Journal of Systems Architecture 165 (2025) 103433

Contents lists available at ScienceDirect

Journal of Systems Architecture

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

MADDPG-based task offloading and resource pricing in edge collaboration environment

Zhao Tong [a],*, Xin Deng [a], Yuanyang Zhang [a], Jing Mei [a], Can Wang [a], Keqin Li [b,c]

[a] College of Information Science and Engineering, Hunan Normal University, Changsha, 410012, China
[b] College of Information Science and Engineering, Hunan University, and National Supercomputing Center, 410082, Changsha, China
[c] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

ARTICLE INFO

Keywords:
Mobile edge computing
Resource pricing
Stackelberg game
Task offloading

ABSTRACT

With the rapid advancement of fifth-generation communication technologies, the data produced by the Internet of Everything is growing exponentially. As mobile cloud computing struggles to keep up with the demands for massive data processing and low latency, mobile edge computing (MEC) has emerged as a solution. By shifting services from centralized cloud platforms to edge servers located closer to data sources, MEC achieves reduced latency, enhanced computing efficiency, and an improved user experience. This paper introduces a task offloading algorithm designed for a multi-base station cooperative mobile edge environment, addressing the challenges of task offloading and resource pricing. The system architecture includes a macro base station and several micro base stations, strategically deployed in a densely populated mobile device area. Each mobile device serves as an autonomous decision-making unit, offloading tasks to an optimal base station. We model the interactions between base stations and end-users using a Stackelberg game approach, with strategy optimization achieved through a multi-agent deep deterministic policy gradient algorithm. The proposed TO-SG-MADDPG algorithm intelligently coordinates the policies of multiple base stations and end-users by centralized training and distributed execution, resulting in globally optimal task offloading and resource pricing. The results demonstrate that the proposed algorithm not only reduces the task loss rate but also safeguards the interests of all stakeholders.

1. Introduction

As technologies like big data and blockchain evolve, an explosion in the number of mobile devices connected to the Internet is occurring [1,2]. In this dynamic network environment, data generated by the internet of everything (IoE) is growing exponentially. This surge in digital information is driving rapid technological advancements across various sectors, particularly transforming urban living [3]. The integration of smart homes with digital lifestyles is crafting a new paradigm for urban residents, reshaping everyday experiences and interactions. Cloud computing has emerged as a powerful computing model that provides an efficient solution for processing and storing large-scale data. It provides users with flexible and scalable services by centralizing the management of computing resources [4]. As a centralized data processing model, cloud computing solves the problem of big data processing and application requirements to a certain extent. However, this integration also poses challenges, particularly regarding data transmission delays and increased network congestion [5]. The distance between cloud computing data centers and user terminals often results in notable transmission delays, which become particularly prominent in applications requiring immediate responses. Additionally, the concentration of vast amounts of data processing in the cloud exacerbates network bandwidth strain. This leads to frequent congestion during peak hours, further impacting the efficiency and stability of data transmission.

To address the above challenge, mobile edge computing (MEC) emerges at the historic moment. MEC technology shifts the computational tasks from the cloud to the edge nodes near the source of data generation for processing [6], thereby effectively mitigating the issues associated with cloud computing. In modern residential life, end-users (EUs) have increased demand for instantaneous and personalized computing [7]. By deploying macro base stations (Ma-BSs) or multiple micro base stations (Mi-BSs) in residential areas, MEC can provide faster and more reliable computing services for multiple EUs. This meets the diverse needs of EUs for smart homes, video surveillance, social entertainment, etc. The advantages of MEC over traditional cloud

* Corresponding author.
E-mail addresses: tongzhao@hunnu.edu.cn (Z. Tong), 202120293782@hunnu.edu.cn (X. Deng), 202220294007@hunnu.edu.cn (Y. Zhang), jingmei1988@163.com (J. Mei), wangcan@hunnu.edu.cn (C. Wang), lik@newpaltz.edu (K. Li).

https://doi.org/10.1016/j.sysarc.2025.103433
Received 19 November 2024; Received in revised form 12 April 2025; Accepted 28 April 2025
Available online 15 May 2025

computing are highlighted in three key fields. Firstly, by processing tasks on servers located closer to users, MEC significantly reduces the latency associated with task transmission. Secondly, offloading tasks to the edge of the network significantly reduces the amount of data that must be transmitted to central cloud servers, thereby alleviating network bandwidth congestion. Finally, by processing tasks locally on edge devices, MEC minimizes the risk of privacy breaches that could occur when sensitive data is transmitted across the network [8]. In particular, as sensitive user data is processed and stored locally rather than being uploaded to centralized cloud servers, MEC reduces the attack surface for potential data breaches [9,10].

Despite its significant advantages in reducing latency and enhancing privacy protection, MEC faces several critical challenges, including efficient task offloading, reliable cloud execution, and ensuring consistent service delivery [11]. Among these, task offloading is a core component, aiming to optimize the execution location of computing tasks to improve system performance and quality of experience (QoE). Task offloading involves migrating tasks from end users (EUs) to base stations (BSs), but the diversity of tasks complicates this process. Different tasks have varying requirements: for example, video surveillance involves computation-intensive tasks, virtual and augmented reality are latency-sensitive, and simple operations like text processing require minimal resources and low response times. This diversity necessitates intelligent resource allocation to meet the varying demands of EUs.

Another key issue is service monetization, including pricing and incentive mechanisms for computing resources [12]. Resource pricing and task offloading in edge systems are particularly challenging due to task heterogeneity, decentralized server placement, and competition for resources among EUs [13]. EUs aim to obtain maximum computational resources at the lowest cost and prefer offloading to the nearest BS with low latency, while BSs, constrained by limited computational capacity, seek to maximize profits. For instance, Tong et al. [14] proposed a Stackelberg game-based pricing model to address the profit conflict between EUs and BSs in a single-server scenario with homogeneous tasks [15]. However, this approach struggles to adapt to heterogeneous or dynamic environments. Although the Stackelberg game assumes rational decision-making, it provides a practical abstraction that simplifies complex interactions while remaining analytically tractable. This makes it a widely used and effective framework for modeling resource allocation and task offloading in MEC systems. To address this limitation, our work investigates a more complex and dynamic MEC system in a residential area, involving heterogeneous computing resources (Ma-BS and Mi-BS) and multiple EUs with diverse computing requirements. To balance resource allocation and enhance the network's computational capacity, we consider cooperative task offloading between Ma-BSs and Mi-BSs.

To coordinate the conflicting interests of EUs and BSs, we introduce the Stackelberg game as an effective tool for modeling the leader-follower relationship. Specifically, Ma-BSs and Mi-BSs act as leaders, setting resource prices, while EUs act as followers, responding with optimal offloading strategies. The goal is to achieve a system-wide equilibrium that maximizes utility for both parties. By leveraging deep reinforcement learning, our approach uniquely combines the Stackelberg game and the MADDPG algorithm to model and solve this complex task offloading problem. This combination explicitly considers task heterogeneity by classifying tasks based on latency requirements and task size. Tasks are optionally offloaded to Ma-BSs or neighboring Mi-BSs, which are configured with energy harvesting devices to facilitate sustainable task execution. Moreover, by integrating energy harvesting in Mi-BSs, our approach enables efficient and sustainable task processing. Experimental results demonstrate the effectiveness of this method in reducing both latency and task loss rate. The contributions of this paper can be concluded as follows:

- The study focuses on the task offloading problem in a residential dense scenario, involving multiple Ma-BS, Mi-BSs, and numerous EUs. In such a scenario, computational resources are shared among BSs and work collaboratively to provide services to EUs. Based on real-time state information observed, EUs autonomously formulate and execute their offloading strategies.
- Considering the heterogeneity of tasks, the tasks randomly generated by EUs are categorized according to the latency requirement and task size. Tasks can be optionally offloaded to a Ma-BS or neighboring Mi-BSs, which are configured with energy harvesting devices that can facilitate task execution by harvesting green energy.
- The relationship between Ma-BS, Mi-BSs and EUs is formulated as a Stackelberg game. The agents of BSs as the leader first determine the unit price of the resource, and the EUs as the followers then decide the offloading location based on the resource price. Both the leader and the followers know only incomplete information about the environment and are committed to optimizing their long-term average economic efficiency.
- The optimization problem is transformed into a Markov decision process (MDP) and then solved by a multi-intelligence deep deterministic gradient policy algorithm. The innovative integration of the MADDPG algorithm verifies that the proposed algorithm is suitable for performing latency-sensitive tasks and is effective in reducing the latency and task loss rate of the system.

The organization of the rest sections of this paper is shown below. Section 2 demonstrates the related work on task offloading for edge computing both nationally and internationally. Model definitions and relevant assumptions are described in Section 3. Section 4 presents the task offloading mechanism based on the Stackelberg game. Section 5 demonstrates our proposed task offloading and resource pricing algorithm. In Section 6, we perform simulation experiments and analyze the results. The conclusion of the paper and future perspectives are presented in the last Section.

2. Related work

Task offloading is a pivotal research area in MEC due to its directly related to the performance enhancement of end-users and the improvement of user experience [16]. This section presents an analysis of research related to task offloading from the perspective of multi-type system architectures.

2.1. MEC systems with single server and single user

Mao et al. [17] proposed low-complexity algorithms to minimize the weighted sum of execution delay and device energy consumption. This problem addresses joint task offloading and transmission power allocation in MEC systems with multiple independent tasks. Chen et al. [18] considered an edge offloading scenario where a single end-user operates a multitude of applications in conjunction with a base station. They introduced an optimization problem aimed at the synergistic allocation of tasks and the strategic adjustment of CPU cycle frequencies, thereby enhancing the efficiency and performance of resource management in edge computing environments. Mao et al. [19] investigated the computational offloading problem in green MEC systems with harvestable energy and proposed an online algorithm based on Lyapunov optimization with low complexity.

Although single-server and single-user research provides a starting point for understanding the fundamental issues of MEC task offloading, research conducted on single-server and single-user system frameworks has limitations. Such simplified models do not accurately reflect the complexities of real-world environments. For instance, they do not account for the variability of network conditions, resource competition among multiple users, or synergy issues among servers. These shortcomings highlight the need for more sophisticated models and approaches that can better capture the dynamics and intricacies of multi-user and multi-server MEC systems.

