} f_{m,n}^{\mathrm{edge}} \le F_n, \forall n \in \mathcal{N} \\
& C4: \; r_{m,j} \ge r_{\min}, \; \forall m \in \mathcal{M}, \forall j \in \mathcal{N} \\
& C5: \; P_{\min} \le p_m \le P_{\max}, \; \forall m \in \mathcal{M} \\
& C6: \; \textstyle\sum_j \beta_{m,j} = 1, \; \forall m \in \mathcal{M} \\
& C7: \; \beta_{m,j} \in \{0,1\}, \forall m \in \mathcal{M}, \forall j \in \mathcal{N} \\
& C8: \; \rho_m \in \{0,1\}, \forall m \in \mathcal{M} \\
& C9: \rho_m \left(\sum_n a_{m,n}^{\mathrm{edge}} + a_m^{\mathrm{cloud}} - 1\right) = 0, \forall m \in \mathcal{M}
\end{aligned}$$

where w_1 and w_2 are the weights for balancing between utility and energy consumption. $C1$ specifies the maximal tolerable delay for each task, $C2$ indicates the minimum computing resource requirement [46]. $C3$ specifies the maximum computing capacity that can be provided by each MEC server, while $C4$ refers to the minimum data rate requirement for offloading, and $C5$ indicates the range of transmission power for each user. $C6$ indicates that each user can be associated with only one satellite. Finally, $C9$ indicates that a task can only be offloaded to a single edge or cloud server.

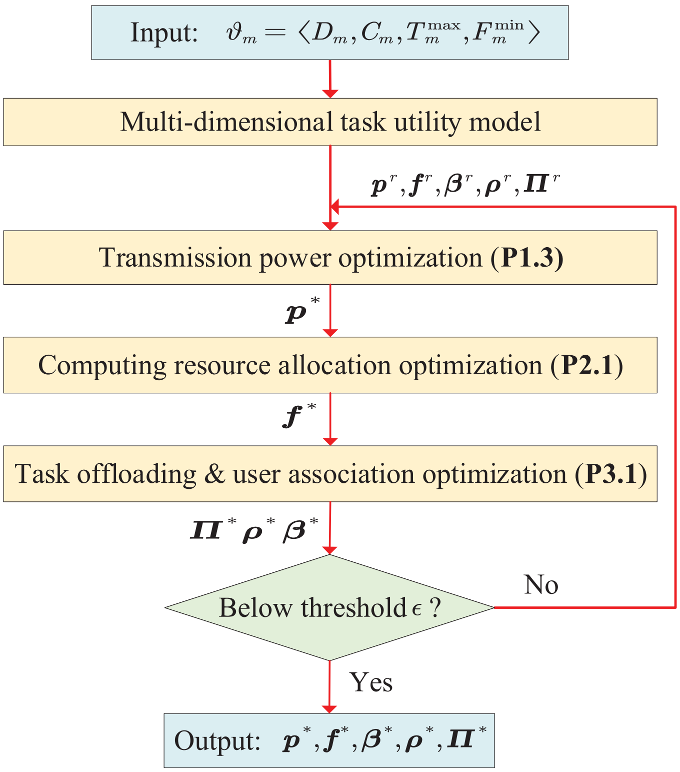


Fig. 3. The architecture of EDHTR.

V. Proposed Task Offloading and Resource Allocation Scheme

Due to the non-convexity and mixed types of variables, solving $\mathbf{P1}$ with a standard approach is infeasible. We propose an iterative framework named EDHTR, which is based on a combination of DRL and convex optimization, as shown in Fig. 3. In each iteration of EDHTR, the optimization starts with transmission power control, followed by computational resource allocation, and finally task offloading and user association. The iterative optimizations continue until the termination criterion is met. Note that, the order of solving various subproblems does not affect convergence and the achievable performance, since the value of the objective function is improved after each iteration until convergence is achieved. It is also worth mentioning that solving various subproblems causes certain energy consumption and occupies certain computing resources, which may impact the resource availability and system performance. However, as in many wireless communication systems, the proposed optimizations are performed before task offloading and relevant communication/computational processes take place. Thus, the energy and computing resource consumptions for performing various optimizations do not impact the energy and delay performance for task offloading and processing, and they are not considered in our optimization model.

A. Optimal Weights for Task Utility Function

The EM algorithm is an iterative method for maximizing the log-likelihood of the data with two steps: The expectation (E) step and the maximization (M) step. For each task m, the EM algorithm first initializes the parameters $\boldsymbol{\theta}_m = \{\theta_{m,1}, \ldots, \theta_{m,K}\}$, where $\theta_{m,k} = (\mu_{m,k}, \sigma_{m,k})$ are the mean and variance of the k-th Gaussian distribution for task m. Then, the E and M steps are iteratively performed until the algorithm converges to a local optimal solution. Note that GMM is sensitive to initial parameters and may converge to different local optima. Thus, under different initial parameters, we execute the EM algorithm multiple times to obtain the optimal weights in the task utility model. Specifically, given the utility model parameters at the current iteration, the E step calculates the probability of each data point belonging to each Gaussian distribution, which is formulated as:

$$\gamma_{m,k} = \frac{\pi_{m,k}\phi\left(\boldsymbol{\vartheta}_m|\theta_{m,k}\right)}{\sum_{k=1}^{K}\pi_{m,k}\phi\left(\boldsymbol{\vartheta}_m|\theta_{m,k}\right)}. \tag{28}$$

Based on the calculated probabilities, the M step updates the utility model parameters by maximizing the log-likelihood. The parameters are updated as:

$$\mu_{m,k} = \frac{\sum_{m\in\mathcal{M}}\gamma_{m,k}\boldsymbol{\vartheta}_m}{\sum_{m\in\mathcal{M}}\gamma_{m,k}}, \tag{29}$$

$$\sigma_{m,k} = \frac{\sum_{m\in\mathcal{M}}\gamma_{m,k}\left(\boldsymbol{\vartheta}_m - \mu_{m,k}\right)\left(\boldsymbol{\vartheta}_m - \mu_{m,k}\right)^T}{\sum_{m\in\mathcal{M}}\gamma_{m,k}}, \tag{30}$$

$$\pi_{m,k} = \frac{\sum_{m\in\mathcal{M}}\gamma_{m,k}}{N}. \tag{31}$$

When the iterations terminate, the optimal $\boldsymbol{\theta_m}^*$ and $\boldsymbol{\pi_m}^*$ are determined for task m, i.e. the GMM-based task utility function is obtained.

B. Power Control Optimization

With given $\{\boldsymbol{\rho}, \boldsymbol{\beta}, \boldsymbol{\Pi}, \mathbf{f}\}$, the power control problem of is formulated as:

$$\begin{aligned}
\mathbf{P1.1}: &\min_{\mathbf{p}} w_1 \sum_{m\in\mathcal{M}} U_2\left[\frac{\lambda_2\left(\boldsymbol{\pi}_m^*(2)\right)}{T_m^{\max}}\left((1-\rho_m)\frac{C_m}{f_m^{\text{local}}}\right.\right.\\
&\quad + \rho_m\sum_{n\in\mathcal{N}} a_{m,n}^{\text{edge}}\left(\frac{D_m}{r_{m,j}} + T_{j,n}^{\text{sat}} + \frac{C_m}{f_{m,n}^{\text{edge}}}\right)\\
&\quad \left.\left. + \rho_m a_m^{\text{cloud}}\left(\frac{D_m}{r_{m,j}} + T_C + \frac{C_m}{f_m^{\text{cloud}}}\right)\right)\right]\\
&\quad + w_2\sum_{m\in\mathcal{M}}\left((1-\rho_m)\kappa C_m\left(f_m^{\text{local}}\right)^2\right.\\
&\quad \left. + \rho_m\sum_{n\in\mathcal{N}} a_{m,n}^{\text{edge}}\left(\frac{p_m D_m}{r_{m,j}} + \kappa C_m\left(f_{m,n}^{\text{edge}}\right)^2\right)\right)\\
&\quad \text{s.t.} \quad C1,\ C4, \text{ and } C5.
\end{aligned}$$

Similar to $\mathbf{P1}$, $\mathbf{P1.1}$ is a non-convex optimization problem, which cannot be solved by standard methods. Thus, we apply SCA to obtain a suboptimal solution of $\mathbf{P1.1}$. We first introduce a set of slack variables $\mathbf{s} = \{s_{m,j} = r_{m,j}, \forall m \in \mathcal{M}\}$. Then, we approximate each $r_{m,j}$ as a concave function with the following lemma.

Lemma 1: For a given p_m^τ, $r_{m,j}$ can be approximated as the following concave function:

$$r_{m,j} \geq \bar{r}_{m,j} - \breve{r}_{m,j}^{up} \triangleq \breve{r}_{m,j}^{\tau}, \tag{32}$$

where

$$\bar{r}_{m,j} = \log_2\left(\sum_{i=1}^{m-1} \beta_{i,n} p_i h_{i,n} + \sigma^2 + p_m h_{m,n}\right), \tag{33}$$

$$\breve{r}_{m,j}^{\rm up} = \upsilon_{m,j}\left(p_i - p_i^\tau\right) + \nu_{m,j}, \tag{34}$$

$$\upsilon_{m,j} = \sum_{i=1}^{m-1} \frac{\beta_{i,n} h_i \log_2 e}{\sum_{j=1}^{m-1} \beta_{j,n} p_j^\tau h_j + \sigma^2}, \tag{35}$$

$$\nu_{m,j} = \log_2\left(\sum_{i=1}^{m-1} \beta_{i,n} p_i^\tau h_j + \sigma^2\right). \tag{36}$$

Proof: First, we rewrite $r_{m,j}$ is as follows:

$$r_{m,j} = \bar{r}_{m,j} - \breve{r}_{m,j}, \tag{37}$$

where $\bar{r}_{m,j}$ is concave and $\breve{r}_{m,j}$ is given by

$$\breve{r}_{m,j} = \log_2\left(\sum_{i=1}^{m-1} \beta_{i,n} p_i h_i + \sigma^2\right), \tag{38}$$

because $\breve{r}_{m,j}$ is concave, we can derive its upper bound $\breve{r}_{m,j}^{\rm up}$ for a given p_m^τ in iteration τ:

$$\breve{r}_{m,j} \leq \upsilon_{m,j}\left(p_i - p_i^\tau\right) + \nu_{m,j} \triangleq \breve{r}_{m,j}^{\rm up}, \tag{39}$$

where $\upsilon_{m,j}$ and $\nu_{m,j}$ are constants. Then, $\breve{r}_{m,j}$ can be replaced by its upper bound $\breve{r}_{m,j}^{\rm up}$. Accordingly, $r_{m,j}$ is approximated as a concave function $\breve{r}_{m,j}^{\tau}$. □

By introducing slack variables, the task delay can be represented as:

$$\begin{aligned} T_m^{\rm total} &= (1-\rho_m)\frac{C_m}{f_m^{\rm local}} + \rho_m a_m^{\rm cloud}\left(\frac{D_m}{s_{m,j}} + T_C + \frac{C_m}{f_m^{\rm cloud}}\right) \\ &\quad + \rho_m \sum_{n\in\mathcal{N}} a_{m,n}^{\rm edge}\left(\frac{D_m}{s_{m,j}} + T_{j,n}^{\rm sat} + \frac{C_m}{f_{m,n}^{\rm edge}}\right) \triangleq \tilde{T}_m. \end{aligned} \tag{40}$$

Based on Lemma 1, add the corresponding constraint $s_{m,j} - \breve{r}_{m,j}^{\tau} \geq 0$, $\mathbf{P1.1}$ is rewritten as:

$$\begin{aligned} \mathbf{P1.2}: &\min_{\{\mathbf{p},\mathbf{s}\}} w_1 \sum_{m\in\mathcal{M}} U_2\left(\frac{\lambda_2\left(\boldsymbol{\pi}_m^*(2)\right)\tilde{T}_m}{T_m^{\max}}\right) \\ &\quad + w_2 \sum_{m\in\mathcal{M}}\left(\left(1-\rho_m\right)\kappa C_m\left(f_m^{\rm local}\right)^2\right. \\ &\quad \left. + \rho_m \sum_{n\in\mathcal{N}} a_{m,n}^{\rm edge}\left(p_m\frac{D_m}{s_{m,j}} + \kappa C_m\left(f_{m,n}^{\rm edge}\right)^2\right)\right) \\ &\text{s.t.}\quad P_{\min} \leq p_m \leq P_{\max}, \forall m \in \mathcal{M} \\ &\qquad r_{\min} - s_{m,j} \leq 0, \forall m \in \mathcal{M} \\ &\qquad \tilde{T}_m \leq T_m^{\max}, \forall m \in \mathcal{M} \\ &\qquad s_{m,j} - \breve{r}_{m,j}^{\tau} \geq 0, \forall m \in \mathcal{M}. \end{aligned}$$

Note that the final term in the objective function is not convex. Thus, $\mathbf{P1.2}$ is not a convex optimization problem. To address this issue, another set of slack variables, denoted as $\boldsymbol{\eta} = \{\eta_{m,j} = p_m/s_{m,j}, \forall m \in \mathcal{M}\}$, is used to approximate the expression $p_m/s_{m,j}$ using a convex function. Utilizing the Taylor expansion, the linear form of the item $p_m/s_{m,j}$ is derived as:

$$\begin{aligned} \eta_{m,j} = \frac{p_m}{s_{m,j}} &\geq \frac{p_m^\tau}{s_{m,j}^\tau} + \frac{1}{s_{m,j}^\tau}\left(p_m - p_m^\tau\right) \\ &\quad - \frac{p_m^\tau}{\left(s_{m,j}^\tau\right)^2}\left(s_{m,j} - s_{m,j}^\tau\right) \triangleq \hat{\zeta}_{m,j}. \end{aligned} \tag{41}$$

With the additional constraint $\eta_{m,j} \geq \hat{\zeta}_{m,j}$. Thus, the optimization problem $\mathbf{P1.2}$ can be reformulated as:

$$\begin{aligned} \mathbf{P1.3}: &\min_{\{\mathbf{p},\mathbf{s},\boldsymbol{\eta}\}} w_1 \sum_{m\in\mathcal{M}} U_2\left(\frac{\lambda_2\left(\boldsymbol{\pi}_m^*(2)\right)\tilde{T}_m}{T_m^{\max}}\right) \\ &\quad + w_2 \sum_{m\in\mathcal{M}}\left(\left(1-\rho_m\right)\kappa C_m\left(f_m^{\rm local}\right)^2\right. \\ &\quad \left. + \rho_m \sum_{n\in\mathcal{N}} a_{m,n}^{\rm edge}\left(D_m\eta_m + \kappa C_m\left(f_{m,n}^{\rm edge}\right)^2\right)\right) \\ &\text{s.t.}\quad P_{\min} \leq p_m \leq P_{\max}, \forall m \in \mathcal{M} \\ &\qquad r_{\min} - s_{m,j} \leq 0, \forall m \in \mathcal{M} \\ &\qquad \tilde{T}_m \leq T_m^{\max}, \forall m \in \mathcal{M} \\ &\qquad s_{m,j} - \breve{r}_{m,j}^{\tau} \geq 0, \forall m \in \mathcal{M} \\ &\qquad \hat{\zeta}_{m,j} - \eta_m \leq 0, \ \forall m \in \mathcal{M}. \end{aligned}$$

The objective function and constraints of $\mathbf{P1.3}$ are convex, making $\mathbf{P1.3}$ a convex optimization problem, which can be solved with the CVX toolbox.