2.2. MEC systems with single server and multiple users

To address above shortcomings, researchers have started to work on single-server and multi-user systems [20–25]. In a dynamic MEC environment under single-server-multi-user, Zhou et al. [20] proposed a value iterative reinforcement learning method to determine the joint offloading policy to minimize the average long-term energy consumption in the system. To avoid dimensional catastrophe, the DDQN-based offloading algorithm is further proposed. To mitigate the significant dropout effect in federated learning, Ji et al. [21] proposed an edge-assisted federated learning. The scheme helps the client train the model by enabling the dropouts to transfer part of the computation to the edge server. Experimental results demonstrate that the method effectively optimizes the size of the offloaded data, significantly reduces the system latency, and exhibits more outstanding performance compared to traditional federated learning. To address the ongoing challenge of insufficient battery capacity in IoT devices, Chen et al. [22] introduced a comprehensive mixed energy supply model designed to co-optimize local computing, offloading continuity, and edge computing decisions. They designed an online dynamic offloading algorithm based on stochastic theory, which can effectively reduce the system cost. In order to improve the utilization efficiency of computing resources, Seo et al. [23] broke through the traditional uniform pricing model and proposed a differentiated pricing algorithm that determines the unit price of resources through the server's computing resource usage. Moreover, the relationship between the server and followers is modeled as a Stackelberg game, and the equilibrium strategy is obtained using supervised learning.

Although these studies advance the understanding of task offloading in multi-user systems, they often overlook the challenges related to task heterogeneity and do not adequately address the complexity of collaborative edge resource management.

2.3. MEC systems with multiple servers and multiple users

As the number of EUs increases, a single server may face a performance bottleneck, while multiple servers can share the load and increase the concurrent processing capacity of the system. Liu et al. [26] considered a three-tier MEC system that includes several smart devices with randomly generated tasks, an edge server, and a cloud server that acts as a secondary offload. They presented a task offloading scenario that seeks to minimize average latency, then derived and solved it using Lyapunov optimization and duality theory. Similarly, Bahreini et al. [27] formulated the resource allocation problem in a three-tier system as a mixed-integer nonlinear problem and solved it with an envy-free auction mechanism, showing improvements in social welfare and revenue. Pang et al. [28] introduced an ID-assisted computational model to minimize user costs and maximize server and ID profits, addressing conflicts of interest and outperforming comparison algorithms in response time and energy consumption. Mitsis et al. [29] proposed a dynamic price-aware MEC model to derive optimal offloading strategies and service prices for MEC servers, using prospect theory to capture user satisfaction. While comprehensive, this model does not account for latency constraints. Xiao et al. [30] introduced a priority-based scheduling strategy for IoT applications, using deep reinforcement learning to optimize task offloading while controlling energy consumption.

In contrast to the above existing research, this work investigates task offloading strategies in a residential dense edge environment. Collaboration among edge servers to enhance resource utilization efficiency is fully considered. A Stackelberg game model is developed to ensure the economic efficiency of BSs and EUs. At the same time, delay and energy consumption are reduced. Unlike traditional single-server or single-task models, our work accounts for the heterogeneity of tasks in practical MEC scenarios. Based on this framework, this work presents a task offloading and pricing scheme for MEC collaborative environments, achieving a more realistic representation of diverse scenarios and improving upon single-task models.

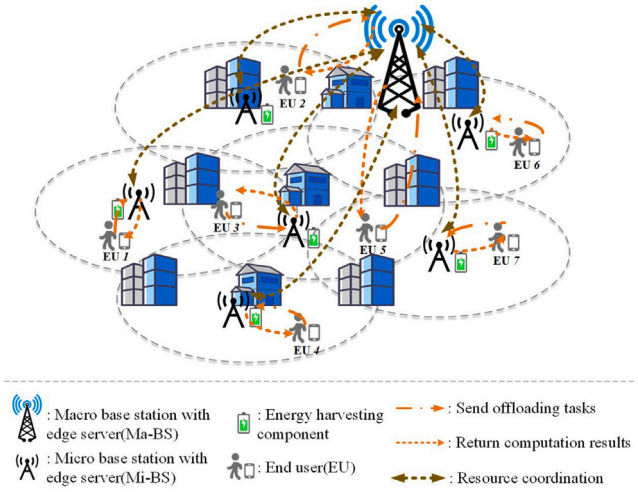

Fig. 1. Task offloading architecture under the multi-BS collaborative MEC system. Multiple EUs are offloading computational tasks to Ma-BS or a Mi-BS, and both parties are playing for their own optimal benefit.

3. Model description and assumptions

This section introduces the task offloading model tailored for the MEC server collaborative computing environment, detailing both the systemic framework and the operational mechanisms of task offloading.

3.1. MEC system model

This paper investigates the problem of task offloading in a multi-BS multi-EU MEC environment, in which a Ma-BS with abundant computational resources and a number of auxiliary computational Mi-BSs together form a computational network. The proposed MEC system architecture is shown in Fig. 1. Both Ma-BS and Mi-BSs are installed with computationally capable MEC servers that supply computational and caching resources within their network coverage. Each BS is also fitted with an energy harvesting device that relies on renewable energy sources for computing. In the following discussion, unless explicitly stated, Ma-BS and Mi-BSs represent the MEC servers connected to them, respectively. During the task offloading process, orthogonal frequency division multiple access (OFDMA) technique is used to divide the spectrum into overlappable subcarriers. The orthogonality between subcarriers is utilized to achieve simultaneous transmission between multiple EUs. The total channel bandwidth of each BS is shared by all the EUs offloaded to the BS. Based on the task requirements, the offloading strategy is developed to meet the EUs' needs. Let Δt denotes the length of each time slot. The main parameters of this paper are listed in Table 1.

For edge service providers, the number of Mi-BSs is denoted by m and the set of Mi-BSs is $\mathcal{M} = \{1, 2, \ldots, j, \ldots, m-1, m\}$. Ma-BS and Mi-BSs will provide network bandwidth and computation within a certain range, and the network coverage information of Ma-BS is denoted by $\{\beta_0, \gamma_0\}$. Its network coverage is a circle with β_0 as the center and γ_0 as the radius, and all the EUs within the circle can offload the generated tasks to the Ma-BS for computation. It is assumed that all EUs are located within the network coverage of the Ma-BS. Similar to the Ma-BS, the network coverage information of the jth Mi-BS is denoted as $\{\beta_j, \gamma_j\}$. Moreover, denote the set of EUs in the network by $\mathcal{N} = \{1, 2, \ldots, i, \ldots, n-1, n\}$. Assume that the EUs do not possess computational capabilities and need to offload the tasks entirely, with no computation performed locally. End-users (EUs) transmit their task attributes (e.g., size, latency tolerance) and real-time network state information (e.g., channel gain, BS availability) to the nearest BS or the

Table 1
Definition of notations.

Symbol	Definition
$\mathcal{M}$	The set of Mi-BSs
$\mathcal{N}$	The set of EUs
f	The computational power possessed by the BS
B	The channel bandwidth
I	The average interference power of the BS
g_i	The task generation symbol for EU i
q_i	The size of the tasks generated by EU i
c_i	The average number of CPU cycles required for 1-bit EU i's task
τ_i	The maximum task latency tolerance of EU i
α_i	The type indication of the task generated by EU i
qos_i	The task priority symbol of EU i
r_i	The task transmission rate from EU i to BS
d_i^{tr}	The task transmission delay from EU i to BS
e_i^{tr}	The task transmission energy consumption from EU i to BS
d_i^{exe}	The task execution delay of EU i at the BS
e_i^{exe}	The task execution energy consumption of EU i at the BS
σ^{exe}	The penalty for a BS's execution of task time-outs
Γ	The penalty for a BS's task being lost
G^L	The utility function of the leader
G_i^F	The utility function of the ith follower
N_i	The signaling coverage of EU i
P_i	The transmission power of EU i
Q^e	The available energy for the BS

BS with the strongest channel gain within their coverage. This initial transmission enables the selected BS to execute the pre-trained TO-SG-MADDPG model and determine the optimal offloading destination (Ma-BS or Mi-BS). Once the decision is made, the EU directly offloads the task to the target BS. The initial transmission of task metadata (e.g., task size, QoS requirements) incurs negligible overhead compared to the actual task data transmission, as metadata is small in size. Consequently, the model focuses on the energy and latency costs of offloading the full task to the final BS, as formulated in Eqs. (3)–(5). The signaling coverage of EU i at time slice t is indicated by $N_i(t)$. If EU i is within the signaling range of the Ma-BS only, then $N_i(t) = \emptyset$. Instead, $N_i(t) = \{m, n\}$ denotes that EU i is within the coverage range of the Ma-BS, Mi-BS m and Mi-BS n.

In general, the process of offloading tasks from EU i to Ma-BS or Mi-BS j is divided into three phases. First, the EU offloads the generated task to the Ma-BS/Mi-BS j over the wireless channel. Then, the Ma-BS and Mi-BS j caches and executes the task. Finally, the corresponding server returns the computed results to EU i. Similar to [31], this paper omit the delay in returning the results since the returned result data is much smaller compared to the offloaded tasks. Next, the task model, task offloading to Ma-BS and Mi-BSs are described in detail, respectively.

3.2. End-user task model

Assume that the tasks generated by the EUs are all independently and identically distributed. For clarity in modeling and analysis, at each time slot, each EU randomly generates tasks according to the Bernoulli distribution obeying the parameter $\lambda \in [0, 1]$. $g_i(t) \in \{0, 1\}$ $(i \in N)$ denotes the task arrival indicator, where $g_i(t) = 1$ denotes that EU i generates a task at moment t and $g_i(t) = 0$ indicates that EU i does not generate any task at moment t. Despite the statistical similarity of tasks, the network conditions experienced by each EU, such as channel quality, proximity to different BSs, and BS load, vary significantly. This necessitates individual offloading strategies to optimize performance.

The task attributes of the EUs are represented by the quintuple $task_i(t) = (q_i(t), c_i(t), \tau_i(t), \alpha_i(t), qos_i(t))$, where the workload characteristics are represented at the bit level to reflect both data transmission and computational demands. $q_i(t)$ denotes the data size of the task generated by the EU i. $c_i(t)$ is the number of CPU cycles required to execute 1 bit of that task, providing a direct measure of the computational cost that aligns with the data-centric nature of MEC tasks. Furthermore, $\tau_i(t)$ denotes the delay tolerance of the EU i for that task, *i.e.* , the maximum value of the execution delay. In addition, the task type flag is denoted as $\alpha_i(t)$. The $qos_i(t)$ indicates the priority level of the task, which divides the task into two categories: high-priority tasks and low-priority tasks. When the task has high priority, make $qos_i(t) = 1$, otherwise, $qos_i(t) = 0$. Due to diverse user scenarios, multiple types of tasks are considered. The parameter settings for each task type are as follows,

$$\alpha_i(t) = \begin{cases} 0, & \text{generate a standard task,} \\ 1, & \text{generate a delay-sensitive task,} \\ 2, & \text{generate a computation-intensive task.} \end{cases}$$

When the task generated by EU i at moment t is a standard task, parameters such as task size and CPU consumption are set with reference to [32], and $qos_i(t) = 0$. Similarly, when the generated task is a delay-sensitive task, making $qos_i(t)$ equal to 1. When a computation-intensive task is generated, its values of task size and CPU consumption are larger. At each time slot t, EU i offloads the generated task to the Ma-BS or a neighboring Mi-BS for computation. The task going to be executed at time slot t of EU i is denoted by $a_i(t)$ as

$$a_i(t) = \begin{cases} 0, & \text{offload task to the Ma-BS,} \\ j, & \text{offload task to the Mi-BS } j. \end{cases}$$

When the task is offloaded to the Ma-BS, then $a_i(t) = 0$. Alternatively, $a_i(t) = j$ means that the task will be offloaded to the Mi-BS j for execution. Correspondingly, the task offloading indicators for the Ma-BS and Mi-BS are defined as:

$$\begin{cases} \mathbf{1}_{a_i(t)=0} = 1, & \text{offload task to the Ma-BS,} \\ \mathbf{1}_{a_i(t)=j} = 1, & \text{offload task to the Mi-BS } j. \end{cases} \tag{1}$$

These indicators are essential for distinguishing the task offloading decisions between the Ma-BS and Mi-BS and are used in the task execution and transmission delay models.