C. Computing Resource Allocation Optimization

Given $\boldsymbol{\rho}$, $\boldsymbol{\beta}$, $\boldsymbol{\Pi}$, and $\mathbf{p}^*$, the computing resource allocation optimization is formulated as:

$$\begin{aligned} \mathbf{P2.1}: &\max_{\mathbf{f}} w_1 \sum_{m\in\mathcal{M}}\left[U_1\left(\lambda_1\boldsymbol{\pi}_m^*(1)\frac{F_m^{\rm avg}}{F_m^{\min}}\right)\right. \\ &\quad \left. - U_2\left(\lambda_2\boldsymbol{\pi}_m^*(2)\frac{T_m^{\rm total}}{T_m^{\max}}\right)\right] - w_2 \sum_{m\in\mathcal{M}} E_m^{\rm total} \\ &\text{s.t.}\quad C1 - C3. \end{aligned}$$

In $\mathbf{P2.1}$, $\mathbf{f}$ is the only optimization variable. The objective function is a logarithmic function of $\mathbf{f}$, the constraints in $C1$ are the reciprocal functions of $\mathbf{f}$, while $C2$ and $C3$ are specified by affine functions of $\mathbf{f}$. Due to the convexity of these functions, $\mathbf{P2.1}$ is a convex optimization problem.

D. Task Offloading and User Association Optimization

We consider two task offloading scenarios: fully offloading and partial offloading. In the fully offloading scenario, tasks are either processed on the local device or offloaded to edge/cloud

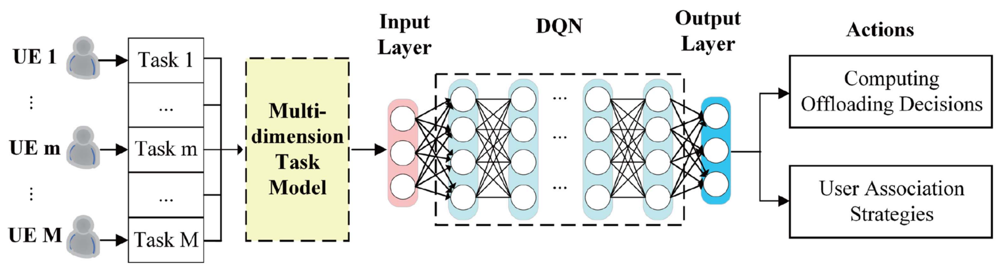


Fig. 4. The framework of the proposed DQN-based computing offloading and user association scheme.

servers, i.e., each ρ_m can only take the binary values of 0 or 1. In the partial offloading scenario, tasks are allowed to be partitioned and partially offloaded to edge/server servers, i.e., each ρ_m is relaxed to a continuous variable in [0,1]. Given $\mathbf{p}^*$ and $\mathbf{f}^*$, the computing offloading and user association subproblem is formulated as:

$$\begin{aligned}\mathbf{P3.1}: \max_{\boldsymbol{\rho},\boldsymbol{\beta},\boldsymbol{\Pi}} \ & w_1 \sum_{m\in\mathcal{M}} \left[U_1\left(\lambda_1 \boldsymbol{\pi}^*_m(1)\frac{F_m^{\text{avg}}}{F_m^{\min}}\right) \right. \\ & \left. -U_2\left(\lambda_2 \boldsymbol{\pi}^*_m(2)\frac{T_m^{\text{total}}}{T_m^{\max}}\right)\right] - w_2 \sum_{m\in\mathcal{M}} E_m^{\text{total}} \\ \text{s.t.} \ & C2,\ C6\ C9.\end{aligned}$$

$\mathbf{P3.1}$ is a complex multi-constrained 0-1 integer optimization problem that is challenging to solve. Therefore, this paper proposes a DRL-based solution.

1) User Association and Fully Offloading: The framework of the proposed approach is shown in Fig. 4. $\mathbf{P3.1}$ is modeled as a Markov decision process (MDP), in which each mobile device acts as an agent. Next, we define the state space $\mathcal{S}$, action space $\mathcal{A}$, and reward $reward$ of the MDP.

The state space of mobile device/agent m is defined as:

$$\text{state}(m) = \{p^*_m, f^*_m, h_{m,1}, \ldots, h_{m,N}, C_m\} \in \mathcal{S}, \tag{42}$$

where p^*_m and f^*_m are the optimal transmission power and computational resource of mobile device m, respectively, which are obtained by the solutions in the previous subsection. The action space of agent m is defined as:

$$\text{action}(m) = \{\rho_m, \beta_{m,1}, \ldots, \beta_{m,N}, \boldsymbol{\Pi}_m\} \in \mathcal{A}, \tag{43}$$

Based on the offloading decision, offloading indicator, and association indicator of each mobile device, the system executes the tasks accordingly and switches to the next state.

The reward function should consider both the objective function and constraints in the optimization problem. Thus, we set the reward as a function of the computing resource utility, processing delay, and energy consumption. The reward received by agent m is given by:

$$\begin{aligned}&\text{reward}(m)\\ &= w_1\left[U_1(\lambda_1\boldsymbol{\pi}^*_m(1)\frac{F_m^{\text{avg}}}{F_m^{\min}}) - U_2(\lambda_2(\boldsymbol{\pi}^*_m(2))\frac{T_m^{\text{total}}}{T_m^{\max}})\right]\\ &\quad - w_2 E_m^{\text{total}} + \min\{T_m^{\max} - T_m^{\text{total}}, 0\}\\ &\quad + \min\{F_m^{\text{avg}} - F_m^{\min}, 0\},\end{aligned} \tag{44}$$

where the latter two terms are penalty terms imposed for the case that the constraints on processing delay and computing resource are violated, respectively.

The formulated MDP is solved with a DRL approach, which is implemented by training a DQN. Algorithm 1 presents the detailed process of DQN training. The training data is generated by the agent's interaction with an environment simulator, which comprises mobile devices, MECs, cloud servers, and channels between them. Mini-batches, comprising of samples $\langle \text{state}(t,m), \text{state}(t+1,m), \text{action}(t,m), \text{reward}(t,m)\rangle$, are utilized to train the DQN.

The policy of each agent, including the offloading decision, offloading indicator, and association factor, is randomized initially and gradually improved as the training of DQN progresses. During the training phase, experience replay is employed. In each iteration, a mini-batch is sampled from the memory to update the weights of the DQN.

2) Partial Offloading: With given $\mathbf{p}^*, \boldsymbol{\beta}^*, \boldsymbol{\Pi}^*, \mathbf{f}^*$, the partial offloading optimization problem is denoted as:

$$\begin{aligned}\mathbf{P4.1}: \max_{\boldsymbol{\rho}} \ & w_1 \sum_{m\in\mathcal{M}} \left[U_1\left(\lambda_1 \boldsymbol{\pi}^*_m(1)\frac{F_m^{\text{avg}}}{F_m^{\min}}\right) \right. \\ & \left. -U_2\left(\lambda_2 \boldsymbol{\pi}^*_m(2)\frac{T_m^{\text{total}}}{T_m^{\max}}\right)\right] - w_2 \sum_{m\in\mathcal{M}} E_m^{\text{total}} \\ \text{s.t.} \ & C1 \text{ and } \text{C8}.\end{aligned}$$

Note that the objective function of problem $\mathbf{P4.1}$ is non-concave. To address this issue, we use the first-order Taylor expansion to approximate the second term of the objective function as follows:

$$\begin{aligned}&\log\left(\lambda_2\boldsymbol{\pi}^*_m(2)T_m^{\text{total}}\right) = \log\left(\hat{x} + \hat{y}\rho_m\right)\\ &\leq \log\left(\hat{x} + \hat{y}\rho_m^{\tau}\right) + \frac{\hat{y}}{(\hat{x} + \hat{y}\rho_m^{\tau})\ln 10}\left(\rho_m - \rho_m^{\tau}\right) \triangleq \hat{\phi}_m,\end{aligned} \tag{45}$$

where

$$\hat{x} = 1 + \lambda_2(\boldsymbol{\pi}^*_m(2))\frac{C_m}{f_m^{\text{local}}}, \tag{46}$$

$$\begin{aligned}\hat{y} = \lambda_2\boldsymbol{\pi}^*_m(2)&\left[\sum_n a_{m,n}^{\text{edge}}\left(\frac{D_m}{r_{m,j}} + T_{j,n}^{\text{sat}} + \frac{C_m}{f_{m,n}^{\text{edge}}}\right)\right.\\ &\left. + a_m^{\text{cloud}}\left(\frac{D_m}{r_{m,j}} + T_C + \frac{C_m}{f_m^{\text{cloud}}}\right) - \frac{C_m}{f_m^{\text{local}}}\right],\end{aligned} \tag{47}$$

Algorithm 1: Training Process of DQN.

1: Initialize the two DQNs $Q(\boldsymbol{S}, \boldsymbol{a};\theta)$ and $Q'(\boldsymbol{S}, \boldsymbol{a};\theta')$ with random weight θ and θ';
2: Initialize the environment according to the system model;
3: **for** each step t **do**
4:   **for** each user m **do**
5:     Obtain the state $\mathrm{state}(t, m)$;
6:     Calculate the greedy impact $\varepsilon(t)$;
7:     Choose a random probability ω;
8:     **if** $\omega \geq \varepsilon(t)$ **then**
9:       With probability $\varepsilon(t)$ select random action $\mathrm{action}(t, m)$;
10:     **else**
11:       $\mathrm{action}(t, m) = \arg\ \max_a Q(\boldsymbol{S}, \boldsymbol{a};\theta)$;
12:       Update the environment with action: $\mathrm{action}(t, m)$;
13:       Obtain the immediate reward: $\mathrm{reward}(t, m)$;
14:     **end if**
15:     Obtain the next state: $\mathrm{state}(t + 1, m)$;
16:     Store experience $\mathrm{state}(t, m)$, $\mathrm{action}(t, m)$ $\mathrm{reward}(t, m)$, $\mathrm{state}(t + 1, m)$ into the experience memory;
17:   **end for**
18:   Sample a mini-batch of data from the memory;
19:   Train Q and Q' networks by minimizing the loss function $L(\theta)$ and $L(\theta')$ respectively;
20:   Update weights θ and θ' by minimizing the loss function.
21: **end for**

Algorithm 2: Energy-QoS-Aware DQN-Based Hybird Task Offloading and Resource Allocation for Full offloading and Partial Offloading (EDHTR and EDHPTR).

Input: $\mathbf{p}^0$, $\mathbf{f}^0$, $\boldsymbol{\beta}^0$, $\boldsymbol{\rho}^0$, $\boldsymbol{\Pi}^0$ and system parameters.
Output: The optimal $\mathbf{p}^*$, $\mathbf{f}^*$, $\boldsymbol{\rho}^*$, $\boldsymbol{\Pi}^*$ and $\boldsymbol{\beta}^*$.
1: Initialize the iteration number $r = 0$, and the maximum tolerance $\epsilon = 10^{-4}$.
2: Mobile devices randomly generate tasks according to simulation parameters.
3: Model tasks utility functions of all mobile devices based on GMM.
4: **repeat**
5: Use the first layer of the proposed framework to solve the continuous optimization problem.
6: Use CVX to solve problem $\mathbf{P1.3}$ and obtain the optimal solutions $\mathbf{p}^{r+1}$ based on the given $\mathbf{f}^r, \boldsymbol{\beta}^r, \boldsymbol{\rho}^r, \boldsymbol{\Pi}^r$.
7: Use CVX to solve problem $\mathbf{P2.1}$ and obtain the optimal solutions $\mathbf{f}^{r+1}$ based on the given $\mathbf{p}^{r+1}, \boldsymbol{\beta}^r, \boldsymbol{\rho}^r, \boldsymbol{\Pi}^r$.
8: Use the second layer of the proposed framework to solve the discrete optimization problem.
9: According to Algorithm 1 to solve problem $\mathbf{P3.1}$ and obtain the $\boldsymbol{\beta}^{r+1}, \boldsymbol{\rho}^{r+1}, \boldsymbol{\Pi}^{r+1}$ based on the given $\mathbf{p}^{r+1}, \mathbf{f}^{r+1}$.
10: **if** partial offloading (EDHPTR) **then**
11:   Use CVX to solve problem $\mathbf{P4.1}$ and obtain the optimal partial computing offloading factor $\boldsymbol{\rho}^{r+1}$.
12: **end if**
13: **until** The increase of objective value is below a threshold ϵ.

where ρ_m^τ is the Taylor expansion point. The transformed convex optimization problem can be expressed as:

$$\begin{aligned}\mathbf{P4.2}:&\max_{\boldsymbol{\rho}}\ w_1\left[\sum_{m\in\mathcal{M}}\log\left(\lambda_1\boldsymbol{\pi}_m^*(1)\frac{F_m^{\mathrm{avg}}}{F_m^{\min}}\right)-\sum_{m\in\mathscr{H}_d}\hat{\phi}_m\right]\\&\quad-w_2\sum_{m\in\mathcal{M}}E_m^{\mathrm{total}}\\&\text{s.t.}\quad C1,\ C2,\ \text{and C8}.\end{aligned}$$

the objective function of $\mathbf{P4.2}$ is convex and the constraints are all affine. Therefore, it is a standard convex optimization problem and can be solved directly by the CVX toolbox.

The procedure of the proposed task offloading schemes (EDHTR and EDHPTR) is presented in Algorithm 2.

3) Complexity and Convergence Analysis: Let τ_1 be the number of iterations for training the DQN. For each iteration, the complexity of the proposed algorithm can be calculated according to the number of neural network parameters. In each iteration, agents use fully connected neural networks to generate actions. Let L be the number of neural network layers, and let c_l represent the number of neurons in the fully connected layer l. The time complexity for updating the parameters of a fully connected layer is denoted by $T_f = \mathcal{O}(\sum_{l=1}^{J} c_l c_{l-1})$. Therefore, the total time complexity of Algorithm 1 is $\mathcal{O}(\tau_1 T_f)$. Let τ_2 be the number of iterations required to reach the convergence criterion of Algorithm 2. Let ψ be the time complexity of solving other variables through the CVX toolbox. Then, the time complexity of EDHTR algorithm is $\mathcal{O}(\tau_2(\tau_1 T_f + \psi))$.

Verifying the convergence of the proposed DQN method with analytical methods is challenging. In this paper, the proposed algorithm utilizes a convergence-guaranteed iterative implementation of BCD [47]. The convergence of the proposed algorithms is demonstrated by simulation results in Section VI.

VI. Simulation Results

In simulations, we consider an MEC network consisting of 4 LEO satellites at an altitude of 500 km. The orbits of satellites are based on a Walker constellation, consisting of 48 orbital planes, each with 96 satellites. 40 mobile devices are uniformly distributed within a 20 km × 20 km square area. The DQN is a fully connected neural network with 5 layers, 3 of which are hidden layers with 500, 250, and 120 neurons respectively. The activation function is Rectified Linear Unit (ReLU). The initial learning rate is 0.01 and decreases exponentially over time. The neural network is trained with an Adam optimizer.