3.3. Task offload to edge servers (Ma-BS and Mi-BSs)

The EUs within the coverage of the wireless network can offload tasks to the Ma-BS or Mi-BS. The MEC servers deployed on the Ma-BS and Mi-BSs collaboratively provide computational resources to the EUs based on the network environment. In this section, we describe the generalized communication model and task execution model, which apply to both Ma-BS and Mi-BSs. The differences between them are highlighted where necessary.

3.3.1. Communication model

The initial transmission of task metadata (e.g., QoS requirements, task size) to the decision-making BS is assumed to occupy minimal bandwidth and energy due to its small data volume. Thus, its overhead is negligible compared to the subsequent task offloading phase. This simplification ensures tractability while preserving the model's focus on optimizing the dominant costs of task execution and transmission. Assume that the number of tasks offloaded to the edge servers at time t is denoted as $V_b(t)$ for the Ma-BS and $V_j(t)$for the Mi-BS. For the sake of generality, we set the channel as a Gaussian white noise channel, which is a classical model that captures the primary noise characteristics in wireless communication. This choice provides a controlled environment for theoretical analysis. Denote the channel gain at time slice t as $g_b(t)$ for the Ma-BS and $g_j(t)$ for the Mi-BS. To minimize transmission overhead, EU i selects the nearest BS or the BS with the strongest

channel gain $g_b(t)$ or $g_j(t)$, based on real-time network conditions. Then the task transmission rate of EU i offloaded to the edge server is

$$\begin{cases} r_{i,b}(t) = \dfrac{B_b}{|V_b(t)|} \cdot \log_2\left(1 + I_b^{-1} \cdot P_{i,b} \cdot g_b(t)\right), \\ r_{i,j}(t) = \dfrac{B_j}{|V_j(t)|} \cdot \log_2\left(1 + I_j^{-1} \cdot P_{i,j} \cdot g_j(t)\right). \end{cases} \tag{2}$$

where $\|V_b(t)\|$, $\left\|V_j(t)\right\|$ is the number of tasks offloaded to the Ma-BS, Mi-BS. B_b and B_j represent the channel bandwidths of the Ma-BS and Mi-BS, respectively. Similarly, I_b and I_j presents the average interference power of the edge server, capturing overall interference effects while simplifying the model by avoiding explicit modeling of all interference sources. $P_{i,b}$ and $P_{i,j}$ is the transmission power of EU i for the Ma-BS and Mi-BS, respectively.

Then, the transmission delay of the EU i offloading task to the edge server is

$$\begin{cases} d_{i,b}^{tr}(t) = \dfrac{q_i}{r_{i,b}(t)} \cdot \mathbf{1}_{a_i(t)=0}, \\ d_{i,j}^{tr}(t) = \dfrac{q_i}{r_{i,j}(t)} \cdot \mathbf{1}_{a_i(t)=j}. \end{cases} \tag{3}$$

Subsequently, the energy consumption of EU i during the transmission process can be obtained as

$$\begin{cases} e_{i,b}^{tr}(t) = d_{i,b}^{tr}(t) \cdot P_{i,b}, \\ e_{i,j}^{tr}(t) = d_{i,j}^{tr}(t) \cdot P_{i,j}. \end{cases} \tag{4}$$

Eventually, the total transmission delay d_b^{tr} of the EUs offloaded to the Ma-BS, d_j^{tr} of the EUs offloaded to the Mi-BS can be obtained as

$$\begin{cases} d_b^{tr}(t) = \max\limits_{i \in V_b(t)} d_{i,b}^{tr}(t), \\ d_j^{tr}(t) = \max\limits_{i \in V_j(t)} d_{i,j}^{tr}(t). \end{cases} \tag{5}$$

3.3.2. Task execution model

Assume that once when all the tasks arrive at the edge server, the server starts to perform the execution of the tasks. The server first sorts the arriving tasks and then executes them in order. The details are described below.

After the EU i offloads a task to the edge server, the task execution time for a device offloading a task to Ma-BS or Mi-BS is given by

$$\begin{cases} d_{i,b}^{exe}(t) = \dfrac{c_i \cdot q_i}{f_b} \cdot \mathbf{1}_{a_i(t)=0}, \\ d_{i,j}^{exe}(t) = \dfrac{c_i \cdot q_i}{f_j} \cdot \mathbf{1}_{a_i(t)=j}, \end{cases} \tag{6}$$

where f_b and f_j denotes the computational power possessed by the MEC server of Ma-BS and Mi-BS, respectively, which is equally distributed among the users.

It is assumed that all EUs send tasks simultaneously in each time slot. For both Ma-BS and Mi-BSs, tasks are categorized and managed based on their characteristics and quality of service (QoS) requirements.

For standard and computation-intensive tasks, a queue $Q_{b,qos=0}^{tr}(t)$ for the Ma-BS and $Q_{j,qos=0}^{tr}(t)$ for the Mi-BS j is generated by ordering the task arrival times according to their transmission delays. Similarly, delay-sensitive tasks are sorted based on the increasing order of their transmission delays, forming queues $Q_{b,qos=1}^{tr}(t)$ for the Ma-BS and $Q_{j,qos=1}^{tr}(t)$ for the Mi-BS j.

These two queues for each base station are then merged, with the delay-sensitive tasks $Q_{b,qos=1}^{tr}(t)$ and $Q_{j,qos=1}^{tr}(t)$ being prioritized and inserted before the standard tasks $Q_{b,qos=0}^{tr}(t)$ and $Q_{j,qos=0}^{tr}(t)$, respectively. This process results in new queues $Q_b^{tr}(t)$ for the Ma-BS and $Q_j^{tr}(t)$ for the Mi-BS j.

The edge servers, both Ma-BS and Mi-BSs, execute tasks sequentially according to their respective queues $Q_b^{tr}(t)$ and $Q_j^{tr}(t)$, ensuring that latency-sensitive tasks are prioritized for execution. If a task $i \in V_b(t)$ (for Ma-BS) or $i \in V_j(t)$ (for Mi-BS j) is the k-th task in its queue, the delay $d_{i,b|i\in V_b(t)}(t)$ or $d_{i,j|i\in V_j(t)}(t)$ represents the time the task waits for execution and the sum of the execution times of the previous k tasks at the respective base station.

Denote the penalty for performing a task beyond the tolerance latency constraint at time slot t as $\sigma_b^{exe}(t)$ when tasks are offloaded to Ma-BS, and as $\sigma_j^{exe}(t)$ when tasks are offloaded to Mi-BS. Let L denote the set of tasks that are not completed on time, where there are l tasks. Then $\sigma_b^{exe}(t)$ and $\sigma_j^{exe}(t)$ can be denoted as

$$\begin{cases} \sigma_b^{exe}(t) = \sum\limits_{i=1}^{l} \mathbf{1}_{d_{i,b}(t)+d_b^{tr}(t)>\tau_i(t)|a_i(t)=0}, \\ \sigma_j^{exe}(t) = \sum\limits_{i=1}^{l_j} \mathbf{1}_{d_{i,j}(t)+d_j^{tr}(t)>\tau_i(t)|a_i(t)=j}. \end{cases} \tag{7}$$

where the value of $\mathbf{1}$ is 1 if the condition $d_{i,b}(t)+d_b^{tr}(t) > \tau_i(t)|a_i(t) = 0$ is satisfied; otherwise, the value of $\mathbf{1}$ is 0. The application of the indicator function for the Mi-BS follows the same rationale as for the Ma-BS.

In addition, the task discard penalty for the edge server is defined. If the aggregate of the transmission delay, waiting delay, and execution delay of the m_{th} task in $Q^{tr}(t)$ does not exceed δt, while that of the $(m+1)_{th}$ task is greater than δt, the tasks after the m_{th} task are discarded. In our experiments, the duration of each time slot Δt is set to $0.1s$. The corresponding penalty can be expressed as

$$\Gamma_b(t) = \{\|V_b(t)\| - m\}^+, \quad \Gamma_j(t) = \{\|V_j(t)\| - m_j\}^+, \tag{8}$$

where m and m_j represent the number of tasks in Q_b^{tr} and Q_j^{tr} that cannot be completed on time within time slice t respectively. The standby energy consumption of Ma-BS and Mi-BS is denoted by e_b^{await} and e_j^{await}, respectively. The task execution energy consumption for the two base stations can be expressed as:

$$\begin{cases} e_b^{\text{exe}}(t) = \sum\limits_{i=1}^{n} k_b \cdot \int_{\Delta t_i} \left| f_b(t) \cdot \mathbf{1}_{a_i(t)=0} \right|^2 dt, \\ e_j^{\text{exe}}(t) = \sum\limits_{i=1}^{n} k_j \cdot \int_{\Delta t_i} \left| f_j(t) \cdot \mathbf{1}_{a_i(t)=j} \right|^2 dt, \end{cases} \tag{9}$$

where Δt_i is the task execution latency of EU i, and k_b and k_j are the energy consumption factors relevant to the chip structures of Ma-BS and Mi-BS, respectively.

Based on the above, the energy queue dynamics for Ma-BS and Mi-BS can be unified as:

$$Q_b^e(t+1) = \min\left\{ \left[Q_b^e(t) - e_b^{\text{exe}}(t) - e_b^{\text{await}}\right]^+ + e_b^h(t), E_b^{\max} \right\}, \tag{10}$$

$$Q_j^e(t+1) = \min\left\{ \left[Q_j^e(t) - e_j^{\text{exe}}(t) - e_j^{\text{await}}\right]^+ + e_j^h(t), E_j^{\max} \right\}. \tag{11}$$

Here, $e_b^h(t)$ and $e_j^h(t)$ denote the energy harvested by the Ma-BS and Mi-BS at time t, while $E_b^{\max}$ and $E_j^{\max}$ represent the maximum energy capacities of the two base stations, respectively.