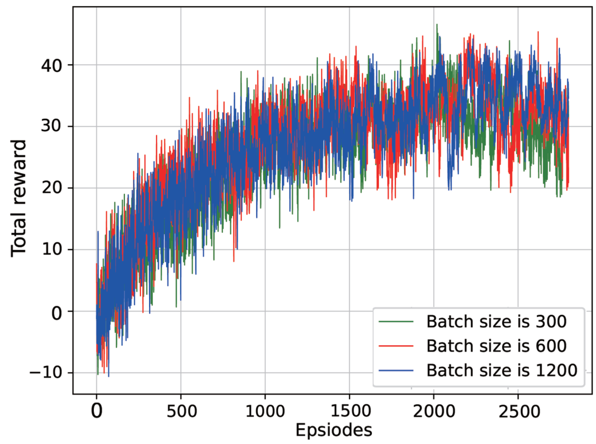


Fig. 5. Reward of all mobile devices versus episodes.

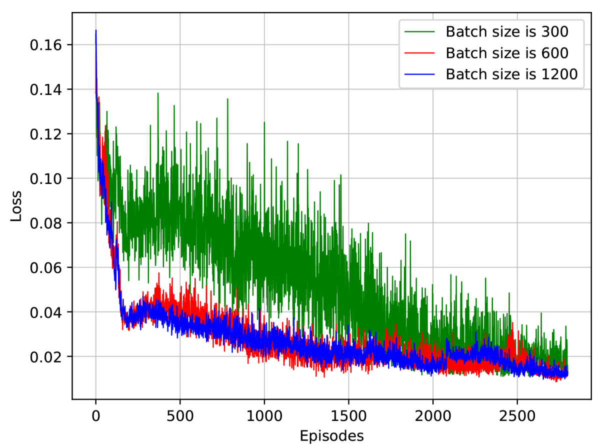


Fig. 6. Training loss versus episodes.

The simulations are conducted using TensorFlow 1.15.4 with Python 3.6 on an Intel Core i7-7700 processor with 16 GB memory.

A. Convergence Performance

The convergence of the proposed algorithms is shown in Fig. 5. The results indicate that the total rewards under all batch sizes first increase and eventually stabilize after around 2000 training episodes. However, when the batch size is 300, the convergence performance of EDHTR is less stable after the 2200th episode, whereas the learning procedure converges more steadily for batch sizes of 600 and 1200.

Fig. 6 presents the accumulated loss during training of the proposed algorithm with different mini-batch sizes. The result indicates that although the impact of batch size on the reward is not significant, it may have a greater impact on the loss. When the batch size is 300, the loss exhibits significant fluctuations during training, but eventually converges after around 2500 episodes. When the batch size exceeds 600, EDHTR exhibits good convergence performance.

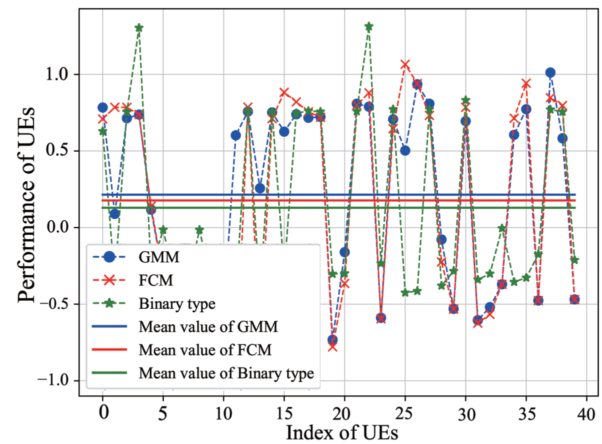


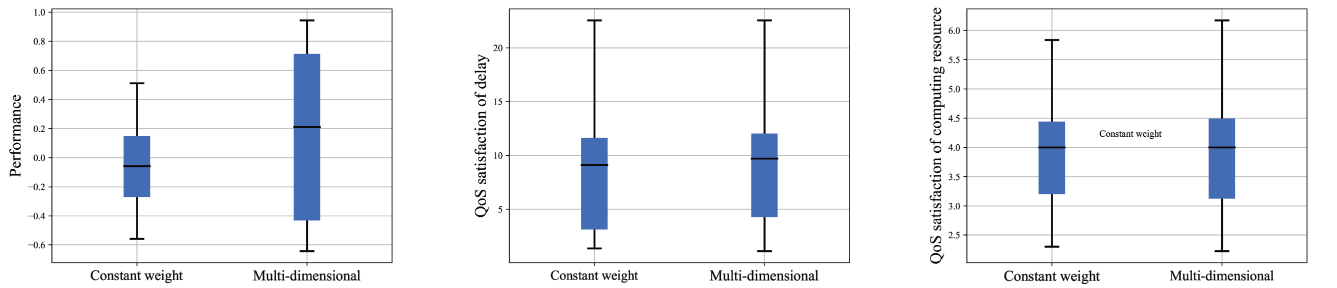
Fig. 7. The influence of classification algorithm on the performance of mobile devices.

B. Performance Under Different Utility Models

To demonstrate the performance of our proposed customized utility function, we compare it with several common utility function models:

1) *FCM-based utility model:* Replace the GMM with the FCM algorithm as the foundation for the utility model. FCM is a partitioning-based clustering algorithm. The basic idea of FCM is to make the similarity between objects divided into the same category as large as possible, and the similarity between objects in different categories as small as possible.
2) *The Binary utility model:* Binary type task utility function can only reflect the satisfaction with delay or computing resources. That is to say, it only considers the utility of tasks from a single dimension, such as only focusing on delay or only focusing on computing resources.
3) *The constant-weight utility model:* A method with predetermined weights for each dimension of resources, which is used in many previous works [48], [49]. It typically allocates a weight of 0.5 to both latency and computational resource dimensions. Although it accounts for task demand diversity, the fixed weights cannot be dynamically adjusted based on task parameters.

Fig. 7 shows the impact of different task utility models. For the Binary type task utility model, the tasks generated by mobile devices are classified as either compute-intensive or delay-sensitive, which is commonly used in the previous literature [50], [51]. The results show that the GMM-based task utility model is better than that of the FCM-based and Binary type. The reason is that GMM employs multiple Gaussian distributions to approximate a model that is close to the actual distribution of task parameters. The FCM algorithm is outperformed by the GMM algorithm since it is not fit for classifying parameters distributed with non-convex shapes, while the distribution of task parameters is highly likely to be non-convex. Thus, the outcome provided by FCM may not accurately reflect the task requirements. In the case of the Binary type, the task utility function can only reflect the level of satisfaction on either delay



(a) Total Utility Performance (b) QoS satisfaction of delay (c) QoS satisfaction of computing resources

Fig. 8. The influence of different task utility models on the QoS satisfaction. (a) Total utility performance. (b) QoS satisfaction of delay. (c) QoS satisfaction of computing resources.

or computation resource. However, most tasks are supported by a combination of multiple types of resources, hence the level of satisfaction is determined by multiple dimensions of resource demand. Therefore, the performance of the Binary type is the worst among all algorithms. This demonstrates the necessity of the multi-dimensional heterogeneous task utility function.

To validate the effectiveness of different task utility models in satisfying various dimensions of QoS metrics/resource demand, we evaluate the "QoS satisfaction" of each dimension. The QoS satisfaction of task m at the delay and computing resource dimension are calculated by $T_m^{\mathrm{total}}/T_{\max}$ and $F_{\mathrm{avg}}/F_{\min}$, respectively.

Fig. 8 shows the QoS satisfaction under different task utility models. The constant-weight utility model has a predetermined weight for each dimension of the resource, which is consistent with the parameter settings in most literature [52], [53]. In the constant-weight utility model, a weight of 0.5 is equally assigned to delay and computing resources. The results in Fig. 8 indicate that the proposed multi-dimensional heterogeneous task utility model outperforms the constant-weight utility model. This advantage stems from the nature of biased task demands, which are harvested by the proposed utility model via allocating higher weights to preferred dimensions. This way, the proposed utility model is more effective in satisfying high-weight requirements when devising task offloading and resource allocation strategies, this is also the reason for the greater variation in performance of the proposed task model. Although the constant weight utility model accounts for task demand diversity, it can not dynamically adjust the weights based on task parameters and thus fails to accommodate the diversified needs of tasks.

C. Different Task Offloading and User Association Strategies

Fig. 9 shows the reward obtained by each mobile device under four task offloading and user association strategies:

- *EDHTR-Random:* Mobile devices randomly select the associated LEO satellite and the offloading destination.
- *EDHTR-Greedy:* Mobile devices employ a greedy approach for task offloading and user association, which is associated with the LEO satellite with the highest transmission rate and offload the task to the MEC server with the most remaining computing resources.

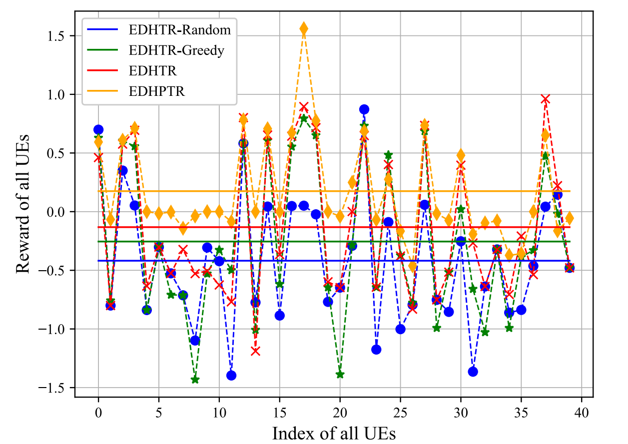


Fig. 9. The performance of different task offloading and user association strategies.

- *EDHTR:* The proposed DRL-based method for full offloading scenario.
- *EDHPTR:* The proposed DRL-based method for partial offloading scenario.

As expected, the EDHTR-Random strategy achieves the lowest reward. The EDHTR-Greedy strategy only achieves a marginal improvement over the EDHTR-Random strategy, which is caused by the imbalanced load distribution among LEO satellites. This imbalance can lead to resource under-utilization or over-utilization, which in turn increases the task completion latency. Furthermore, the greedy offloading strategy does not take into account the load capacity of edge servers, which results in low computing resource utilization. In contrast, the EDHTR strategy fully considers the diversity of task demands by designing customized utility functions and exploiting the advantages of DRL for intelligent decision-making, resulting in better performance over its counterparts. Finally, in partially offloaded scenarios, the EDHPTR strategy demonstrates a marginal improvement over EDHTR, since EDHPTR enables concurrent utilization of computing resources across various servers, resulting in improved computing resource utilization and reduced processing latency. Fig. 10 shows the average performance of all devices under various strategies. The performance of EDHTR-Greedy has the largest deviation among all

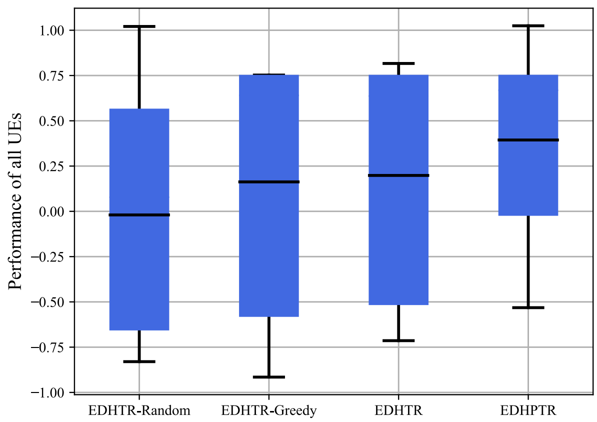


Fig. 10. The performance differences between different mobile devices under various strategies.

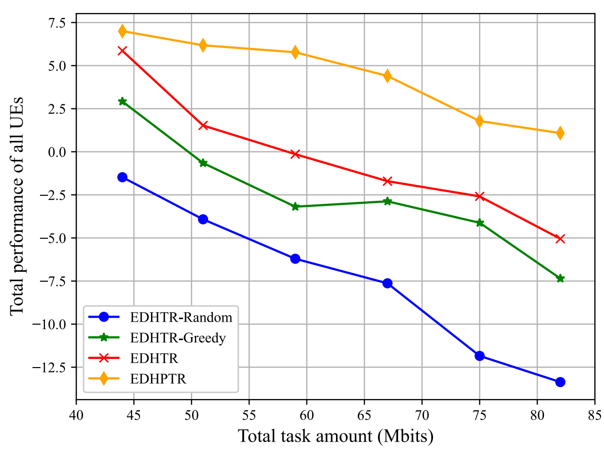


Fig. 11. The impact of task amount on the performance of all mobile devices.

strategies. This results from its "selfish" selection, where some mobile devices may occupy too many resources, leaving insufficient resources for the other devices. On the other hand, the performance differences between EDHPTR and EDHTR algorithms are significantly smaller. This indicates that the proposed algorithms can effectively guarantee the QoS requirements for all mobile devices, while simultaneously enhancing the overall system performance.

The impact of the total number of tasks on the performance of different algorithms is demonstrated in Fig. 11. Simulations are carried out on 10 distinct network topologies, and the outcomes are subsequently averaged. The results demonstrate that with an increase in the number of tasks, there is a gradual decline in the performance of all the algorithms. Since tasks can be processed concurrently on local and remote servers, EDHPTR offers an upper-bound performance for comparison purposes. Moreover, the EDHTR strategy jointly optimizes resource utilization and energy efficiency, resulting in more intelligent user association and efficient resource allocation strategies. Consequently, both EDHTR and EDHPTR exhibit superior performance compared to the other two algorithms. For EDHTR-Greedy, when the size of tasks exceeds 59 Mbits, mobile devices that prioritize

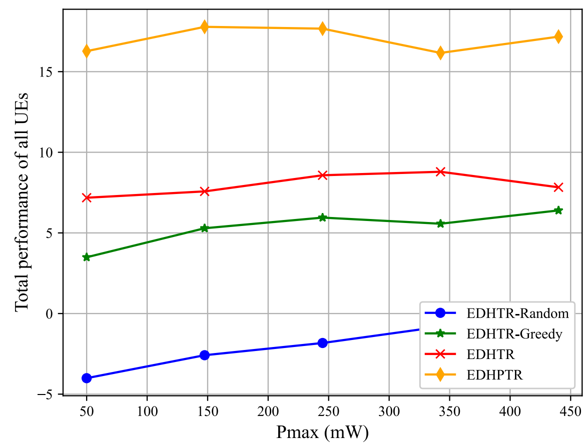


Fig. 12. The influence of transmission power on the performance of all mobile devices.

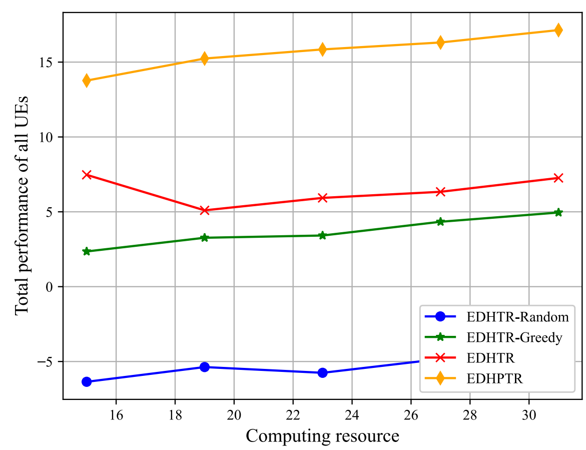


Fig. 13. The impact of computing resources on the performance of all mobile devices.

low latency tend to offload more tasks to MEC servers, aiming to reduce computing delay and energy consumption owing to sufficient computing resources available at MEC. This results in a slight enhancement in EDHTR-Greedy's performance within the range of 59 to 67 Mbits. However, as the size of task data continues to rise, the limited computing resources are split among an increasing number of tasks, resulting in higher processing delay and energy consumption, which is reflected in a decline in overall performance for EDHTR-Greedy. EDHTR-Random exhibits the worst performance since the strategy does not take into account task utility and system resource utilization. The results show that the proposed algorithm can dynamically adapt to varying task loads and achieve joint optimization of resource utilization and energy efficiency. Fig. 12 shows the effect of transmission power on user performance. As can be seen, the increase in power has a negligible effect on performance, because while the increase in power reduces the transmission delay during offloading of the task, it also increases the energy consumption.