4. Task offloading and resource allocation scheme based on Stackelberg

In this section, the task offloading and resource allocation problem for Ma-BSs, Mi-BSs and EUs based on Stackelberg is first introduced. This optimization problem is then transformed into a MDP problem (see Table 2).

4.1. Problem formulation

To enable the system to support efficient computation, storage and location awareness, co-existence and collaboration among various computation nodes has been an innovative computing paradigm. As

Table 2
Definition of notations.

Symbol	Definition
$u(t)$	Leader's computing resource pricing strategy
$a(t)$	Task offloading strategies of EUs
$G^L[u(t), a(t)]$	Leader's utility function at time t
$G_i^F[u(t), a(t)]$	Followers' utility function at time t
$\omega_1, \omega_2, \omega_3, \omega_4$	Weight coefficients for the leader's utility components
$\lambda_1, \lambda_2, \lambda_3$	Weight coefficients for the followers' cost components

shown in Fig. 1, multiple BSs with different computational capabilities, location information, and energy harvesting characteristics are deployed around the EUs. The tasks generated by the EUs have different priorities, computational requirements, and latency constraints. Therefore, efficient collaboration between Ma-BS and Mi-BSs is important to improve the task offloading efficiency of heterogeneous MEC systems. EUs dynamically select the nearest BS or the BS with the strongest channel gain to offload tasks, based on real-time network conditions. This enables efficient resource utilization and reduces transmission overhead.

In order to promote the long-term network benefits of the cell, all heterogeneous MEC servers (Ma-BS or Mi-BSs) cooperatively manage the computational resources based on their observed information about the environment and the EUs, and together provide services to the EUs. Meanwhile, the EUs can automatically choose which server to offload tasks to based on the observed information. Thus, the problem can be formulated as a two-stage Stackelberg game where the servers is the leader and the EUs are the followers.

To better quantify the leader's benefits, denote the leader's utility function $G^L(t)$ as

$$\begin{aligned} G^L[u(t), a(t)] &= \omega_1 \cdot (\sum_{i=1}^{n} u_b(t) \cdot q_i(t) \cdot c_i(t) \cdot \mathbf{1}_{a_i(t)=0} + \\ &\sum_{i=1}^{n}\sum_{j=1}^{m} u_j(t) \cdot q_i(t) \cdot c_i(t) \cdot \mathbf{1}_{a_i(t)=j}) - \omega_2 \cdot (\Gamma_b(t) + \sum_{j=1}^{m} \Gamma_j(t)) \\ &- \omega_3 \cdot (e_b^{exe}(t) + \sum_{j=1}^{m} e_j^{exe}(t)) - \omega_4 \cdot (\sigma_b^{exe}(t) + \sum_{j=1}^{m} \sigma_j^{exe}(t)), \end{aligned} \tag{12}$$

where $u(t) = \{u_b(t), u_1(t), \ldots, u_j(t), \ldots, u_m(t)\}$. The $u(t)$ and $a(t)$ are the leader's computing resource pricing strategy and the task offloading strategies.

The leader's utility function in time slice t consists of four main components: the EUs' resource payment, the task discarding penalty, the task execution energy consumption, and the exceeding tolerance delay penalty. Besides, w_1, w_2, w_3 and w_4 are the weight coefficients of the corresponding components.

The utility function of each follower is expressed as its expense, such as resource payment, delay overhead and transmission energy consumption. It is worth stating that since the penalty for exceeding the tolerated delay is considered in the utility function of the leader, it is not considered from the followers' point of view. In order to more visually compare the reward values of the leader and the follower, the cost function of the follower is deformed so that the result is set to a positive value. The utility function of each follower is expressed by

$$\begin{aligned} G_i^F[u(t), a(t)] &= \lambda_1 \cdot (u_b(t) \cdot q_i(t) \cdot c_i(t) \cdot \mathbf{1}_{a_i(t)=0} + \sum_{i=1}^{m} u_j(t) \\ &\cdot q_i(t) \cdot c_i(t) \cdot \mathbf{1}_{a_i(t)=j}) + \lambda_2 \cdot ((d_{i,b}^{tr}(t) + d_{i,b}(t)) \cdot \mathbf{1}_{a_i(t)=0} + \\ &\sum_{i=1}^{m} (d_{i,j}^{tr}(t) + d_{i,j}(t)) \cdot \mathbf{1}_{a_i(t)=j}) + \lambda_3 \cdot (e_{i,b}^{tr} \cdot \mathbf{1}_{a_i(t)=0} + \sum_{i=1}^{m} e_{i,j}^{tr} \\ &\cdot \mathbf{1}_{a_i(t)=j}) + \xi, \end{aligned} \tag{13}$$

where the weighting coefficients are constants. The λ_1, λ_2, λ_3 are negative and ξ is positive.

At each time slot, the ability of Ma-BS and Mi-BS to consume energy is limited by the current energy queue level. In addition, to develop resource pricing strategies and task offloading decisions with a long-term perspective, based on Eqs. (12) and (13), we state the optimal problem as maximizing the leader's mean long-run aggregate returns $\hat{G^L}$ and minimizing the followers' average long-term cumulative overhead $\hat{G_i^F}$ as follows.

$$\mathrm{P1}: \max \hat{G^L} = \frac{1}{T}\sum_{t=1}^{T} G^L(t), \tag{14}$$

$$\min \hat{G_i^F} = \frac{1}{T}\sum_{t=1}^{T} G_i^F(t), \tag{15}$$

$$\text{s.t.} \quad a_i(t) \in \{0, 1, \ldots, n\}, \tag{16}$$

$$e_b^{exe}(t) \le Q_b^e(t), \quad e_j^{exe}(t) \le Q_j^e(t), \tag{17}$$

where the constraint Eq. (16) indicates that the offloading decision of the EUs may be either Ma-BS or Mi-BS within a time slot. Constraint (17) denotes that the task execution energy consumption of Ma-BSs and Mi-BSs cannot exceed the energy level of the current time slot.

The above optimization problem aims to maximize the utility of the leader (BSs) and minimize the costs of the followers (EUs). Since resource pricing, offloading decisions, and communication are coupled with each other, the computational complexity increases dramatically as the number of EUs increases. Traditional optimization methods are difficult to solve this problem quickly, so a deep reinforcement learning (DRL) approach is used to deal with the proposed resource pricing and task offloading problem.

4.2. Optimizing problems with MDP conversion

To solve the formulated problem P1 using DRL, it is essential to transform it into the standard form of MDP.

We use the MADDPG method to train the model. The training process is conducted offline at the BSs with sufficient computational resources. Then, the optimal offloading and resource allocation strategies are obtained based on the trained model. The leader and each follower act as an agent for reaching the desired goal in the complex competitive and cooperative environment. During each stage, by observing and reacting to the environment, each agent searches for the best solution of the problem. In the following, key components of this transformation are described, including the player set, the environment space set, the observation space set, the action space set, and the reward function [33].

4.2.1. Player set

The player set containing $n + 1$ elements is denoted as $\boldsymbol{P} \triangleq \{L, F_1, F_2, \ldots, F_n\}$, where L denotes the server player (leader) agent and F_i denotes the ith EUs player (follower).

4.2.2. Environment state space

The state of the environment for each time slice consists of the following main components. The first is the state of the leader. In each time slice, the environment is changing in real time. Therefore. The leader's state consists of the available energy levels of Ma-BSs $Q_b^e(t)$ and Mi-BSs $Q_j^e(t)$ ($j \in M$) and the energy harvested by Ma-BSs $e_b^h(t)$ and Mi-BSs $e_j^h(t)$ ($j \in M$) in each time slice. The second is the state of the follower. For each EU, its state is mainly its location information. Denote the network coverage information of each EU by $N_i(t)$. $N_i(t) = \{m, n\}$ denotes that EU i is within the coverage range of the Ma-BS, Mi-BS m and Mi-BS n. The last component is the channel state, which mainly consists of the channel gains of Ma-BS $g_b(t)$ and Mi-BSs $g_j(t)$ ($j \in M$).

Thus, at time slice t, the environmental state space $s(t)$ can be represented as

$$\begin{aligned} s(t) \triangleq \{ & Q_b^e(t), Q_1^e(t), \dots, Q_j^e(t), \dots, Q_m^e(t), \\ & e_b^h(t), e_1^h(t), \dots, e_j^h(t), \dots, e_m^h(t), \\ & N_1(t), \dots, N_i(t), \dots, N_n(t), \\ & g_b(t), g_1(t), \dots, g_j(t), \dots, g_m(t)\} \end{aligned} \tag{18}$$

4.2.3. Observation state space

The observation state space reflects the strategically important information in the environment that each agent needs to have to make decisions. At the start, the observation state space of the leader agent O^L consists of the current energy levels of the Ma-BS and Mi-BSs as well as the location information of each EU, which can be represented as

$$\begin{aligned} o^L(t) \triangleq \{ & Q_b^e(t), Q_1^e(t), \dots, Q_j^e(t), \dots, Q_m^e(t), \\ & N_1(t), \dots, N_i(t), \dots, N_n(t)\}. \end{aligned} \tag{19}$$

The observation state space of the ith follower agent $o_i^F(t) \in O_i^F(t)$ is its location information, denoted as

$$o_i^F(t) \triangleq N_i(t). \tag{20}$$

The observation state space of the ith follower agent $o_i^F(t)$ includes its location $N_i(t)$ and the channel gains $g_b(t), g_i(t)$ of all BSs within its coverage. This allows EUs to proactively assess BS suitability without additional coordination.

4.2.4. Action space

The action state space contains all the decisions that each agent may take. In time slice t, the leader agent sets the selling price of the resource for each Ma-BS and Mi-BS, and its action state space can be characterized as

$$a^L(t) \triangleq u(t) = \{u_b(t), u_1(t), \dots, u_j(t), \dots, u_m(t)\}. \tag{21}$$

Then, each follower makes a decision to offload the task based on the price set by the leader and its own location information. The action state space of the follower i can be expressed as

$$a_i(t) \triangleq \{0, 1, \dots, j, \dots, m\}. \tag{22}$$

4.2.5. Reward function

After executing an action, every agent gets an instant return that measures the impact of the action taken by the agent in the current environment $s(t)$. Subsequently, each agent will reach the next environment state and update its strategy based on the reward received to guide to an optimal state. Based on Eqs. (12) and (13), the reward functions for the leader r^L and the follower r_i^F in time slice t are defined respectively as

$$r^L[s(t), u(t)] = G^L[u(t), a(t)], \tag{23}$$

$$r_i^F[s(t), a(t)] = G_i^F[u(t), a(t)]. \tag{24}$$

The leader agent aims to maximize the comprehensive revenue of all servers, while all follower agents aim to minimize their own costs. To obtain the optimal resource pricing strategy and offloading strategy, this paper introduces a task offloading algorithm which combines the Stackelberg game and MADDPG (TO-SG-MADDPG). The algorithm is presented in the next section.

5. Solutions for Stackelberg game

In this section, the MADDPG-based task offloading mechanism is introduced. Next, the workflow of the MADDPG algorithm is described, and then the proposed TO-SG-MADDPG algorithm is introduced and analyzed. Our experiments are based on a simulated residential dense environment, ensuring that the evaluation closely reflects real-world scenarios.

5.1. Workflow of MADDPG algorithm

The proposed task offloading system contains a leader agent used for edge server co-computation and n EU agents. The deep deterministic policy gradient (DDPG) algorithm of a single agent only considers the states it observes [34]. Whereas, in the proposed system, the policies made by each agent interact with each other, e.g., when an EU offloads a task to a certain Ma-BS/Mi-BS, the computational resources of this BS will be reduced. Then, other EUs might select to offload the task to a Ma-BS/adjacent Mi-BS with sufficient computational resources. In addition, the status of each agent in the traditional MADDPG algorithm is parallel, which cannot reflect the sequential decision order of the leader and the follower, so this paper incorporates the Stackelberg game into the MADDPG algorithm.

MADDPG is a multi-agent reinforcement learning algorithm which is an extension of DDPG. It is specifically designed to solve the problem of collaborative decision making by multiple agents [35]. MADDPG is based on the actor–critic algorithm. Each agent has its own actor network and critic network with parameters θ_i^μ and θ_i^Q, respectively, to obtain resource pricing and task offloading strategies through the DDPG algorithm. The actor network makes action decisions based on the agent's state, while the critic network is responsible for evaluating the goodness of the agent's actions. To improve the stability of the algorithm, each agent also has a target actor network with parameter $\theta_i^{\mu'}$ and a target critic network with parameter $\theta_i^{Q'}$. To enhance the efficiency and robustness of training, the MADDPG algorithm utilizes an experience replay buffer mechanism to store previous experiences observed by agents in the environment, including information such as states, actions and rewards. In each training period, the agents are trained with random samples from the experience replay buffer to minimize the correlation between the samples. Aiming to be flexible in dynamically changing multi-agent environments, the MADDPG mechanism improves the elements of the experience replay buffer compared to the DDPG algorithm. Each agent's transition also contains the states and actions of other agents. The workflow of the MADDPG algorithm is described as follows.

Each agent possesses four deep neural networks: current actor network, current critic network, and the corresponding target actor network and target critic network, respectively. The latter two are used to guide the stability and convergence of the learning process. To begin with, a centralized joint state-value function $Q_i(s, a)$ is defined for all agents. During training, the critic network evaluates the current state and actions based on global information, and simultaneously adjusts the strategies of the actor network. The actor network then makes the optimal strategy $a_i(t) = \mu(s_i(t), \theta_i^\mu)$ based on local observations. At each iteration, the agents update the parameters of the actor network and the critical network by randomly sampling mini-batch from the experience replay buffer. For the actor network, its parameters are updated by minimizing the gradient of the agent policy, which is given by

$$\begin{aligned} \nabla_{\theta_i^\mu} J(\mu_i) = \frac{1}{\chi} \sum_j \nabla_{\theta_i^\mu} \mu_i(o_i^j) \nabla_{a_i} Q_i^\mu(s^j, a_1^j, \dots, a_i^j, \dots, \\ a_{N+1}^j)|_{a_i = \mu_i(o_i^j)}, \end{aligned} \tag{25}$$

where j is the index of the sample and χ is the size of the mini-batch. In addition, Q_i^μ denotes the state–action value function used by the ith agent as centralized training.

Different from the actor network, the parameters of the critic network are updated by minimizing the mean-square error loss function, which is denoted as

$$\begin{aligned} L(\theta_i^Q) = \frac{1}{\chi} \sum_j (y^j - Q_i^\mu(s^j, a_1^j, \dots, a_{N+1}^j))^2, \\ y^j = r_i^j + \gamma Q_i^{\mu'}(s'^j, a_1', \dots, a_{N+1}')|_{a_k' = \mu_k'(o_k^j)}, \end{aligned} \tag{26}$$

where y^j denotes the value of target Q and γ is the discount factor. $Q_i^{\mu'}(s'^j, a_1', \dots, a_{N+1}')|_{a_k' = \mu_k'(o_k^j)}$ is the state–action value at the next time step.

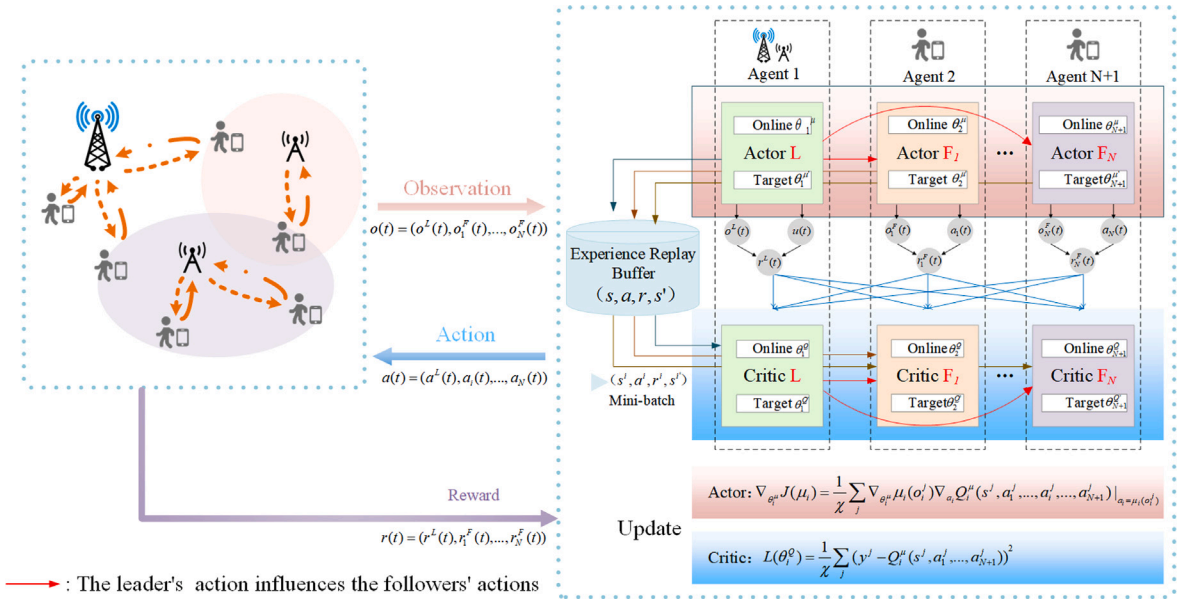

Fig. 2. Framework for TO-SG-MADDPG algorithm in the proposed MEC system.

Finally, the parameters of the target network are updated by soft update. The update formula for the target actor network and the target critic network is given by

$$\theta_i^{\mu\prime} \leftarrow \tau\theta_i^{\mu} + (1-\tau)\theta_i^{\mu\prime}, \tag{27}$$

$$\theta_i^{Q\prime} \leftarrow \tau\theta_i^{Q} + (1-\tau)\theta_i^{Q\prime}, \tag{28}$$

where the soft update parameter is expressed as $\tau \in [0, 1]$.

5.2. TO-SG-MADDPG algorithm

To solve the game problem in the proposed model, it is feasible to centrally train service provider agent and user agents using the MADDPG algorithm based on the Stackelberg game (TO-SG-MADDPG). For a clearer understanding, Fig. 2 shows the framework of the algorithm. Agents can update their strategies independently based on their reward function, actor network and critic network. Different from the traditional MADDPG algorithm, during the training of the TO-SG-MADDPG, the leader (L) first formulates a strategy according to observation. Each follower agent ($F_i, i \in N$) receive the leader's strategy and subsequently update their own strategies. Resource pricing and task offloading strategies can be extracted from the model after the model has been well-trained.

The TO-SG-MADDPG algorithm for the optimization problem of the model is concluded in Algorithm 1, where T time slots are included in each episode.

Next, the computational complexity of TO-SG-MADDPG algorithm is analyzed. The computational complexity of the training process is primarily associated with the layers of the deep neural network (DNN) utilized by each agent and the number of neurons in each layer. Specifically, let S^L and ζ_s^L denote the number of layers in the leader's DNN and the number of neurons in sth layer. S_i^F and $\zeta_{i,s}^F$ are the number of layers in the follower i's DNN and the number of neurons in sth layer, respectively. In a single training step, the computational complexity of the DNN is $\mathcal{O}(N_D(\sum_{s=1}^{S^L-1}\zeta_s^L\zeta_{s+1}^L) + \sum_{i=1}^{n} N_D(\sum_{s=1}^{S_i^F}\zeta_{i,s}^F\zeta_{i,s+1}^F))$, where N_D is the mini-batch size [36]. Since the TO-SG-MADDPG algorithm is completed by a finite number of DNNs and each agent only needs to deal with its own strategy. Therefore, the computational complexity of the leader and follower is $\mathcal{O}(\sum_{s=1}^{S^L-1}\zeta_s^L\zeta_{s+1}^L)$ and $\mathcal{O}(\sum_{s=1}^{S_i^F}\zeta_{i,s}^F\zeta_{i,s+1}^F)$, respectively.

Algorithm 1: TO-SG-MADDPG Algorithm for Resource Pricing and Task Offloading

Input: Number of agents, actor network architecture, critical network architecture
Output: Parameters of the target network
Initialize the parameters of actor network θ^{μ} and critic network θ^{Q}, the parameters of target actor network $\theta^{\mu'}$ and critic network $\theta^{Q'}$;
Initialize the experience replay buffer D for each agent;
for *each episode* **do**
  Initialize the initial state $o^L(t)$ for Ma-BSs and Mi-BSs, and the initial state $o_i^F(t)$ for each EU;
  Obtain the environment initial state space $s(t)$;
  for *each time slot* $t = 1,2,...,T$ **do**
    For agent L, select action $a^L(t) = \mu_{\theta^L}(o^L(t))$;
    For each agent F_i, select action $a_i(t) = \mu_{\theta_i^F}(o_i^F(t))$;
    Execute action $a(t) = (a^L(t), a_1(t), ..., a_N(t))$, obtain its reward $r^L(t)$, $r_i(t)$ and next observation $o(t+1)$;
    for *each agent* $i \in \{L, F_1, ..., F_N\}$ **do**
      Randomly sample a minibatch of χ samples from the replay buffer D;
      Set $y^j = r_i^j + \gamma Q_i^{\mu'}(s'^j, a_1', ..., a_{N+1}')|_{a_k' = \mu_k'(o_k^j)}$;
      Update critic's online network through minimizing the loss $L(\theta_i^Q) = \frac{1}{\chi}\sum_j (y^j - Q_i^{\mu}(s^j, a_1^j, ..., a_{N+1}^j))^2$;
      Update actor's online network utilizing the policy gradient according to Eq. (25);
    end
    Update target network parameters for each agent based on Eqs. (27) and (28).
  end
end

6. Performance evaluation

In this section, the effectiveness of the TO-SG-MADDPG algorithm is evaluated through simulation. we evaluate the effectiveness of the TO-SG-MADDPG algorithm through simulation, aiming to approximate

Table 3
Configurations of simulation parameters.