Fig. 13 shows the impact of computing resources of MEC servers on user performance. As the amount of computing resources increases, more computing resources are available for computing-intensive tasks, and it also results in less computational delay of delay-sensitive tasks, both of which contribute to increasing task utility and user satisfaction.

VII. Conclusion

In this paper, we investigated the problem of task offloading and resource allocation in SMEC systems with heterogeneous task demands. First, we proposed a GMM-based task utility model to capture the diverse task demands, in which a customized task utility function is derived for every individual task. Next, we formulated the joint optimization of task offloading, power control, computing resource allocation, and user association as an MNLIP, aiming to maximize the sum task utility of all UEs. Then, we proposed a multi-layer iterative framework to solve the MNLIP. Through simulations conducted for both fully offloading and partial offloading scenarios, the proposed schemes demonstrate superior performance compared to several benchmark schemes. In future works, we will explore more comprehensive task utility models with additional dimensions of resource demand and QoS requirement, as well as design customized task offloading and resource allocation strategies for SMEC networks with different sizes and dynamics. Considering that satellite-ground links are less reliable due to long propagation distance and satellite mobility, we will further investigate the scenario where task offloading may be interrupted by link failure.

References

[1] Z. Zhang, W. Zhang, and F.-H. Tseng, "Satellite mobile edge computing: Improving QoS of high-speed satellite-terrestrial networks using edge computing techniques," *IEEE Netw.*, vol. 33, no. 1, pp. 70–76, Jan./Feb. 2019.

[2] R. Xie, Q. Tang, Q. Wang, X. Liu, F. R. Yu, and T. Huang, "Satellite-terrestrial integrated edge computing networks: Architecture, challenges, and open issues," *IEEE Netw.*, vol. 34, no. 3, pp. 224–231, May/Jun. 2020.

[3] J. Fu, J. Hua, J. Wen, K. Zhou, J. Li, and B. Sheng, "Optimization of achievable rate in the multiuser satellite IoT system with SWIPT and MEC," *IEEE Trans. Ind. Inform.*, vol. 17, no. 3, pp. 2072–2080, Mar. 2021.

[4] R. Wang et al., "Collaborative computation offloading and resource allocation in satellite edge computing," in *Proc. IEEE Glob. Commun. Conf.*, 2022, pp. 5625–5630.

[5] F. Chai, Q. Zhang, H. Yao, X. Xin, R. Gao, and M. Guizani, "Joint multi-task offloading and resource allocation for mobile edge computing systems in satellite IoT," *IEEE Trans. Veh. Technol.*, vol. 72, no. 6, pp. 7783–7795, Jun. 2023.

[6] J. Zhang, J. Du, Y. Shen, and J. Wang, "Dynamic computation offloading with energy harvesting devices: A hybrid-decision-based deep reinforcement learning approach," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9303–9317, Oct. 2020.

[7] M. Feng, M. Krunz, and W. Zhang, "Joint task partitioning and user association for latency minimization in mobile edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 70, no. 8, pp. 8108–8121, Aug. 2021.

[8] M. Masoudi and C. Cavdar, "Device vs edge computing for mobile services: Delay-aware decision making to minimize power consumption," *IEEE Trans. Mobile Comput.*, vol. 20, no. 12, pp. 3324–3337, Dec. 2021.

[9] Y. Liu, Y. Li, Y. Niu, and D. Jin, "Joint optimization of path planning and resource allocation in mobile edge computing," *IEEE Trans. Mob. Comput.*, vol. 19, no. 9, pp. 2129–2144, Sep. 2020.

[10] M. Luglio, M. Marchese, F. Patrone, C. Roseti, and F. Zampognaro, "Performance evaluation of a satellite communication-based MEC architecture for IOT applications," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 58, no. 5, pp. 3775–3785, Oct. 2022.

[11] M. Jia, L. Zhang, J. Wu, Q. Guo, and X. Gu, "Joint computing and communication resource allocation for edge computing towards huge LEO networks," *China Commun.*, vol. 19, no. 8, pp. 73–84, Aug., 2022.

[12] H. Liu and G. Cao, "Deep reinforcement learning-based server selection for mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 70, no. 12, pp. 13351–13363, Dec. 2021.

[13] L. Ale, N. Zhang, X. Fang, X. Chen, S. Wu, and L. Li, "Delay-aware and energy-efficient computation offloading in mobile-edge computing using deep reinforcement learning," *IEEE Trans. Cogn. Commun. Netw.*, vol. 7, no. 3, pp. 881–892, Sep. 2021.

[14] X. Qiu, L. Liu, W. Chen, Z. Hong, and Z. Zheng, "Online deep reinforcement learning for computation offloading in blockchain-empowered mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 68, no. 8, pp. 8050–8062, Aug. 2019.

[15] Z. Tong, J. Cai, J. Mei, K. Li, and K. Li, "Dynamic energy-saving offloading strategy guided by Lyapunov optimization for IoT devices," *IEEE Internet Things J.*, vol. 9, no. 20, pp. 19903–19915, Oct. 2022.

[16] R. Lin et al., "Energy-efficient computation offloading in collaborative edge computing," *IEEE Internet Things J.*, vol. 9, no. 21, pp. 21305–21322, Nov. 2022.

[17] Q. Liu, J. Li, R. Zhou, J. Wei, S. Liu, and Q. Li, "Physical-layer security for multiuser computation offloading with Lyapunov optimization," in *Proc. IEEE Veh. Technol. Conf.*, 2022, pp. 1–5.

[18] Z. Yang, C. Pan, K. Wang, and M. Shikh-Bahaei, "Energy efficient resource allocation in UAV-enabled mobile edge computing networks," *IEEE Trans. Wireless Commun.*, vol. 18, no. 9, pp. 4576–4589, Sep. 2019.

[19] F. Wang, W. Chen, H. Tang, and Q. Wu, "Joint optimization of user association, subchannel allocation, and power allocation in multi-cell multi-association OFDMA heterogeneous networks," *IEEE Trans. Commun.*, vol. 65, no. 6, pp. 2672–2684, Jun. 2017.

[20] J. Zhao, Q. Li, Y. Gong, and K. Zhang, "Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 8, pp. 7944–7956, Aug. 2019.

[21] C.-F. Liu, M. Bennis, M. Debbah, and H. V. Poor, "Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing," *IEEE Trans. Commun.*, vol. 67, no. 6, pp. 4132–4150, Jun. 2019.

[22] L. Tan, Z. Kuang, L. Zhao, and A. Liu, "Energy-efficient joint task offloading and resource allocation in OFDMA-based collaborative edge computing," *IEEE Trans Wireless Commun.*, vol. 21, no. 3, pp. 1960–1972, Mar. 2022.

[23] H. Tang, H. Wu, Y. Zhao, and R. Li, "Joint computation offloading and resource allocation under task-overflowed situations in mobile-edge computing," *IEEE Trans. Netw. Serv. Manage.*, vol. 19, no. 2, pp. 1539–1553, Jun. 2022.