Parameters	Value
The Channel Bandwidth of Ma-BS	80 MHz
The Channel Bandwidth of Mi-BS	40 MHz
The network coverage radius of Ma-BS	40
The network coverage radius of Mi-BS	20
Maximum available energy for Ma-BS	1×10^3 mJ
Maximum available energy for Mi-BS	1×10^3 mJ
The maximum delay tolerance of EU i	[0.002, 0.01] s
The size of the experience buffer	20 000
Learning rate for action and value networks	0.0001
Activation function for action and value networks	Relu

a realistic multi-user MEC scenario. Firstly, the parameter settings of the experiments are introduced, and then 4 sets of experiments are designed to verify the convergence as well as the performance of the algorithm.

6.1. Experimental setup

Python and TensorFlow are utilized to perform the TO-SG-MADDPG algorithm for resource pricing and task offloading. Consider a dynamic time-varying multi-resident MEC system with randomly distributed 1 Ma-BS and $m = 2$ Mi-BSs. The bandwidth of the Ma-BS is 80 MHz and the bandwidth of each Mi-BS is 40 MHz. The network coverage radius γ_0, γ_j of the two types of BSs are 40 and 20, respectively. The $n = 6$ EUs are randomly distributed around the BSs. While the simulation setup is relatively small in scale, it is designed to clearly validate the core mechanisms of the algorithm. With its distributed decision-making structure and DRL-based framework, the algorithm can efficiently scale to larger MEC systems. Assume that the energy collected by Ma-BS in each time slot is a random number obeying a uniform distribution within [20, 80] mW. The energy collected by Mi-BSs obeys a uniform distribution of [10, 40] mW [19,37]. Besides, the channel gain of the wireless network of Ma-BS is randomly selected within a set {−15, −12, −9, −7, −5} dB. The set interval of the channel gain of Mi-BSs is {−10, −8, −5, −3, −2} dB. The parameters in the utility function of the leader are set as $\omega_1 = 0.5$, $\omega_2 = 0.2$, $\omega_3 = 0.1$, and $\omega_4 = 0.2$. The setting of parameters in the cost function for each follower are $\lambda_1 = 0.6$, $\lambda_2 = 0.2$, and $\lambda_3 = 0.2$. These values were chosen based on the relative importance of each component in practical edge computing scenarios, ensuring a balanced trade-off between cost, task execution efficiency, and system latency. Both actor network and critical network in our proposed TO-SG-MADDPG algorithm are four-layer fully connected neural networks. The ReLU function is employed as the activation function for both the actor network and the critical network. The tasks in the simulation are divided into three categories: standard (40%), delay-sensitive (30%), and computation-intensive (30%).

Furthermore, the values of other important experimental parameters are demonstrated in Table 3.

6.2. Result analysis

6.2.1. Convergence experiment on reward value

In the experiments, the convergence of the proposed mechanism is verified. The arrival probability of the task for EUs is set to be $\lambda = 0.5$, and $\xi = 0.4$. Fig. 3 shows the trend of the leader and follower reward values. As can be seen, the Q-values of both the leader and follower agents gradually stabilize after approximately 250 iterations, demonstrating the algorithm's relatively fast convergence speed in MEC systems. This is because the network parameters are randomly initialized for every agent. In the early stages of training, the agents will try various strategies for different explorations, as the agents may not have learned the optimal strategy. As the training progresses, the agents gradually shift to stable strategies that can achieve high rewards.

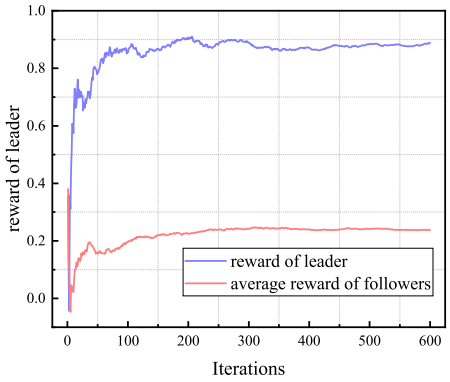

Fig. 3. Average rewards for leader and followers during the training period.

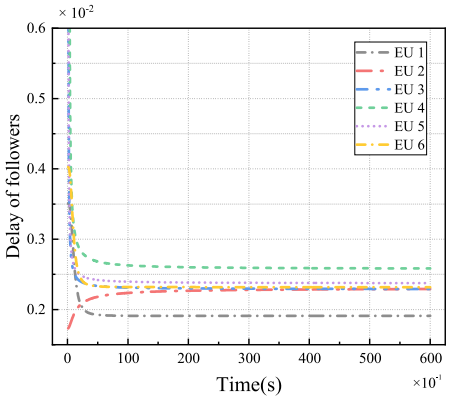

Fig. 4. Average latency trend for each EU during the training period.

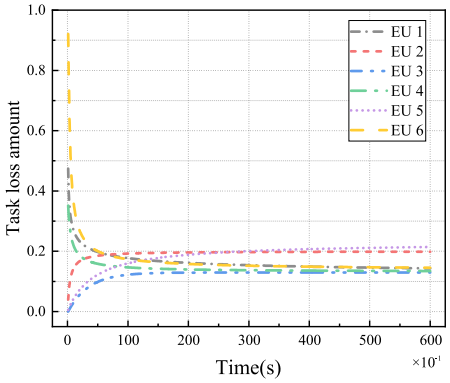

Fig. 5. Task loss amount trend for each EU during the training period.

As shown in Fig. 3, the fluctuation in reward values after convergence is minimal, indicating that the algorithm not only converges quickly but also achieves stable, reliable results, demonstrating its robustness in maintaining performance over time.

This convergence process highlights the dynamic adjustment of the leader's pricing strategy during the training phase. The leader (BSs) refines the resource pricing based on task demands and follower (EUs) responses, gradually guiding the system to an equilibrium where resource allocation and task offloading are optimized.

6.2.2. Performance experiments for each EU

Figs. 4 and 5 demonstrate the effectiveness of the algorithm for the performance at the EU side. The 6 EUs are denoted by EU 1 to EU 6. Additionally, we analyze the performance of TO-SG-MADDPG from the

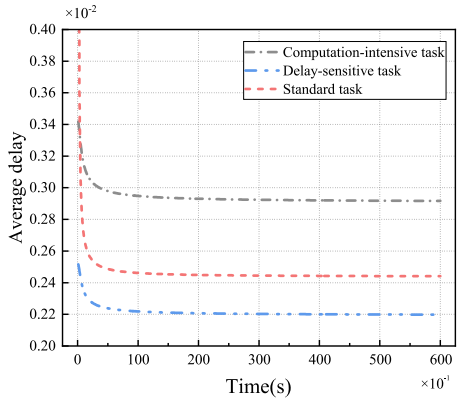

Fig. 6. The trends of average latency for different task types.

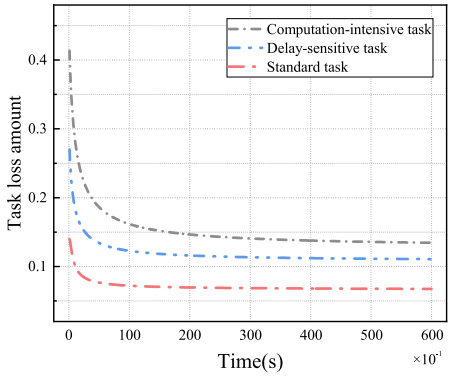

Fig. 7. The trends of task loss amount for different task types.

perspective of latency and task loss amount of the EUs. The task loss amount is calculated as the ratio of lost tasks to the total number of tasks.

Fig. 4 illustrates the variation of the latency of the six EUs with iterations. At the beginning of the iteration, the latency of all EUs except EU 2 is high due to the randomness of the initial value. Subsequently, the latency of EU 2 slowly rises while the latency of the remaining EUs decreases rapidly. This is because the leader agent allocates more resources to EU 2 during the early stage of the experiment. With the resource competition among the EUs, the Ma-BS and Mi-BSs adjust their pricing strategies and gradually allocate reasonable computational resources to the EUs. The latency of each EUs reaches stabilization when it proceeds to about the 200th generation.

Task loss amount is a direct measure of user experience and system performance, by which we can effectively evaluate the proposed algorithm. As shown in Fig. 5, the task loss shows significant fluctuations within the first 100 iterations, indicating that the resource allocation strategy is still adapting. After approximately 100 iterations, the task loss stabilizes, reflecting the fast convergence of the system. Moreover, the low fluctuation of the task loss after stabilization demonstrates the high convergence accuracy of the algorithm, ensuring stable and efficient task execution. With continued iterations, the leader agent gradually adapts its pricing strategy to allocate intelligent resources depending on task prioritization and the performance of the EUs. This makes the system gradually converge to a resource allocation with a more reasonable amount of task loss after 100 iterations, and achieves stable and efficient task execution.

6.2.3. Performance experiments on different task types

In this part, we analyze and compare the average latency, task loss amount and energy consumption of the three task types. The value

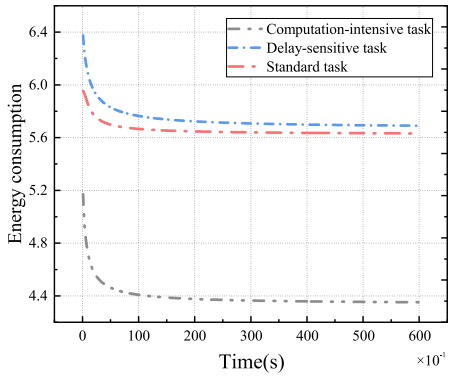

Fig. 8. The trends of energy consumption for different task types.

intervals for the size and computational density of the computationally intensive tasks are $[3, 9] \times 10^3$ and $[6, 10] \times 10^4$, respectively.

According to Fig. 6, the average latency of the three tasks gradually decreases as the number of iterations increases. This is because the agents adapt to the different task attributes by learning and adjusting so that the system can effectively satisfy the task requirements. This demonstrates the robustness of the algorithm in dynamically learning and adapting to various task demands. Under different task attributes, the latency curves show different trends. Computation-intensive tasks have the highest latency, which is due to the fact that such tasks require more computational resources and may be constrained by system resources resulting in higher latency. Standard tasks have the second highest latency and latency-sensitive tasks have the lowest latency. Latency-sensitive tasks are more likely to receive timely response and resource allocation, and thus perform better in terms of latency.

The pricing mechanism ensures that latency-sensitive tasks, which are more time-critical, are prioritized in resource allocation by dynamically adjusting the price to reflect their urgency. This enables the system to handle task heterogeneity more effectively and achieve better overall performance.

Similar to the latency experiment, Fig. 7 shows that the amount of lost tasks decreases gradually with iterations. Specifically, computation-intensive tasks have the most amount of loss, followed by delay-sensitive and standard tasks second and third, respectively. This phenomenon can be explained by the characterization of task attributes. For computation-intensive tasks, initially, the random allocation of resources may make it difficult for such tasks to obtain sufficient computational resources, which leads to higher task loss. As the system optimizes the resource allocation, the loss of computation-intensive tasks gradually decreases. In addition, since delay-sensitive tasks have high requirements on delay, the uncertainty of resources at the beginning of the experiment leads to a higher amount of task loss for such tasks. Subsequently, the system intelligently adjusts the resource allocation so that the amount of task loss gradually decreases. The observed trends further validate the effectiveness of the pricing strategy in allocating resources to tasks based on their specific requirements. By leveraging the Stackelberg game framework, the leader agent optimally prices resources to balance the task loss and system efficiency. In addition, standard tasks have relatively balanced requirements on computational resources and timeliness, and thus show a lower amount of task loss at the beginning of the experiment. As the agents learn and adjust, the amount of lost tasks for this type of task still decreases, but its reduction is relatively slow.

Fig. 8 reflects the variation in energy consumption for each type of task. The three curves in the figure all decrease and then stabilize. Latency-sensitive tasks require tasks to be completed in a shorter time. In order to ensure timeliness, the leader agent prefers to allocate more resources, resulting in higher energy consumption for this type

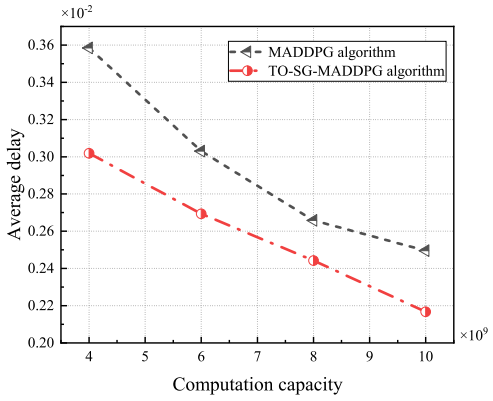

Fig. 9. The trend of average latency based on computing capacity.

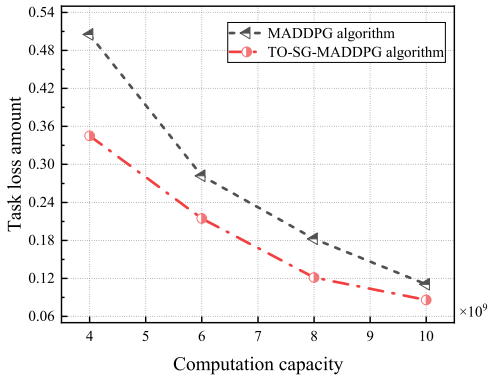

Fig. 10. The trend of task loss amount based on computing capacity.

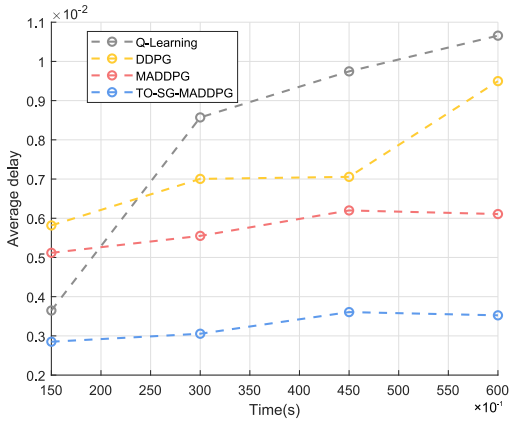

Fig. 11. Average latency comparison of different algorithms.

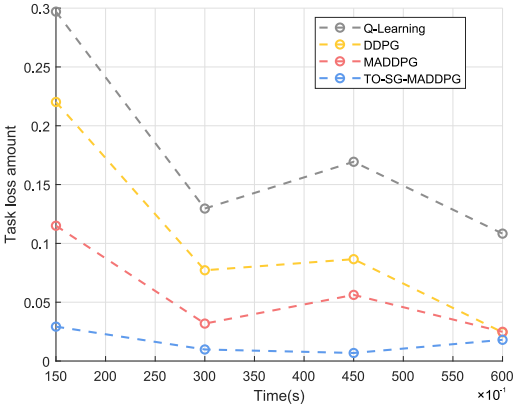

Fig. 12. Task loss amount comparisons of performance of different algorithms.

of tasks. The reason for the low energy consumption of computation-intensive tasks is related to factors such as more task loss. The leader agent prioritizes delay-sensitive tasks, resulting in the lowest energy consumption for computation-intensive tasks. This reflects the relative advantage of the proposed algorithm in adapting to delay-sensitive tasks and standard tasks.

6.2.4. Comparative experiment on computational capacity

Due to differences in application scenarios, experimental setup and optimization objectives, it is difficult to directly compare the proposed TO-SG-MADDPG algorithm with other more advanced algorithms. In this part, we used the classical MADDPG algorithm as a comparison algorithm to conduct experiments on the computational capacity of Ma-BS. The MADDPG algorithm used in this experiment is a multi-agent reinforcement learning algorithm where all agents act simultaneously and learn their policies in a decentralized manner. It involves initializing actor and critic networks, selecting actions based on observations, executing actions and storing transitions, training the networks by minimizing prediction errors, and periodically updating target networks. The computational capacity of Mi-BSs is kept at one-half of that of Ma-BS.

In Fig. 9, the latency of both the MADDPG algorithm and the TO-SG-MADDPG algorithm decreases significantly as the computational capacity of the BSs increases. The increase in the computational capacity of the BSs allows the EUs to obtain more computational resources, thus reducing the latency of the task execution. Results show that the TO-SG-MADDPG algorithm has lower latency compared to the MADDPG algorithm. It is because the TO-SG-MADDPG algorithm utilizes the idea of Stackelberg game. The leader agent makes the decision first and the follower agents adjust their strategies according to the leader's decision, which can better coordinate the resource allocation of the whole system.

Fig. 10 shows that the TO-SG-MADDPG algorithm has relatively low task loss for all kinds of computational capacities. As the computational capacity of BSs increases, the gap between the MADDPG algorithm and the TO-SG-MADDPG algorithm on the amount of task loss gradually decreases. This is because both algorithms are able to allocate resources flexibly as the computational capacity increases. However, the TO-SG-MADDPG algorithm still shows better performance with less computational capacity.

6.2.5. Comparisons with state-of-the-art methods

In order to compare the performance of the proposed TO-SG-MADDPG algorithm with state-of-the-art methods, we conducted experiments evaluating its performance against several advanced reinforcement learning algorithms: MADDPG, DDPG, and Q-Learning.

MADDPG: A multi-agent reinforcement learning algorithm where all agents act simultaneously and learn their policies in a decentralized manner.

DDPG: A model-free, off-policy algorithm used in single-agent environments for continuous action spaces.

Q-Learning: A classical reinforcement learning algorithm that is widely used in single-agent settings and focuses on learning an optimal policy using value iteration.

In Fig. 11, as the number of iterations increases, the average latency for each algorithm decreases. This is due to the algorithms gradually

refining their policies as training progresses. The TO-SG-MADDPG algorithm demonstrates significantly lower latency compared to the other algorithms. This can be attributed to the Stackelberg game-based structure of TO-SG-MADDPG, where the leader agent makes decisions first, and the follower agents adjust their strategies accordingly. This approach facilitates better resource allocation and coordination, leading to reduced delays in task execution.

Fig. 12 presents the task loss amount for each algorithm. As the number of iterations increases, task loss decreases for all algorithms, reflecting better learning and resource allocation over time. Specially, the TO-SG-MADDPG algorithm consistently shows lower task loss across all stages of training.

The results from Figs. 11 and 12 illustrate that, compared to the other state-of-the-art algorithms, the TO-SG-MADDPG algorithm performs better in both minimizing average latency and reducing task loss. It shows effectiveness in resource-constrained multi-agent environments.

These comparisons further demonstrate the advantage of incorporating a pricing mechanism in the TO-SG-MADDPG algorithm. By optimizing resource pricing, the system achieves better coordination between leaders and followers, leading to more efficient task execution and lower task loss. This pricing mechanism enhances the adaptability of the algorithm to different system configurations and task demands, further strengthening its robustness.

7. Conclusions and future work

This paper designs a mobile edge computing task offloading system based on multi-base station cooperation. The system takes into account the cooperative computing capabilities of macro and micro base stations, as well as the stochastic nature of task arrival and energy collection in real application scenarios. Facing three kinds of tasks characterized by computation-intensive, delay-sensitive, and standard, a Stackelberg game is introduced to find the balance point of economic efficiency between base stations and terminal devices. By converting the original problem into a MDP, a task offloading algorithm based on multi-intelligence deep reinforcement learning is proposed. The algorithm is able to learn historical data to optimize task offloading and pricing decisions. Experiments demonstrate that the algorithm is able to reduce latency and task loss while improving system effectiveness. While the proposed model assumes negligible overhead for metadata transmission to the decision-making BS, future work will incorporate fine-grained modeling of initial coordination costs, particularly in scenarios with highly dynamic network conditions or large metadata sizes. The random distribution and mobility of EUs make resource allocation challenging. Future research will explore dynamic resource allocation strategies that consider EU mobility. Based on the real-time location of EUs, resources are flexibly allocated to further improve resource efficiency and system performance.

CRediT authorship contribution statement

Zhao Tong: Funding acquisition, Formal analysis, Data curation, Conceptualization. **Xin Deng:** Methodology, Conceptualization. **Yuanyang Zhang:** Investigation, Formal analysis. **Jing Mei:** Software, Resources. **Can Wang:** Writing – review & editing, Visualization, Supervision, Project administration, Formal analysis, Conceptualization. **Keqin Li:** Validation, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that has been used is confidential.

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: Vision and challenges, IEEE Internet Things J. 3 (5) (2016) 637–646.