[24] Y. Liu, H. Yu, S. Xie, and Y. Zhang, "Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 11, pp. 11158–11168, Nov. 2019.

[25] C. Qiu, H. Yao, F. R. Yu, F. Xu, and C. Zhao, "Deep Q-learning aided networking, caching, and computing resources allocation in software-defined satellite-terrestrial networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 6, pp. 5871–5883, Jun. 2019.

[26] S. Yu, X. Chen, Z. Zhou, X. Gong, and D. Wu, "When deep reinforcement learning meets federated learning: Intelligent multitimescale resource management for multiaccess edge computing in 5G ultradense network," *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2238–2251, Feb. 2021.

[27] Z. Huang, J. Sun, X. Guo, and M. Shang, "Adaptive deep reinforcement learning-based in-loop filter for VVC," *IEEE Trans. Image Process.*, vol. 30, pp. 5439–5451, 2021.

[28] G. Cui, P. Duan, L. Xu, and W. Wang, "Latency optimization for hybrid GEO–LEO satellite-assisted IoT networks," *IEEE Internet Things J.*, vol. 10, no. 7, pp. 6286–6297, Apr. 2023.

[29] Z. Zhao, M. Feng, C. Ke, Z. Chen, and T. Jiang, "Federated deep recurrent Q-learning for task partitioning and resource allocation in satellite mobile edge computing assisted industrial IoT," *IEEE Internet Things J.*, vol. 11, no. 15, pp. 26444–26458, Aug. 2024.

[30] T. K. Rodrigues and N. Kato, "Hybrid centralized and distributed learning for MEC-equipped satellite 6G networks," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 4, pp. 1201–1211, Apr. 2023.

[31] H. Wu, K. Wolter, P. Jiao, Y. Deng, Y. Zhao, and M. Xu, “EEDTO: An energy-efficient dynamic task offloading algorithm for blockchain-enabled IoT-edge-cloud orchestrated computing,” *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2163–2176, Feb. 2021.

[32] X. Yuan, H. Tian, H. Wang, H. Su, J. Liu, and A. Taherkordi, “Edge-enabled WBANs for efficient QoS provisioning healthcare monitoring: A two-stage potential game-based computation offloading strategy,” *IEEE Access*, vol. 8, pp. 92718–92730, 2020.

[33] L. Li, Q. Guan, L. Jin, and M. Guo, “Resource allocation and task offloading for heterogeneous real-time tasks with uncertain duration time in a fog queueing system,” *IEEE Access*, vol. 7, pp. 9912–9925, 2019.

[34] C. He, R. Wang, and Z. Tan, “Energy-aware collaborative computation offloading over mobile edge computation empowered fiber-wireless access networks,” *IEEE Access*, vol. 8, pp. 24662–24674, 2020.

[35] Y. Yang, Z. Liu, X. Yang, K. Wang, X. Hong, and X. Ge, “POMT: Paired offloading of multiple tasks in heterogeneous fog networks,” *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8658–8669, Oct. 2019.

[36] M. Mukherjee et al., “Delay-sensitive and priority-aware task offloading for edge computing-assisted healthcare services,” in *Proc. IEEE Glob. Commun. Conf.*, 2020, pp. 1–5.

[37] K. Aljobory and A. Yazici, “Discrimination on offloading performance in two-class mobile edge computing systems,” in *Proc. 2018 Signal Process. Commun. Appl. Conf.*, 2018, pp. 1–4.

[38] S. M. R. Islam, N. Avazov, O. A. Dobre, and K.-s. Kwak, “Power-domain non-orthogonal multiple access (NOMA) in 5G systems: Potentials and challenges,” *IEEE Commun. Surveys Tut.*, vol. 19, no. 2, pp. 721–742, Secondquarter 2017.

[39] S. Guo and X. Zhao, “Multi-agent deep reinforcement learning based transmission latency minimization for delay-sensitive cognitive satellite-UAV networks,” *IEEE Trans. Commun.*, vol. 71, no. 1, pp. 131–144, Jan. 2023.

[40] L. Li and P. Fan, “Latency and task loss probability for noma assisted MEC in mobility-aware vehicular networks,” *IEEE Trans. Veh. Technol.*, vol. 72, no. 5, pp. 6891–6895, May 2023.

[41] Y. Xu, C. Shen, T.-H. Chang, S.-C. Lin, Y. Zhao, and G. Zhu, “Transmission energy minimization for heterogeneous low-latency NOMA downlink,” *IEEE Trans. Wireless Commun.*, vol. 19, no. 2, pp. 1054–1069, Feb. 2020.

[42] Y. Chen, B. Ai, Y. Niu, H. Zhang, and Z. Han, “Energy-constrained computation offloading in space-air-ground integrated networks using distributionally robust optimization,” *IEEE Trans. Veh. Technol.*, vol. 70, no. 11, pp. 12113–12125, Nov. 2021.

[43] M. Tang and V. W. Wong, “Deep reinforcement learning for task offloading in mobile edge computing systems,” *IEEE Trans. Mobile Comput.*, vol. 21, no. 6, pp. 1985–1997, Jun. 2022.

[44] M. Feng and M. Krunz, “Program placement optimization for storage-constrained mobile edge computing systems: A multi-armed bandit approach,” in *Proc. IEEE Int. Symp. World Wireless, Mobile Multimedia Netw.*, 2021, pp. 149–158.

[45] G. Xuan, W. Zhang, and P. Chai, “EM algorithms of Gaussian mixture model and hidden Markov model,” in *Proc. Int. Conf. Image Process.*, 2001, vol. 1, pp. 145–148.

[46] X. Ma, S. Wang, S. Zhang, P. Yang, C. Lin, and X. Shen, “Cost-efficient resource provisioning for dynamic requests in cloud assisted mobile edge computing,” *IEEE Trans. Cloud Comput.*, vol. 9, no. 3, pp. 968–980, Jul.–Sep. 2021.

[47] P. Tseng, “Convergence of a block coordinate descent method for nondifferentiable minimization,” *J. Optim. Theory Appl.*, vol. 109, pp. 475–494, 2001.

[48] Y. M. Chee, F. Gao, H. M. Kiah, A. C. H. Ling, H. Zhang, and X. Zhang, “Decompositions of edge-colored digraphs: A new technique in the construction of constant-weight codes and related families,” in *Proc. IEEE Int. Symp. Inf. Theory*, 2014, pp. 1436–1440.

[49] H. Zhang and G. Ge, “Optimal ternary constant-weight codes of weight four and distance six,” *IEEE Trans. Inf. Theory*, vol. 56, no. 5, pp. 2188–2203, May 2010.

[50] B. Atan, M. Basaran, N. Calik, S. T. Basaran, G. Akkuzu, and L. Durak-Ata, “AI-empowered fast task execution decision for delay-sensitive IoT applications in edge computing networks,” *IEEE Access*, vol. 11, pp. 1324–1334, 2023.

[51] W. Han, J. Su, S. Lv, P. Zhang, and X. Li, “Task offloading strategies for cloud-side cooperation in compute-intensive scenarios based on edge computing,” in *Proc. IEEE 21st Int. Symp. Commun. Inf. Technol.*, 2022, pp. 148–153.

[52] L. P. Qian, B. Shi, Y. Wu, B. Sun, and D. H. K. Tsang, “NOMA-enabled mobile edge computing for Internet of Things via joint communication and computation resource allocations,” *IEEE Internet Things J.*, vol. 7, no. 1, pp. 718–733, Jan. 2020.

[53] X. Chen, Y. Cai, Q. Shi, M. Zhao, B. Champagne, and L. Hanzo, “Efficient resource allocation for relay-assisted computation offloading in mobile-edge computing,” *IEEE Internet Things J.*, vol. 7, no. 3, pp. 2452–2468, Mar. 2020.