[2] J. Du, Z. Kong, A. Sun, J. Kang, D. Niyato, X. Chu, F.R. Yu, MADDPG-based joint service placement and task offloading in MEC empowered air-ground integrated networks, IEEE Internet Things J. (2023).

[3] Z. Tong, F. Ye, M. Yan, H. Liu, S. Basodi, A survey on algorithms for intelligent computing and smart city applications, Big Data Min. Anal. 4 (3) (2021) 155–172.

[4] Q. Zhang, L. Cheng, R. Boutaba, Cloud computing: state-of-the-art and research challenges, J. Internet Serv. Appl. 1 (2010) 7–18.

[5] J.W. Rittinghouse, J.F. Ransome, Cloud Computing: Implementation, Management, and Security, CRC Press, 2017.

[6] K. Cao, Y. Liu, G. Meng, Q. Sun, An overview on edge computing research, IEEE Access 8 (2020) 85714–85728.

[7] Z. Tong, X. Deng, Z. Xiao, D. He, A.T. Chronopoulos, S. Dustdar, A bilateral game approach for task outsourcing in multi-access edge computing, IEEE Trans. Netw. Serv. Manag. (2023).

[8] Y. Chiang, Y. Zhang, H. Luo, T.-Y. Chen, G.-H. Chen, H.-T. Chen, Y.-J. Wang, H.-Y. Wei, C.-T. Chou, Management and orchestration of edge computing for iot: A comprehensive survey, IEEE Internet Things J. (2023).

[9] F. You, X. Yuan, W. Ni, A. Jamalipour, A novel privacy-preserving incentive mechanism for multi-access edge computing, IEEE Trans. Cogn. Commun. Netw. 10 (5) (2024) 1928–1943.

[10] P. Zhao, J. Tao, K. Lui, G. Zhang, F. Gao, Deep reinforcement learning-based joint optimization of delay and privacy in multiple-user MEC systems, IEEE Trans. Cloud Comput. 11 (2) (2023) 1487–1499.

[11] C. Dong, Y. Tian, Z. Zhou, W. Wen, X. Chen, Joint power allocation and task offloading for reliability-aware services in NOMA-enabled MEC, IEEE Trans. Wirel. Commun. (2023).

[12] B. Baek, J. Lee, Y. Peng, S. Park, Three dynamic pricing schemes for resource allocation of edge computing for IoT environment, IEEE Internet Things J. 7 (5) (2020) 4292–4303.

[13] K. Sadatdiynov, L. Cui, L. Zhang, J.Z. Huang, S. Salloum, M.S. Mahmud, A review of optimization methods for computation offloading in edge computing networks, Digit. Commun. Netw. (2022).

[14] Z. Tong, X. Deng, J. Mei, L. Dai, K. Li, K. Li, Stackelberg game-based task offloading and pricing with computing capacity constraint in mobile edge computing, J. Syst. Archit. 137 (2023) 102847.

[15] Z. Tong, Y. Zhang, J. Mei, W. Ai, K. Li, K. Li, Stackelberg game-based bandwidth allocation and resource pricing for multiuser in MEC system, IEEE Internet Things J. 11 (13) (2024) 23737–23751.

[16] A. Islam, A. Debnath, M. Ghose, S. Chakraborty, A survey on task offloading in multi-access edge computing, J. Syst. Archit. 118 (2021) 102225.

[17] Y. Mao, J. Zhang, K.B. Letaief, Joint task offloading scheduling and transmit power allocation for mobile-edge computing systems, in: 2017 IEEE Wireless Communications and Networking Conference, WCNC, 2017, pp. 1–6.

[18] Y. Chen, N. Zhang, Y. Zhang, X. Chen, W. Wu, X.S. Shen, TOFFEE: Task offloading and frequency scaling for energy efficiency of mobile devices in mobile edge computing, IEEE Trans. Cloud Comput. 9 (4) (2021) 1634–1644.

[19] Y. Mao, J. Zhang, K.B. Letaief, Dynamic computation offloading for mobile-edge computing with energy harvesting devices, IEEE J. Sel. Areas Commun. 34 (12) (2016) 3590–3605.

[20] H. Zhou, K. Jiang, X. Liu, X. Li, V.C. Leung, Deep reinforcement learning for energy-efficient computation offloading in mobile-edge computing, IEEE Internet Things J. 9 (2) (2021) 1517–1530.

[21] Z. Ji, L. Chen, N. Zhao, Y. Chen, G. Wei, F.R. Yu, Computation offloading for edge-assisted federated learning, IEEE Trans. Veh. Technol. 70 (9) (2021) 9330–9344.

[22] Y. Chen, F. Zhao, Y. Lu, X. Chen, Dynamic task offloading for mobile edge computing with hybrid energy supply, Tsinghua Sci. Technol. 28 (3) (2022) 421–432.

[23] H. Seo, H. Oh, J.K. Choi, S. Park, Differential pricing-based task offloading for delay-sensitive IoT applications in mobile edge computing system, IEEE Internet Things J. 9 (19) (2022) 19116–19131.

[24] M. Tao, X. Li, K. Ota, M. Dong, Single-cell multiuser computation offloading in dynamic pricing-aided mobile edge computing, IEEE Trans. Comput. Soc. Syst. (2023).

[25] B. Zhu, K. Chi, J. Liu, K. Yu, S. Mumtaz, Efficient offloading for minimizing task computation delay of NOMA-based multiaccess edge computing, IEEE Trans. Commun. 70 (5) (2022) 3186–3203.
[26] T. Liu, S. Sheng, L. Fang, Y. Zhang, T. Zhang, W. Tong, Latency-minimized and energy-efficient online task offloading for mobile edge computing with stochastic heterogeneous tasks, in: 2019 IEEE 25th International Conference on Parallel and Distributed Systems, ICPADS, IEEE, 2019, pp. 376–383.
[27] T. Bahreini, H. Badri, D. Grosu, Mechanisms for resource allocation and pricing in mobile edge computing systems, IEEE Trans. Parallel Distrib. Syst. 33.
[28] S. Pang, X. He, S. Yu, M. Wang, S. Qiao, H. Gui, Y. Qi, A stackelberg game scheme for pricing and task offloading based on idle node-assisted edge computational model, Simul. Model. Pr. Theory 124 (2023) 102725.
[29] G. Mitsis, E.E. Tsiropoulou, S. Papavassiliou, Price and risk awareness for data offloading decision-making in edge computing systems, IEEE Syst. J. 16 (4) (2022) 6546–6557.
[30] H. Xiao, C. Xu, Y. Ma, S. Yang, L. Zhong, G.-M. Muntean, Edge intelligence: A computational task offloading scheme for dependent IoT application, IEEE Trans. Wirel. Commun. 21 (9) (2022) 7222–7237.
[31] S. Xia, Z. Yao, Y. Li, S. Mao, Online distributed offloading and computing resource management with energy harvesting for heterogeneous MEC-enabled IoT, IEEE Trans. Wirel. Commun. 20 (10) (2021) 6743–6757.
[32] S. Xia, Z. Yao, G. Wu, Y. Li, Distributed offloading for cooperative intelligent transportation under heterogeneous networks, IEEE Trans. Intell. Transp. Syst. 23 (9) (2022) 16701–16714.
[33] X. Zhang, J. Zhang, Z. Liu, Q. Cui, X. Tao, S. Wang, MDP-based task offloading for vehicular edge computing under certain and uncertain transition probabilities, IEEE Trans. Veh. Technol. 69 (3) (2020) 3296–3309.
[34] H. Gao, X. Wang, W. Wei, A. Al-Dulaimi, Y. Xu, Com-DDPG: task offloading based on multiagent reinforcement learning for information-communication-enhanced mobile edge computing in the internet of vehicles, IEEE Trans. Veh. Technol. (2023).
[35] Z. Cheng, M. Min, M. Liwang, L. Huang, Z. Gao, Multiagent DDPG-based joint task partitioning and power control in fog computing networks, IEEE Internet Things J. 9 (1) (2021) 104–116.
[36] X. Guo, Y. Chen, Y. Wang, Learning-based robust and secure transmission for reconfigurable intelligent surface aided millimeter wave UAV communications, IEEE Wirel. Commun. Lett. 10 (8) (2021) 1795–1799.
[37] G. Zhang, W. Zhang, Y. Cao, D. Li, L. Wang, Energy-delay tradeoff for dynamic offloading in mobile-edge computing system with energy harvesting devices, IEEE Trans. Ind. Inform. 14 (10) (2018) 4642–4655.

Zhao Tong received the Ph.D. degree in computer science from Hunan University, Changsha, China, in 2014. From 2017 to 2018, he was a visiting scholar at the Georgia State University. He is currently a full professor at the College of Information Science and Engineering of Hunan Normal University. His research interests include parallel and distributed computing systems, resource management, machine learning algorithm and big data. He has published more than 25 research papers in international conferences and journals. He is a senior member of the IEEE and a senior member of the China Computer Federation (CCF).

Xin Deng received the B.S. degree in computer science and technology from Hengyang Normal University, Hengyang, China, in 2020. She is currently working toward the M.S. degree at the College of Information Science and Engineering, Hunan Normal University, Changsha, China. Her research interests focus on distributed parallel computing, modeling and resource pricing and allocation in mobile edge computing systems, and game theory.

Yuanyang Zhang received the B.S. degree in software engineering from Central South University of Forestry and Technology, Changsha, China, in 2022. She is currently pursuing a master's degree at the College of Information Science and Engineering, Hunan Normal University, located in Changsha, China. Her research interests mainly revolve around the areas of mobile edge computing and game theory.

Jing Mei received the Ph.D. in computer science from Hunan University, China, in 2015. She is currently an associate professor in the College of Information Science and Engineering at Hunan Normal University. Her research interests include parallel and distributed computing, cloud computing, etc. She has published 12 research articles in international conference and journals. She is a member of the IEEE.

Can Wang received the BE degree from Tianjin Normal University, Tianjin, China, in 2023. She is currently pursuing the M.S. degree with the College of Information Science and Engineering, Hunan Normal University, Changsha, China. Her research interests focus on mobile edge computing, task offloading and objective optimization.

Keqin Li is a SUNY Distinguished Professor of computer science with the State University of New York. He is also a Distinguished Professor at Hunan University, China. His current research interests include cloud computing, fog computing and mobile edge computing, energy-efficient computing and communication, embedded systems and cyber–physical systems, heterogeneous computing systems, high-performance computing, computer architectures and systems, CPU–GPU hybrid and cooperative computing, computer networking, machine learning, intelligent and soft computing. He has authored or coauthored over 850 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He currently serves or has served on the editorial boards of the IEEE Transactions on Parallel and Distributed Systems, the IEEE Transactions on Computers, the IEEE Transactions on Cloud Computing, the IEEE Transactions on Services Computing, and the IEEE Transactions on Sustainable Computing. He is an IEEE Fellow.